



# First Thursdays

---

May 2 - Webinar 1: Discover Renode for PolarFire® SoC Design and Debug

June 6 - Webinar 2: How to Get Started with Renode for PolarFire SoC

July 4 - Webinar 3: Learn to Debug a Bare-Metal PolarFire SoC Application with Renode

Aug. 1 - Webinar 4: Tips and Tricks for Even Easier PolarFire SoC Debug with Renode

Sept. 5 - Webinar 5: Add and Debug PolarFire SoC models with Renode

**Oct. 3 - Webinar 6: Add and Debug and Pre-Existing model in PolarFire SoC**

**Nov. 7 - Webinar 7: How to write custom models – filters, offloading, acceleration etc**

**Dec. 5 - Webinar 8: Handling Binaries**

**Contd.**



# Second Thursdays

---

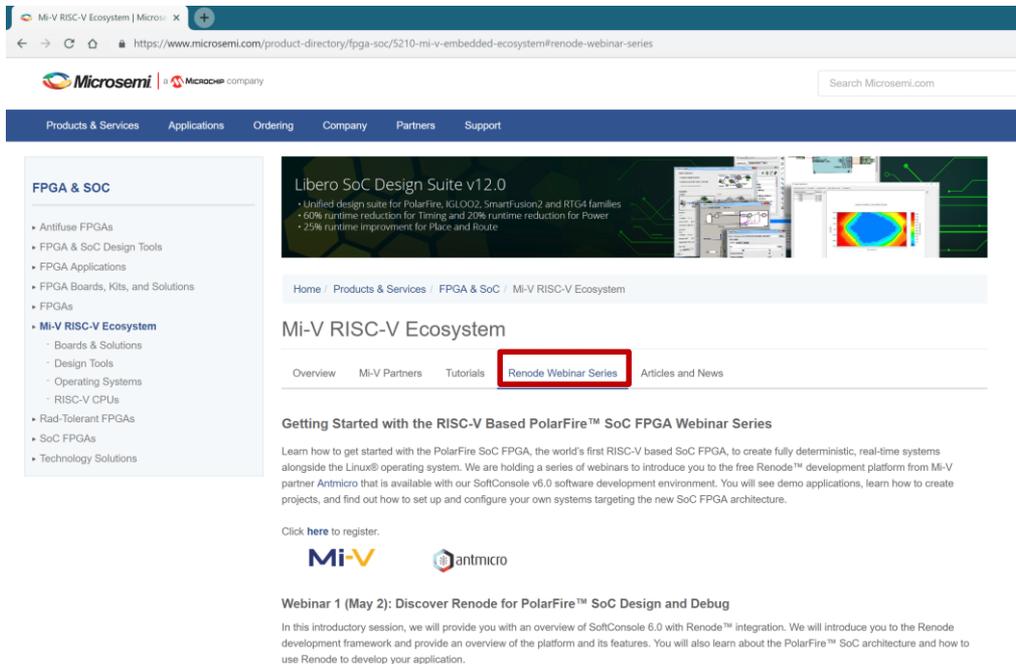
**Jan. 9 - Webinar 9: Run Linux on Renode (PolarFire SoC Model as a Quad-core SMP) – this is not a Linux / Buildroot tutorial**

**Feb. 13 - Webinar 10: Build applications for Linux on PolarFire SoC**

**Mar. 12 - Webinar 11: Introduction to PolarFire SoC MSS Configuration and Software Flow**

**Apr. 9 - Webinar 12: Two baremetal Applications on PolarFire SoC**

**May 14 - Webinar 13: Linux + Real-Time (AMP Mode) on PolarFire SoC**



The screenshot shows a web browser window with the URL <https://www.microsemi.com/product-directory/fpga-soc/5210-mi-v-embedded-ecosystem#renode-webinar-series>. The page features a navigation menu with 'Products & Services', 'Applications', 'Ordering', 'Company', 'Partners', and 'Support'. A sidebar on the left lists 'FPGA & SOC' categories, including 'Mi-V RISC-V Ecosystem'. The main content area highlights the 'Libero SoC Design Suite v12.0' and includes a breadcrumb trail: 'Home / Products & Services / FPGA & SoC / Mi-V RISC-V Ecosystem'. The page title is 'Mi-V RISC-V Ecosystem', and the 'Renode Webinar Series' link in the navigation is highlighted with a red box. Below the title, there is a section for 'Getting Started with the RISC-V Based PolarFire™ SoC FPGA Webinar Series' with a brief description and a 'Click here to register.' link. Logos for 'Mi-V' and 'antmicro' are displayed. The final section is 'Webinar 1 (May 2): Discover Renode for PolarFire™ SoC Design and Debug', with a paragraph of introductory text.

[www.microsemi.com/Mi-V](https://www.microsemi.com/Mi-V) “Renode Webinar Series”

# Introduction

---

- **Where are models in Renode**
- **Ways to add models**
- **How to add a Just In Time (JIT) compiled model**
- **Debugging a model using MonoDevelop**
- **Debugging a model using logs**

# But first!

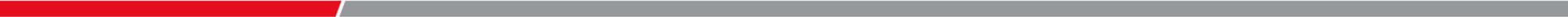
---

- **Have you watched our supporting video?**
- **A supporting video is available in the “Renode Webinar Series” page of the Mi-V ecosystem that outlines the set up steps required to use the tools shown in the webinar**





# Where are models in Renode



# Where are models in Renode

---

- Models can be added to Renode before the software is compiled
- They are then built with the software
- The software source code is available on GitHub
- <https://github.com/renode>



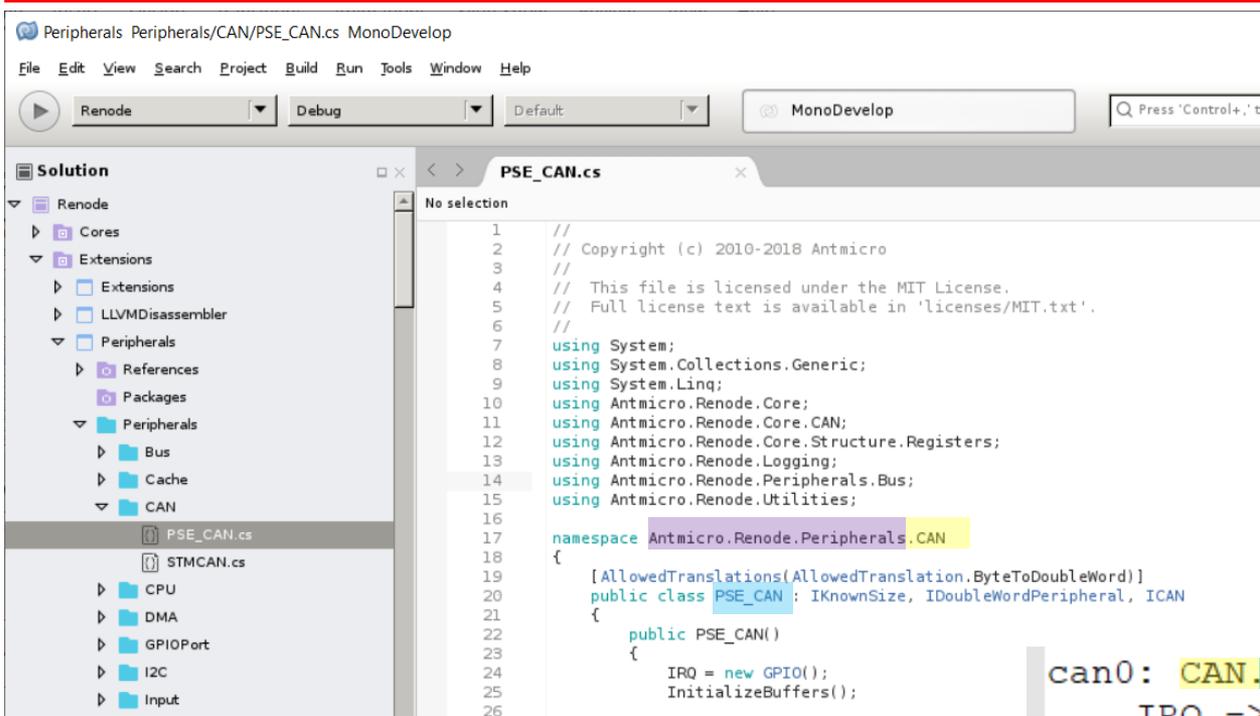
# Where are models in Renode

The screenshot shows the MonoDevelop IDE interface. The Solution Explorer on the left displays a project structure for 'Renode'. Annotations with arrows point to specific folders:

- Renode**: Points to the root project folder.
- Extensions**: Points to the 'Extensions' folder under 'Cores'.
- Peripherals**: Points to the 'Peripherals' folder under 'Extensions'.
- Peripherals**: Points to the 'Peripherals' folder under the main project.

The project structure includes folders like Cores, Extensions, LLVMDisassembler, References, Packages, CPU, DMA, GPIOPort, I2C, Input, IRQControllers, Memory, MemoryControllers, Miscellaneous, and MTD. The 'CAN' folder is expanded, showing files like PSE\_CAN.cs and STMCAN.cs.

# Where are models in Renode



Peripherals Peripherals/CAN/PSE\_CAN.cs MonoDevelop

File Edit View Search Project Build Run Tools Window Help

Renode Debug Default MonoDevelop

Solution

- Renode
  - Cores
  - Extensions
  - LLVMDisassembler
  - Peripherals
    - References
    - Packages
    - Peripherals
      - Bus
      - Cache
      - CAN
        - PSE\_CAN.cs
        - STMCAN.cs
        - CPU
        - DMA
        - GPIOPort
        - I2C
        - Input

```
1 //  
2 // Copyright (c) 2010-2018 Antmicro  
3 //  
4 // This file is licensed under the MIT License.  
5 // Full license text is available in 'licenses/MIT.txt'.  
6 //  
7 using System;  
8 using System.Collections.Generic;  
9 using System.Linq;  
10 using Antmicro.Renode.Core;  
11 using Antmicro.Renode.Core.CAN;  
12 using Antmicro.Renode.Core.Structure.Registers;  
13 using Antmicro.Renode.Logging;  
14 using Antmicro.Renode.Peripherals.Bus;  
15 using Antmicro.Renode.Utilities;  
16  
17 namespace Antmicro.Renode.Peripherals.CAN  
18 {  
19     [AllowedTranslations(AllowedTranslation.ByteToDoubleWord)]  
20     public class PSE_CAN : IKnownSize, IDoubleWordPeripheral, ICAN  
21     {  
22         public PSE_CAN()  
23         {  
24             IRQ = new GPIO();  
25             InitializeBuffers();  
26
```

can0: CAN.PSE\_CAN @ sysbus 0x2010C000  
IRQ -> plic@56

# Where are models in Renode

SC workspace.examples - Renode/platforms/cpus/polarfire.repl - Microsemi SoftConsole v6.0.0.116

File Edit Source Refactor Navigate Search Project Git Run Window Help

Project Explorer

- pse-blinky
- Renode
  - bin
  - licenses
  - platforms
  - boards
  - cpus
    - silabs
      - a20.repl
      - at91rm9200.repl
      - cc2538.repl
      - i386.repl
      - litex\_vexriscv.repl
      - miv.repl
      - mpc5567.repl
      - picosoc.repl
      - polarfire.repl
      - quark-c1000.repl
      - sifive-fe310.repl
      - sifive-fu540.repl
      - stm32f103.repl
      - stm32f4.repl
      - stm32f429.repl
      - stm32f746.repl
      - stm32l151.repl

```

53 mmuart0: UART.NS16550 @ sysbus 0x20000000
54   wideRegisters: true
55   IRQ -> plic@90
56
57 mmuart1: UART.NS16550 @ sysbus 0x20100000
58   wideRegisters: true
59   IRQ -> plic@91
60
61 mmuart2: UART.NS16550 @ sysbus 0x20102000
62   wideRegisters: true
63   IRQ -> plic@92
64
65 mmuart3: UART.NS16550 @ sysbus 0x20104000
66   wideRegisters: true
67   IRQ -> plic@93
68
69 mmuart4: UART.NS16550 @ sysbus 0x20106000
70   wideRegisters: true
71   IRQ -> plic@94
72
73 mmc: SD.PSE_SDController @ sysbus 0x20008000
74   IRQ -> plic@88
75   WakeupIRQ -> plic@89
76
77 spi0: SPI.PSE_SPI @ sysbus 0x20108000
78   IRQ -> plic@54
79
80 spi1: SPI.PSE_SPI @ sysbus 0x20109000
81   IRQ -> plic@55
  
```

ADD UART to the system

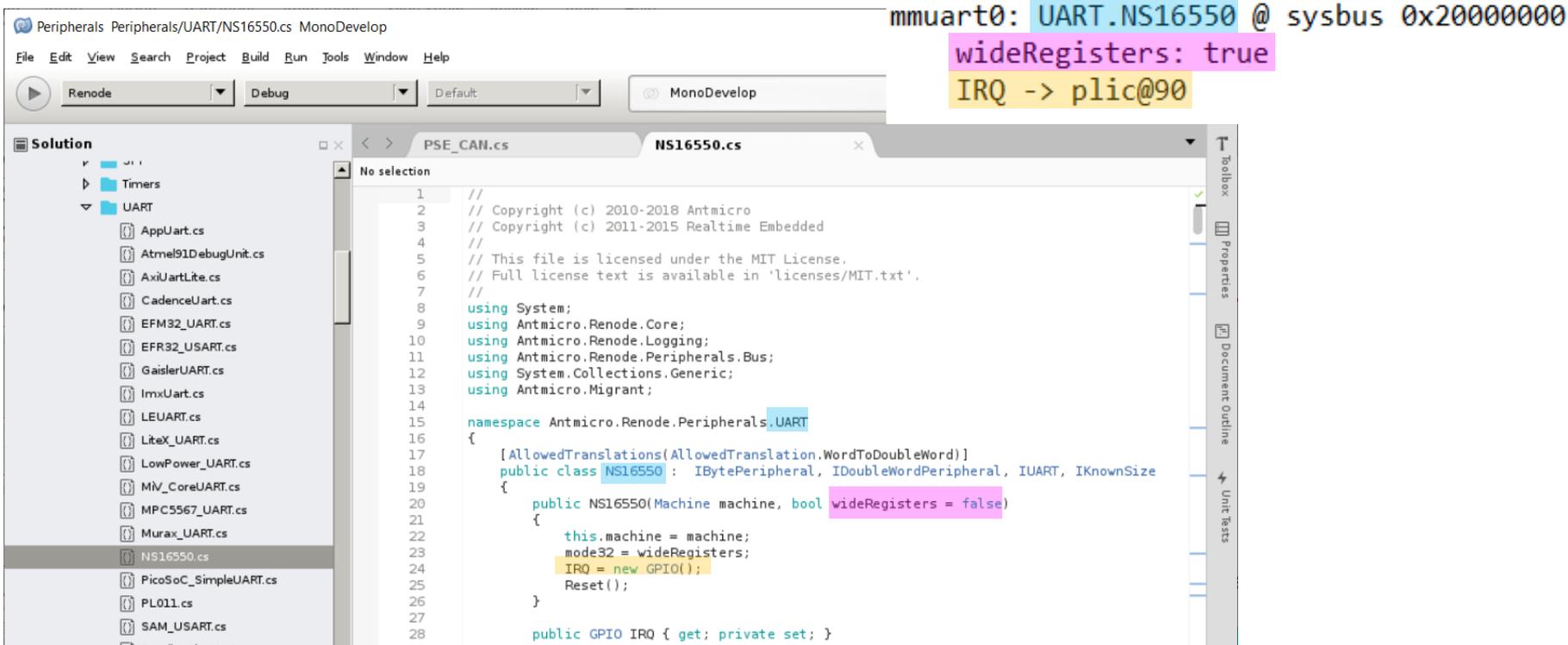
**Connecting a peripheral:**  
It depends on the model!

[name]: [namespace].[name] @ sysbus [address]  
[parameters0]  
[constructors0]  
etc.

Parameters affect the peripherals operation  
Some are optional, depending on default values

Constructors connect parts of the system together  
E.g IRQ out to PLIC

# Where are models in Renode



mmuart0: UART.NS16550 @ sysbus 0x20000000  
wideRegisters: true  
IRQ -> plic@90

```
1 //  
2 // Copyright (c) 2010-2018 Antmicro  
3 // Copyright (c) 2011-2015 Realtime Embedded  
4 //  
5 // This file is licensed under the MIT License.  
6 // Full license text is available in 'licenses/MIT.txt'.  
7 //  
8 using System;  
9 using Antmicro.Renode.Core;  
10 using Antmicro.Renode.Logging;  
11 using Antmicro.Renode.Peripherals.Bus;  
12 using System.Collections.Generic;  
13 using Antmicro.Migrant;  
14  
15 namespace Antmicro.Renode.Peripherals.UART  
16 {  
17     [AllowedTranslations(AllowedTranslation.WordToDoubleWord)]  
18     public class NS16550 : IBytePeripheral, IDoubleWordPeripheral, IUART, IKnownSize  
19     {  
20         public NS16550(Machine machine, bool wideRegisters = false)  
21         {  
22             this.machine = machine;  
23             mode32 = wideRegisters;  
24             IRQ = new GPIO();  
25             Reset();  
26         }  
27  
28         public GPIO IRQ { get; private set; }  
29     }  
30 }
```

# Where are models in Renode

```

> PSE_CAN.cs NS16550.cs
selection
1 //
2 // Copyright (c) 2010-2018 Antmicro
3 // Copyright (c) 2011-2015 Realtime Embedded
4 //
5 // This file is licensed under the MIT License.
6 // Full license text is available in 'licenses/MIT.txt'.
7 //
8 using System;
9 using Antmicro.Renode.Core;
10 using Antmicro.Renode.Logging;
11 using Antmicro.Renode.Peripherals.Bus;
12 using System.Collections.Generic;
13 using Antmicro.Migrant;
14
15 namespace Antmicro.Renode.Peripherals.UART
16 {
17     [AllowedTranslations(AllowedTranslation.WordToDoubleWord)]
18     public class NS16550 : IBytePeripheral, IDoubleWordPeripheral, IUART, IKnownSize
19     {
20         public NS16550(Machine machine, bool wideRegisters = false)
21         {
22             this.machine = machine;
23             mode32 = wideRegisters;
24             IRQ = new GPIO();
25             Reset();
26         }
27     }
28 }

```

Default value



Overwritten

Constructor

```

mmuart0: UART.NS16550 @ sysbus 0x20000000
wideRegisters: true
IRQ -> plic@90

```

```

wdog0: Timers.PSE_Watchdog @ sysbus 0x20001000
frequency: 156250
RefreshEnable -> plic@100 | e51@26
Trigger -> plic@105 | e51@25

```

```

PSE_CAN.cs NS16550.cs PSE_Watchdog.cs
E_Watchdog > DefineRegisters()
4 // This file is licensed under the MIT License.
5 // Full license text is available in 'licenses/MIT.txt'.
6 //
7 using System;
8 using Antmicro.Renode.Core;
9 using Antmicro.Renode.Peripherals.Bus;
10 using Antmicro.Renode.Core.Structure.Registers;
11 using Antmicro.Renode.Time;
12 using Antmicro.Renode.Logging;
13
14 namespace Antmicro.Renode.Peripherals.Timers
15 {
16     public class PSE_Watchdog : BasicDoubleWordPeripheral, IKnownSize
17     {
18         public PSE_Watchdog(Machine machine, long frequency) : base(machine)
19         {
20             internalTimer = new LimitTimer(machine.ClockSource, frequency, this, String.Empty, T
21             internalTimer.LimitReached += TimerLimitReached;
22
23             RefreshEnable = new GPIO();
24             Trigger = new GPIO();
25         }
26
27         public override void Reset()
28         {
29             base.Reset();
30             Trigger.Unset();
31             RefreshEnable.Unset();
32             state = State.ForbiddenRegion;
33             internalTimer.Reset();
34         }
35
36         public long Size => 0x1000;
37     }
38 }

```

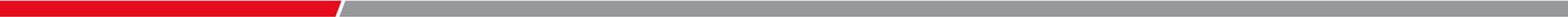
No default value – must be provided





**MICROCHIP**

# Ways To Add models



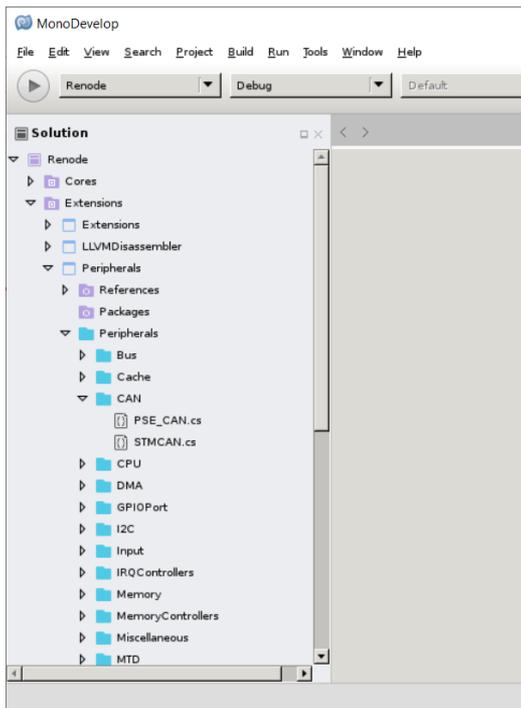
# Ways To Add models

---

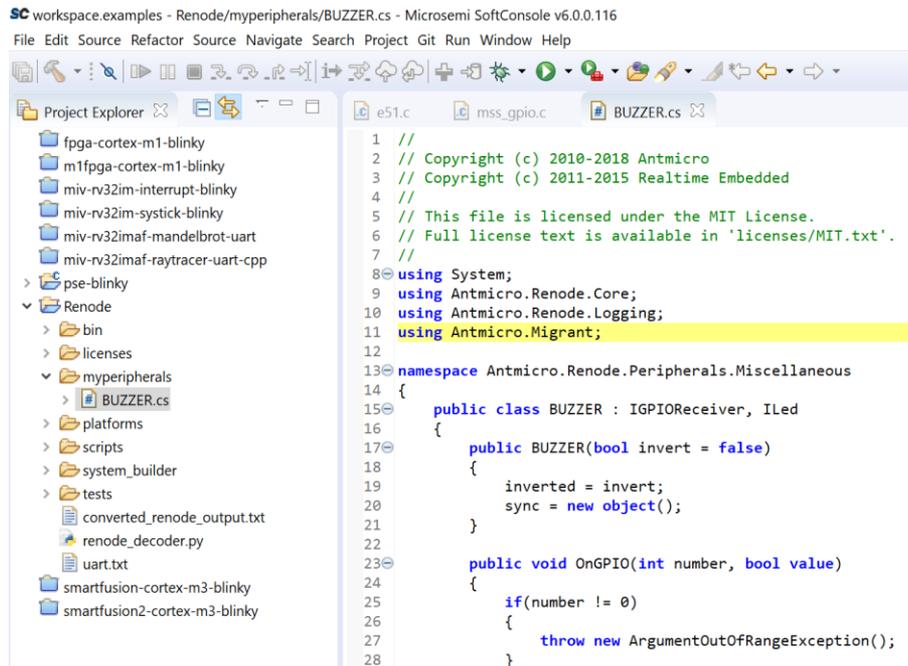
| Pre-compiled                                | JiT (Just in Time compiled)                              |
|---|--|
| Built in with Renode                        | Standalone file separate to Renode                       |
| Run faster                                  | Can be modified without having to re-build               |
| Can't be edited without rebuilding Renode   | Run slower than pre-compiled                             |
| Models included with Renode are precompiled | Develop models using JiT and then build them into Renode |
| Available on Windows and Linux              | Only available on Linux                                  |

# Ways To Add models

## Pre-compiled

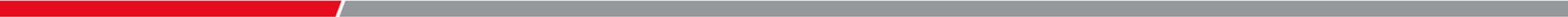


## JiT





# **How to add a Just In Time (JIT) compiled model**



# How to add a Just In Time (JIT) compiled model

---

- **When to use @ in Renode:**
  - If using a relative or full path:
    - E.g “include @../.../scripts/single.....”
    - E.g “include @C:/Microsemi/SoftConsole\_v6.0/....”
  - Not needed if using CWD:
    - E.g “include \$CWD/..../scripts/single....”

# How to add a Just In Time (JIT) compiled model

---

- **Include the C# file for the model**
  - include @[path\_to\_file]
- **Add the model to the system**
  - machine LoadPlatformDescriptionFromString  
“[sysbus\_name]: [class].[name] @ sysbus [address]”

```
polarfire.repl ✕  
53 mmuart0: UART.NS16550 @ sysbus 0x20000000  
54   wideRegisters: true  
55   IRQ -> plic@90  
56
```

[sysbus\_name]: [namespace].[name] @ sysbus [address]

# How to add a Just In Time (JIT) compiled model

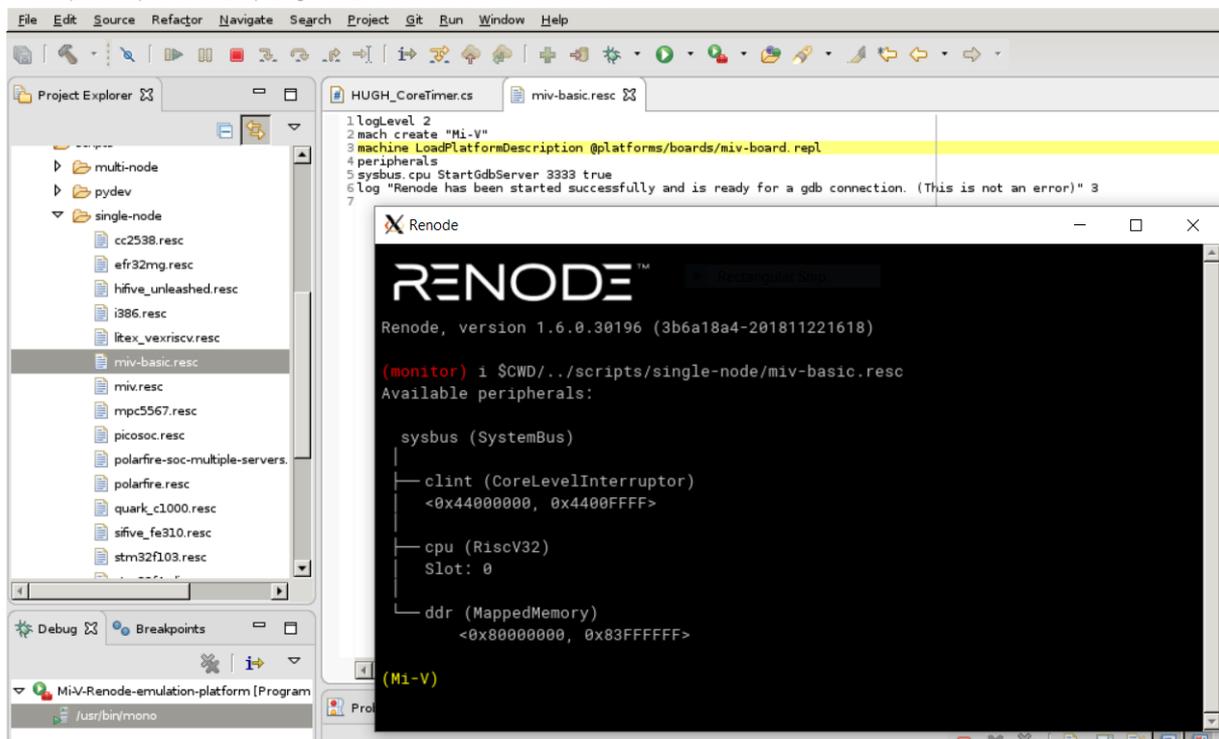
---

- **Commands used:**

1. `include $CWD/../../My_models/HUGH_CoreTimer.cs`
2. `machine LoadPlatformDescriptionFromString  
"Hugh_timer: Timers.Hugh_CoreTimer @sysbus  
0x70000000"`

# How to add a Just In Time (JIT) compiled model

SC workspace.examples - Renode/scripts/single-node/miv-basic.resc - Microsemi SoftConsole v6.0.0.116



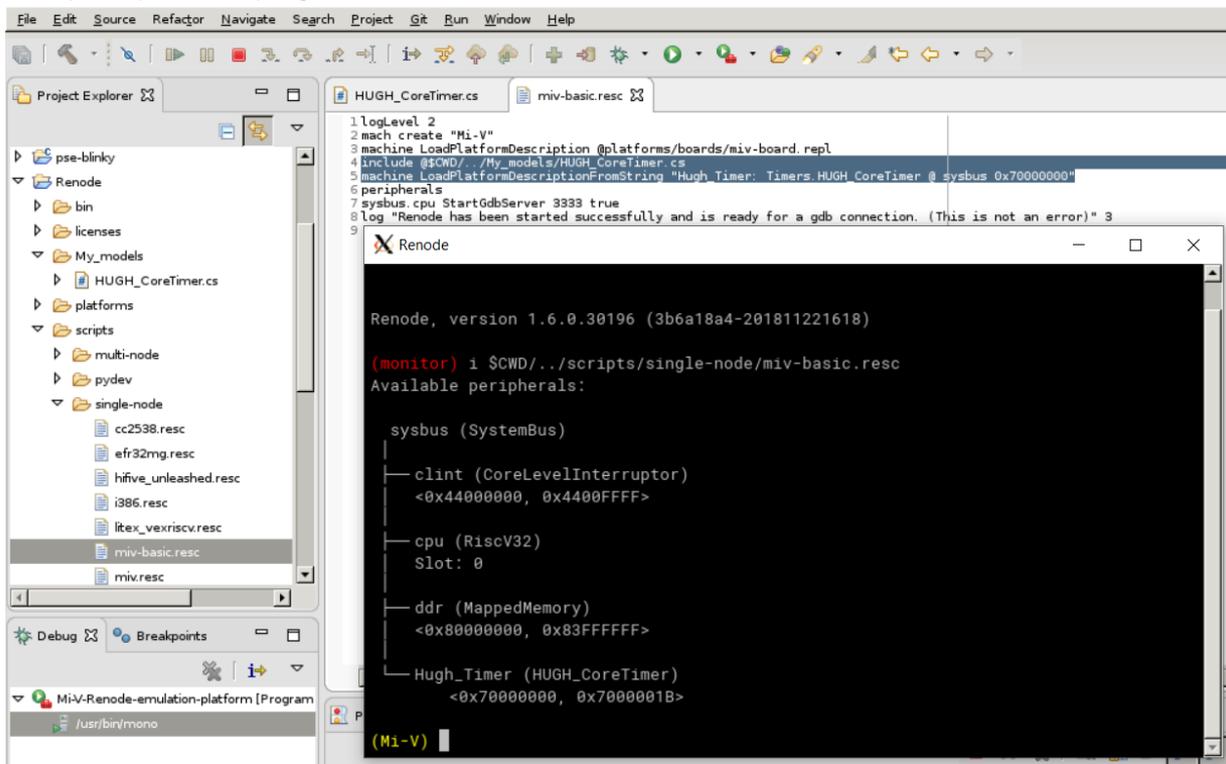
The screenshot shows the Microchip SoftConsole IDE with the following components:

- Project Explorer:** A tree view on the left showing a project structure with folders like 'multi-node', 'pydev', and 'single-node'. Under 'single-node', several '.resc' files are listed, with 'miv-basic.resc' selected.
- Code Editor:** The main window displays the content of 'miv-basic.resc'. The line `machine LoadPlatformDescription @platforms/boards/miv-board.repl` is highlighted in yellow. The script includes log levels, platform loading, peripheral definitions, and a sysbus CPU start command.
- Renode Window:** A separate window titled 'Renode' shows the execution output. It displays the Renode logo, version information, and a list of available peripherals: 'sysbus (SystemBus)', 'clint (CoreLevelInterruptor)', 'cpu (RiscV32)', and 'ddr (MappedMemory)'. The prompt '(Mi-V)' is visible at the bottom.

**Alternatively  
add commands  
to the launch  
script**

# How to add a Just In Time (JIT) compiled model

SC workspace.examples - Renode/scripts/single-node/miv-basic.resc - Microsemi SoftConsole v6.0.0.116



The screenshot shows the Renode IDE interface. The Project Explorer on the left shows a project structure with folders like 'bin', 'licenses', 'My\_models', 'platforms', and 'scripts'. The 'scripts' folder is expanded to show 'single-node' with files like 'cc2538.resc', 'efr32mg.resc', 'hifive\_unleashed.resc', 'i386.resc', 'itek\_vexriscv.resc', 'miv-basic.resc', and 'miv.resc'. The main editor shows the file 'miv-basic.resc' with the following code:

```
1 logLevel 2
2 mach create "Mi-V"
3 machine LoadPlatformDescription @platforms/boards/miv-board.repl
4 include @SCWD/./My_models/HUGH_CoreTimer.cs
5 machine LoadPlatformDescriptionFromString "Hugh Timer: Timers.HUGH_CoreTimer @sysbus 0x70000000"
6 peripherals
7 sysbus.cpu StartGdbServer 3333 true
8 log "Renode has been started successfully and is ready for a gdb connection. (This is not an error)." 3
9
```

The output window shows the Renode version and the available peripherals:

```
Renode, version 1.6.0.30196 (3b6a18a4-201811221618)
(monitor) i $CWD/./scripts/single-node/miv-basic.resc
Available peripherals:

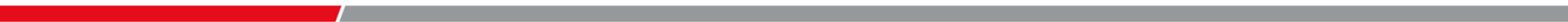
  sysbus (SystemBus)
  |
  |--- clint (CoreLevelInterruptor)
  |   <0x44000000, 0x4400FFFF>
  |
  |--- cpu (RiscV32)
  |   Slot: 0
  |
  |--- ddr (MappedMemory)
  |   <0x80000000, 0x83FFFFFF>
  |
  |--- Hugh_Timer (HUGH_CoreTimer)
  |   <0x70000000, 0x700001B>
  |
  (Mi-V) |
```

**Alternatively  
add commands  
to the launch  
script**



**MICROCHIP**

# **Debugging a model using MonoDevelop**



# Debugging a model using MonoDevelop

---

- **Two options for debugging a model:**
  1. Build Renode from source
    - Full visibility of variables
    - Can suspend execution and debug
  2. Use logs
    - Can set log levels for peripherals and use the logs to debug

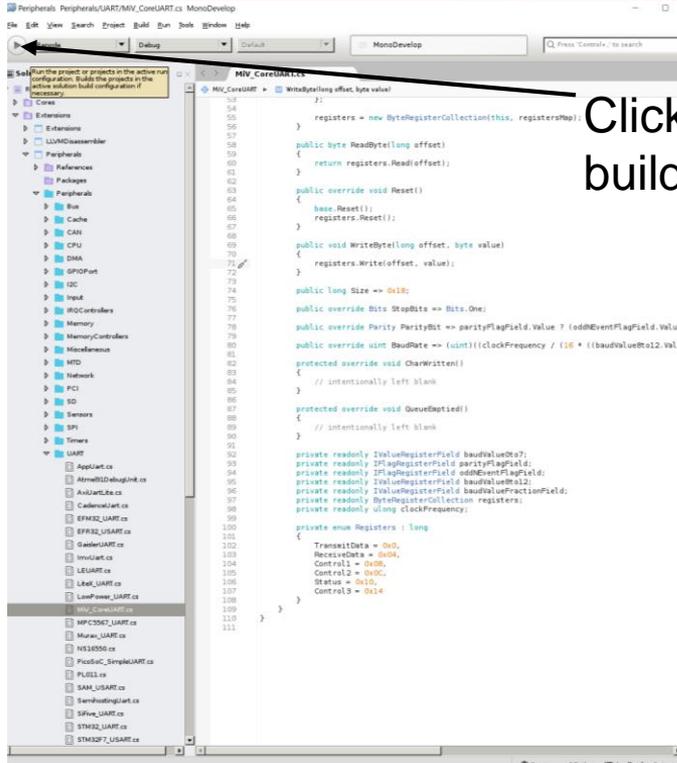
# Debugging a model using MonoDevelop

---

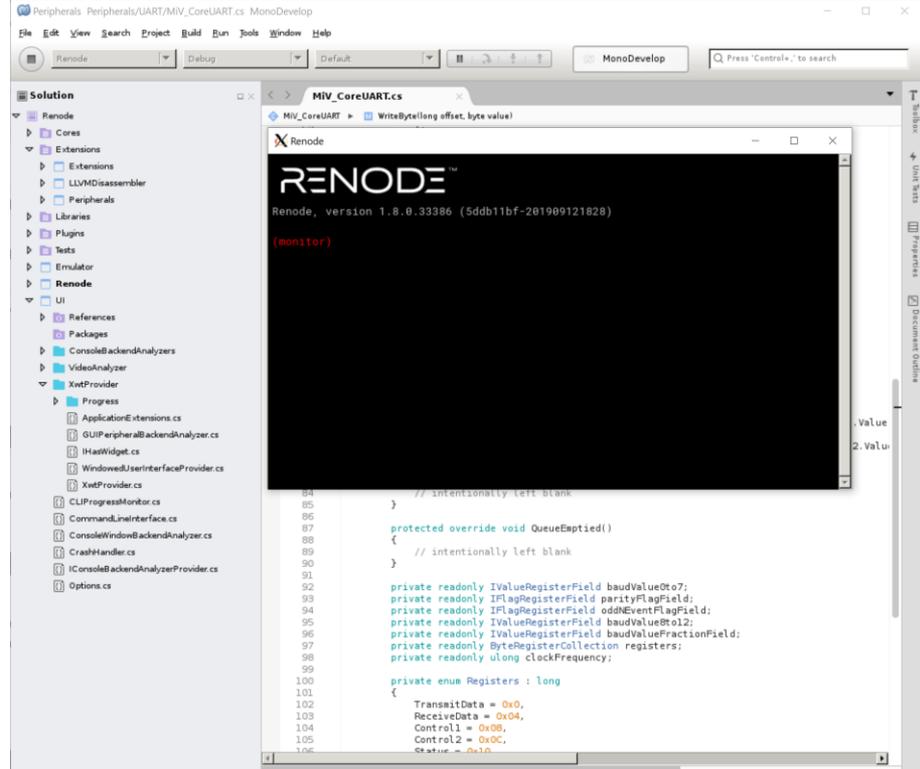
1. **Follow the steps in the supporting video to build Renode from source (build v1.7.1 to match SC 6.0)**
2. **Launch Renode from your C# IDE (e.g monodevelop) and load the platform**
3. **Start the GDB server for debugging**
4. **Run the “attach to renode” debug session in SoftConsole**
5. **Set a breakpoint in the model being tested**



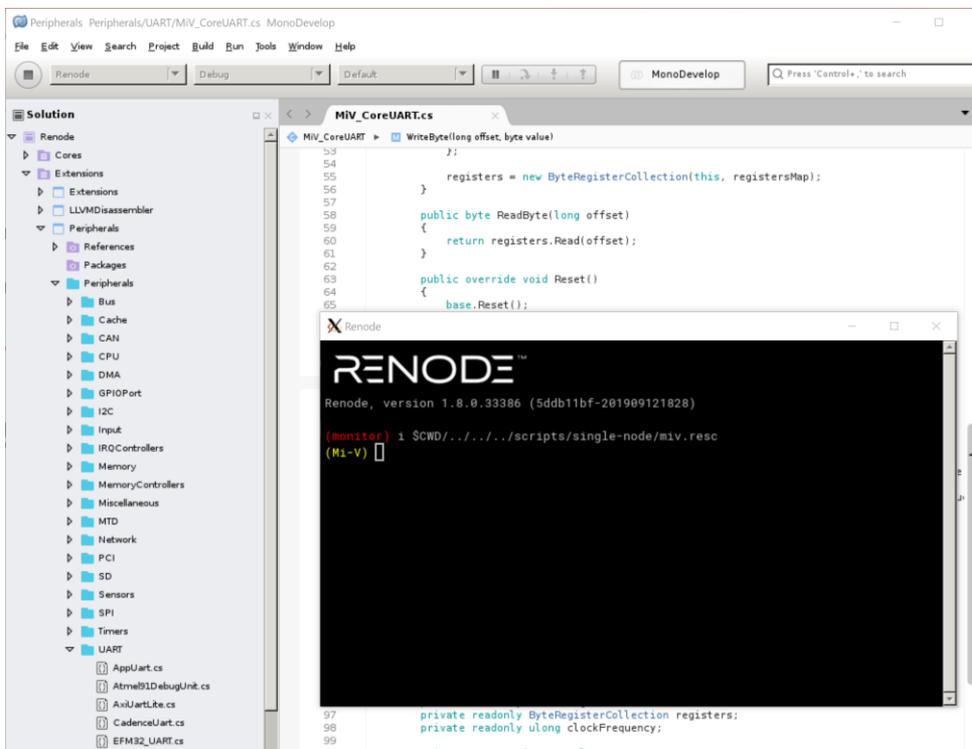
# Debugging a model using MonoDevelop



Click here to build and run



# Debugging a model using MonoDevelop



Paths to launch scripts:

**Mi-V system:**

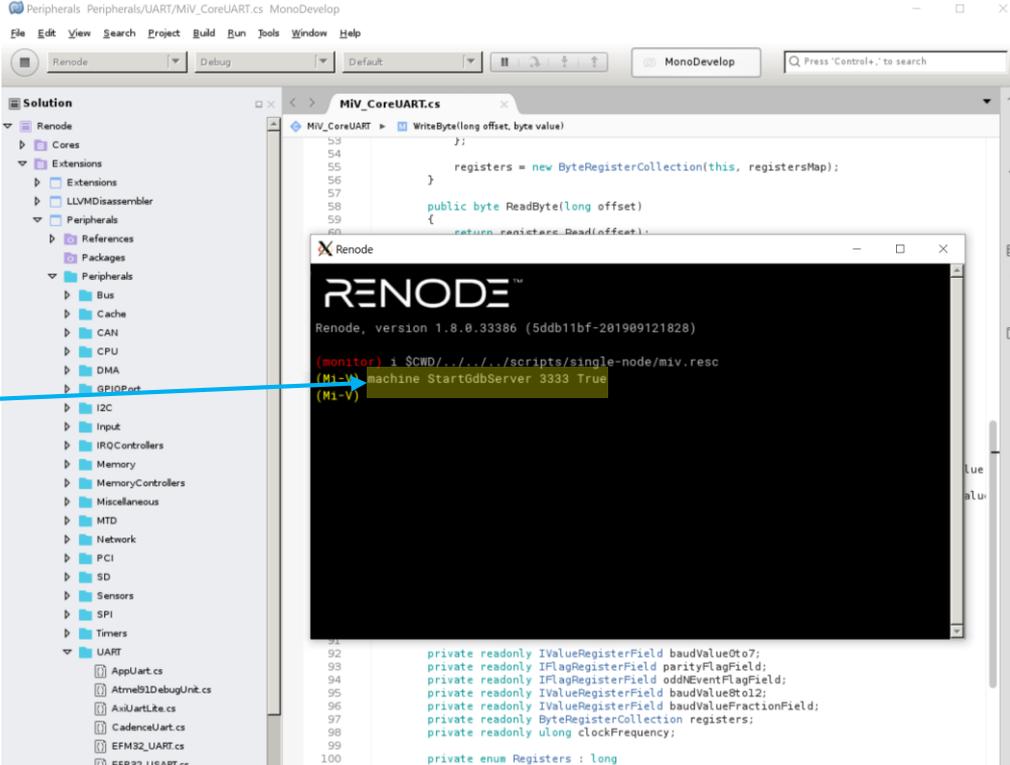
`$CWD/../../../../scripts/single-node/miv.resc`

**PolarFire system:**

`$CWD/../../../../scripts/single-node/polarfire.resc`

# Debugging a model using MonoDevelop

Start the GDB server for the machine



The screenshot shows the MonoDevelop IDE interface. The main window displays the source code for `MIV_CoreUART.cs`, with the `WriteByte(long offset, byte value)` method visible. A terminal window titled "Renode" is overlaid on the code, showing the command `(M1-V) machine StartGdbServer 3333 True` being executed. A blue arrow points from the text "Start the GDB server for the machine" to this command in the terminal. The terminal also shows the Renode version and the current working directory.

```
Peripherals\Peripherals\UART\MIV_CoreUART.cs MonoDevelop
File Edit View Search Project Build Run Tools Window Help
Renode Debug Default MonoDevelop Press 'Control+', to search

Solution
Renode
  Cores
  Extensions
  LLVMDisassembler
  Peripherals
  References
  Packages
  Peripherals
    Bus
    Cache
    CPU
    DMA
    GPIOPort
    I2C
    Input
    IRQControllers
    Memory
    MemoryControllers
    Miscellaneous
    MTD
    Network
    PCI
    SD
    Sensors
    SPI
    Timers
    UART
      AppUart.cs
      AtrmeB1DDebugUnit.cs
      AvxUartLike.cs
      CadenceUart.cs
      EFMS32_UART.cs
      EFR32_USART.cs

MIV_CoreUART.cs
MIV_CoreUART
  WriteByte(long offset, byte value)
  registers = new ByteRegisterCollection(this, registersMap);
  }
  public byte ReadByte(long offset)
  {
  return registers.Read(offset);
  }

Renode
Renode, version 1.8.0.33386 (5ddb11bf-201909121828)
(monitor) | $CWD/../../../../scripts/single-node/miv.resc
(M1-V) machine StartGdbServer 3333 True
(M1-V)

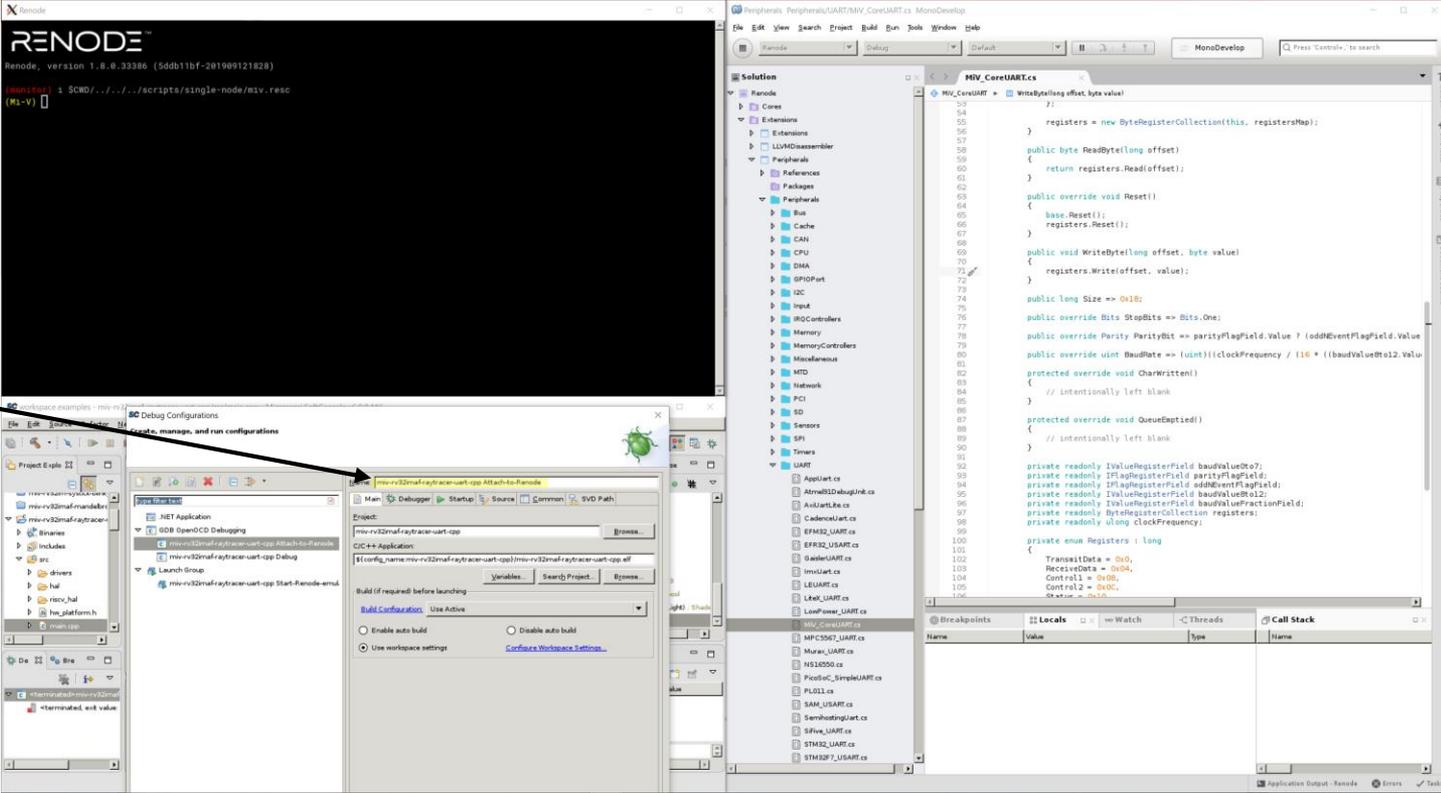
private readonly IValueRegisterField baudValue0to7;
private readonly IFlagRegisterField parityFlagField;
private readonly IFlagRegisterField oddEventFlagField;
private readonly IValueRegisterField baudValue0to12;
private readonly IValueRegisterField baudValueFractionField;
private readonly ByteRegisterCollection registers;
private readonly ulong clockFrequency;

private enum Registers : long
{
```

# Debugging a model using MonoDevelop

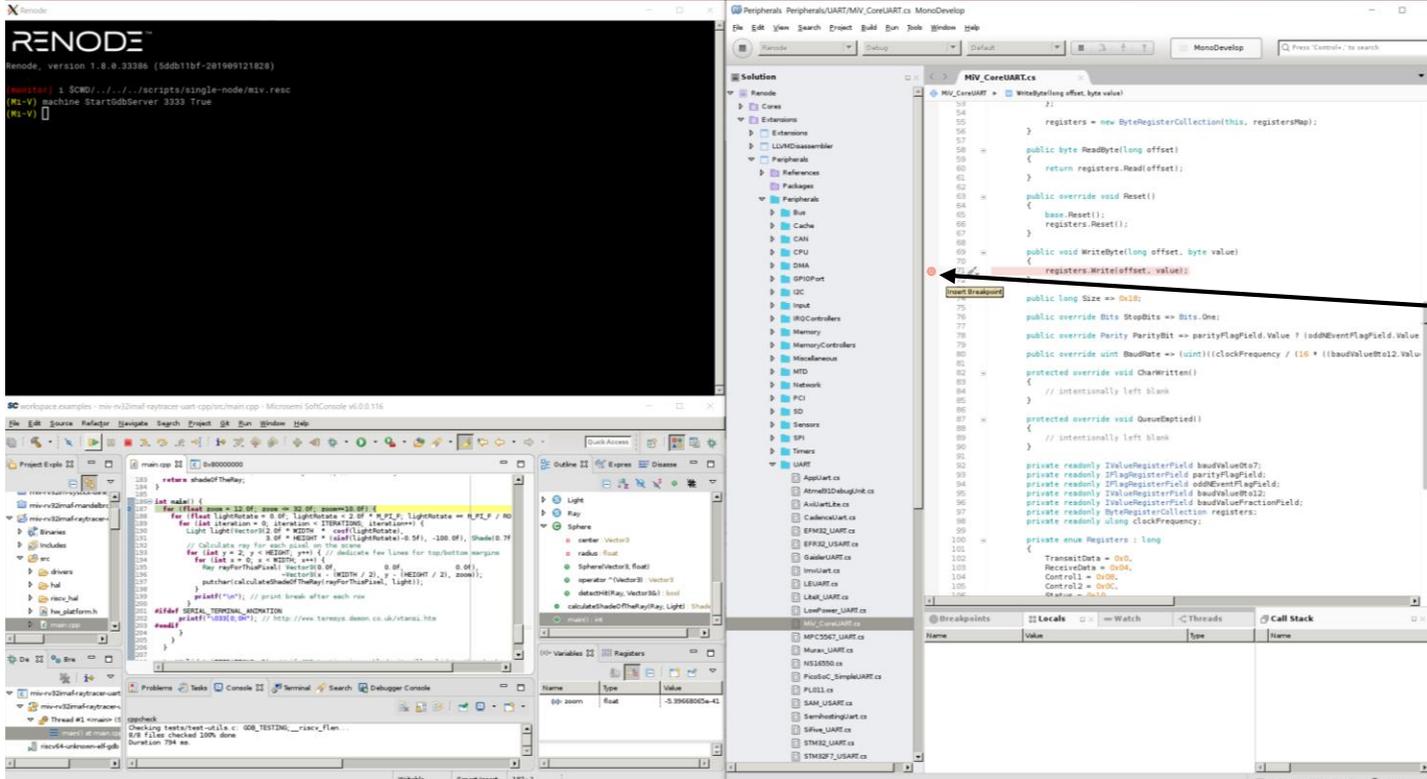
Use the “Attach-to-Renode” debug session to connect to Renode

Do not use the launch group – Renode is already running!



The screenshot displays the MonoDevelop IDE interface. The top-left pane shows the Renode terminal window with the command `renode --script /.../scripts/single-node/mv.resc`. The top-right pane shows the code editor with the `MV_CoreUART.cs` file, containing C# code for a UART peripheral. The bottom-left pane shows the 'Debug Configurations' dialog, where a configuration named 'Attach-to-Renode' is selected. An arrow points from the text 'Do not use the launch group – Renode is already running!' to the 'Launch Group' field in the configuration, which is currently set to 'mv-rv32maf-eyetracer-uart-opp Start-Renode-emu'. The bottom-right pane shows the 'Solution' explorer with a tree view of project files, including `AppInit.cs`, `ARMDebugLink.cs`, `AXI4Lite.cs`, `CadenceUart.cs`, `EFMS2_UART.cs`, `EFMS2_USART.cs`, `GPIO.cs`, `GPIOUart.cs`, `LowPower_UART.cs`, `MCUSPI_UART.cs`, `Mux_UART.cs`, `NS1050.cs`, `Picovoc_SimpleUART.cs`, `PL011.cs`, `SAM_UART.cs`, `SerialDebugUart.cs`, `Silvix_UART.cs`, `STM32_UART.cs`, and `STM32P_USART.cs`. The bottom-right pane also shows the 'Breakpoints', 'Locals', 'Watch', 'Threads', and 'Call Stack' windows.

# Debugging a model using MonoDevelop

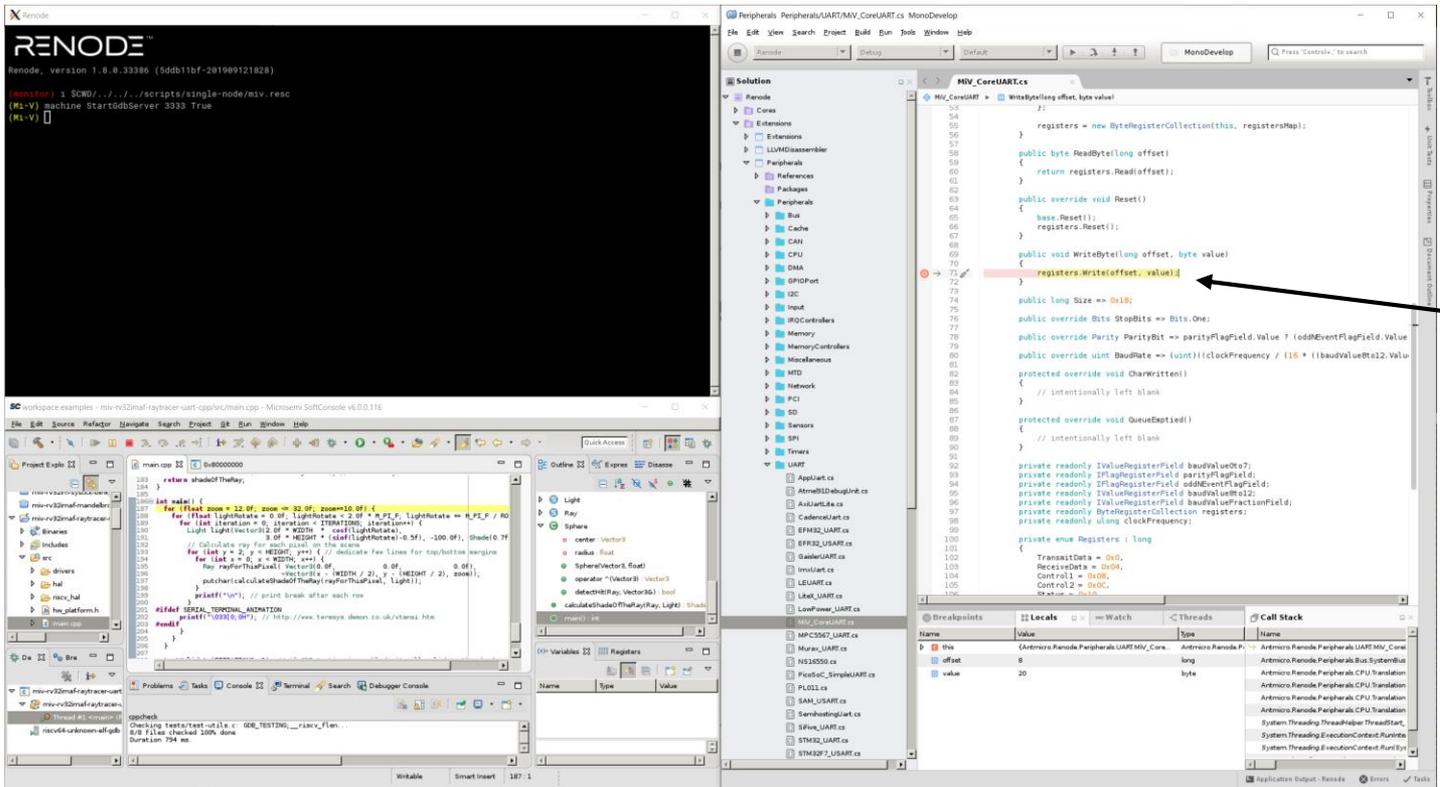


The screenshot shows the MonoDevelop IDE with the following components:

- Solution Explorer:** Shows a project structure for 'Peripherals' and 'MIV\_CoreUART.cs'.
- Code Editor:** Displays C# code for 'MIV\_CoreUART.cs'. A red circle and arrow point to line 61, where a breakpoint is set on the line: `registers.WriteByte(long offset, byte value);`
- Source Window:** Shows the source code for 'main.cpp' with a C++ function `setLightState`.
- Breakpoints Window:** Shows a list of breakpoints, with one active breakpoint at line 61 of 'MIV\_CoreUART.cs'.
- Variables and Registers:** Shows the current state of variables and registers.

**Click on the left of a line to set a breakpoint**

# Debugging a model using MonoDevelop



The screenshot shows the MonoDevelop IDE with the following components:

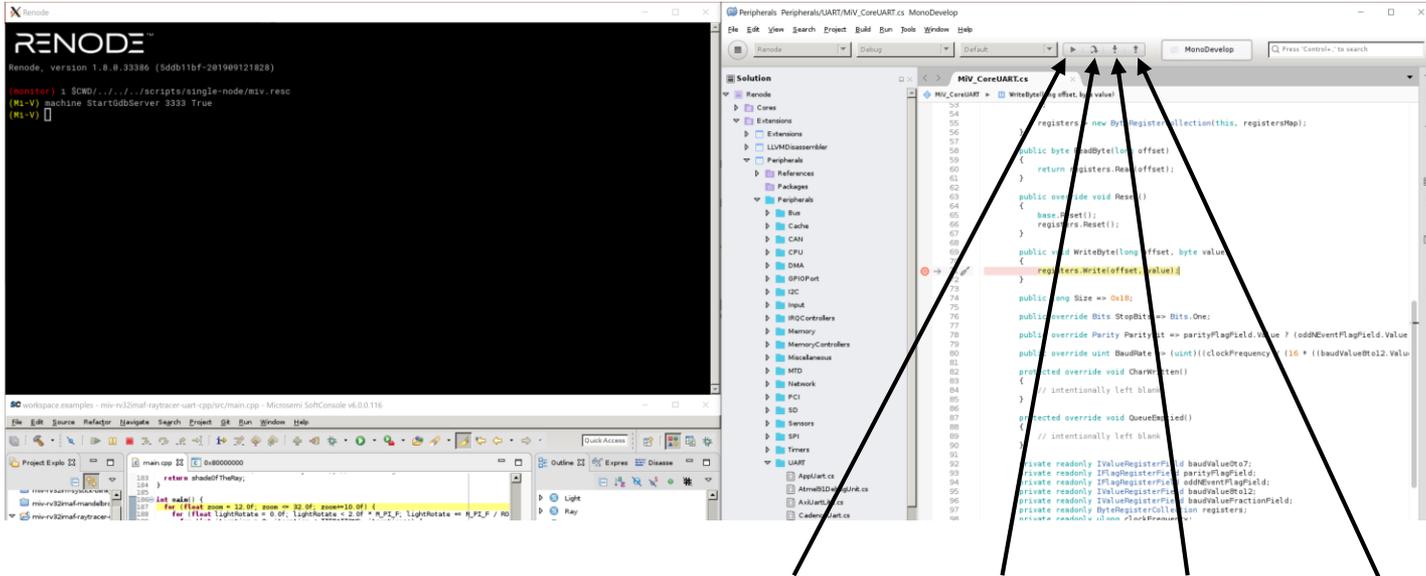
- Solution Explorer:** Displays the project structure for 'Renode', including folders for Extensions, LVMDisassembler, Peripherals, Packages, and various hardware components like Cache, CAN, CPU, DMA, GPIO, Input, I2C, SPI, UART, etc.
- Code Editor:** Shows the source code for 'MNV\_CareUART.cs'. A red arrow points to a breakpoint set on the line:
 

```
public void Write(offset, byte value)
{
    registers.Write(offset, value);
}
```
- Breakpoints/Locals/Threads/Call Stack:** The 'Breakpoints' pane is active, showing a breakpoint at line 71 of 'MNV\_CareUART.cs'. The 'Locals' pane shows the current state of variables:
 

| Name   | Value   | Type  |
|--------|---|---|
| this   | (Antmicro.Renode.Peripherals.UART.MNV_CareUART) | Antmicro.Renode.Peripherals.UART.MNV_CareUART |
| offset | 8   | long  |
| value  | 20  | byte  |
- Debugger Console:** Shows the execution of the 'main' method in 'main.cpp', with a breakpoint hit at line 187.

Run software that uses the model so it reaches the breakpoint

# Debugging a model using MonoDevelop

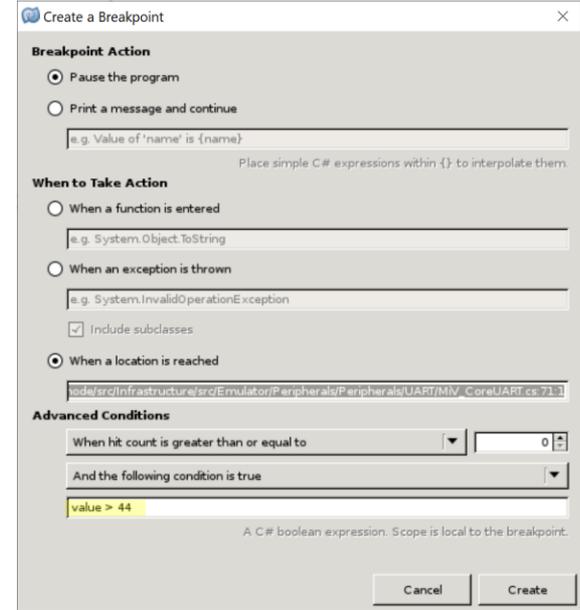
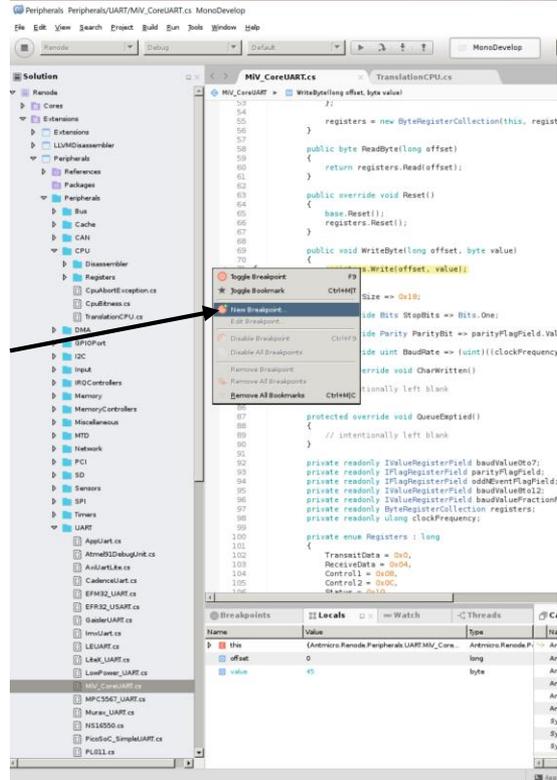


**Resume**    **Step over**    **Step into**    **Step up**

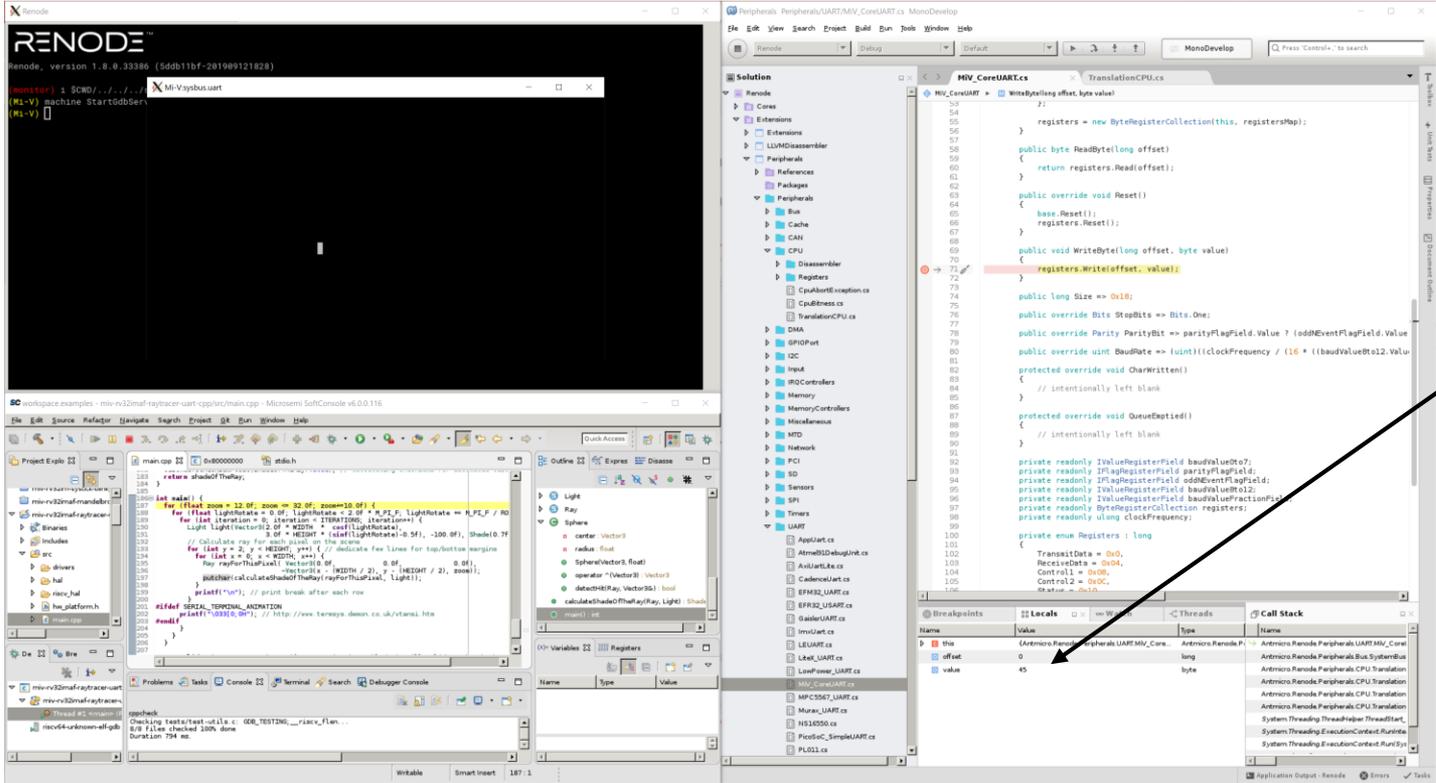
# Debugging a model using MonoDevelop

Custom breakpoints can be created to only halt during certain conditions

Right click on the left and select “New Breakpoint...”



# Debugging a model using MonoDevelop



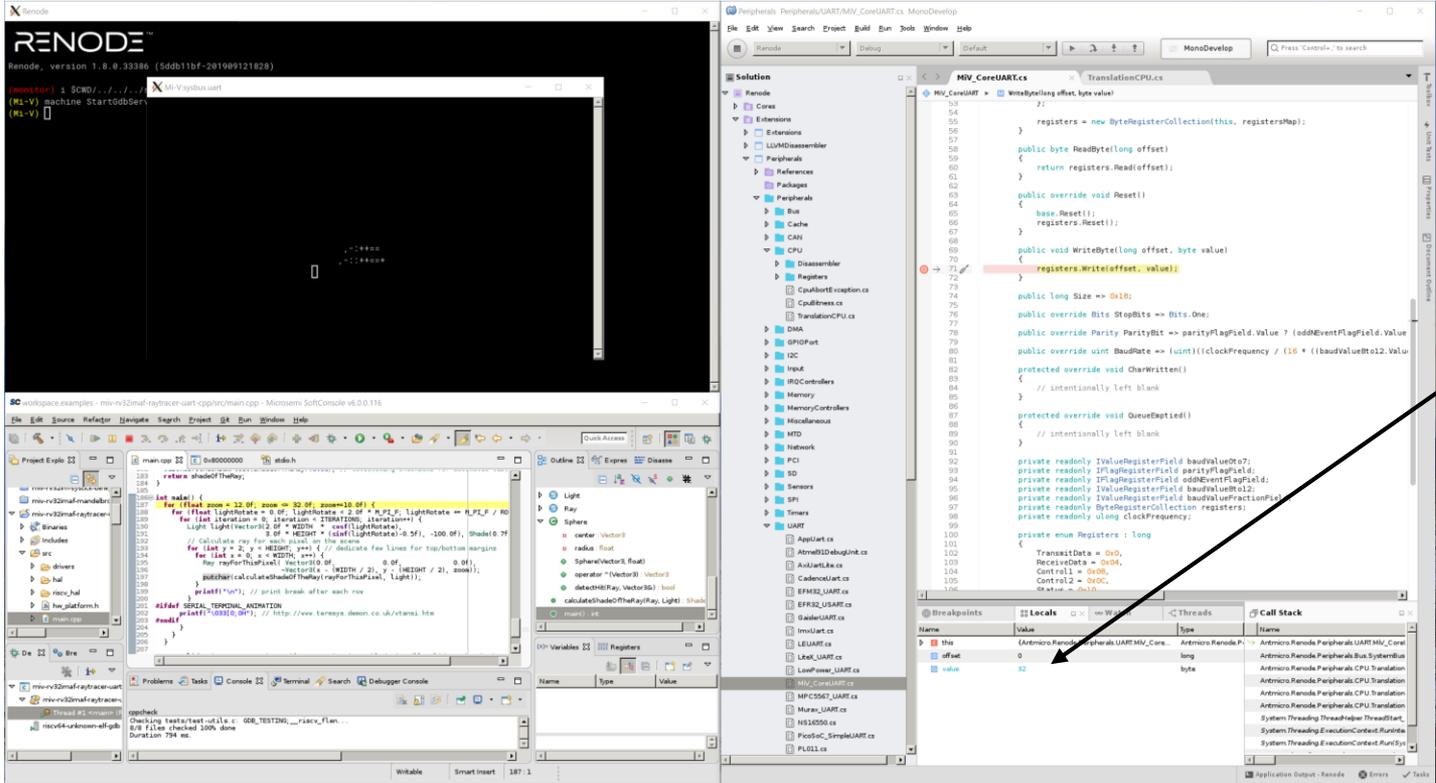
The image shows the MonoDevelop IDE with the following components:

- Top Left:** RENODE logo and version information (version 1.6.0.32386).
- Top Right:** Solution Explorer showing the project structure for 'MNV\_CareUART.cs'.
- Center:** Source code for 'UART' class, including methods like 'ReadByte', 'Reset', and 'WriteByte'.
- Bottom Left:** Source code for 'main.cpp' with a yellow highlight on a loop iteration.
- Bottom Center:** Variable Watch window showing:
 

| Name   | Type  | Value   |
|--------|---|---|
| this   | Antmicro.Renode.Peripherals.UART.MNV_CareUART | Antmicro.Renode.Peripherals.UART.MNV_CareUART |
| offset | long  | 0   |
| value  | byte  | 45  |
- Bottom Right:** Call Stack window showing the current execution context.

Check variable values etc at the bottom

# Debugging a model using MonoDevelop



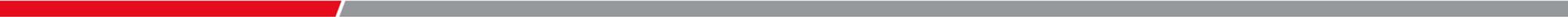
The screenshot displays the MonoDevelop IDE with the following components:

- Top Left:** A terminal window showing the Renode version (1.6.0.3326) and a command prompt.
- Top Right:** A Solution Explorer showing a project structure with folders for Extensions, Peripherals, Packages, and various hardware components like CPU, DMA, and UART.
- Center:** A code editor showing the source code for `MNV_CareUART.cs`. The code includes a `ByteRegisterCollection` and methods for `ReadByOffset`, `Reset`, and `WriteByOffset`.
- Bottom Left:** A Project Explorer showing a file tree with folders like `bin`, `obj`, and `src`.
- Bottom Center:** A debugger console window showing a stack trace. The entry for `this` is highlighted, with a value of `32`. An arrow points from the text 'Check variable values etc at the bottom' to this entry.
- Bottom Right:** A Call Stack window showing the current thread's call stack, including `Antmicro.Renode.Peripherals.UART.MNV_CareUART`.

Check variable values etc at the bottom



# Debugging a model using logs



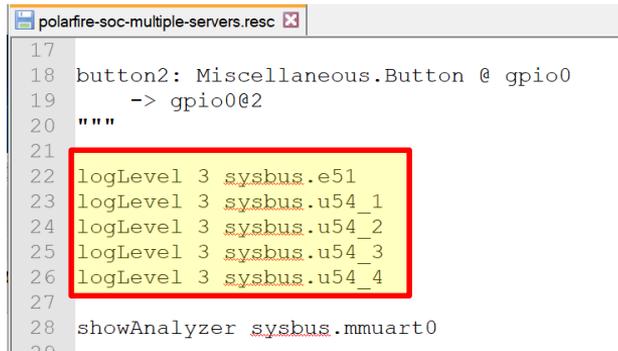
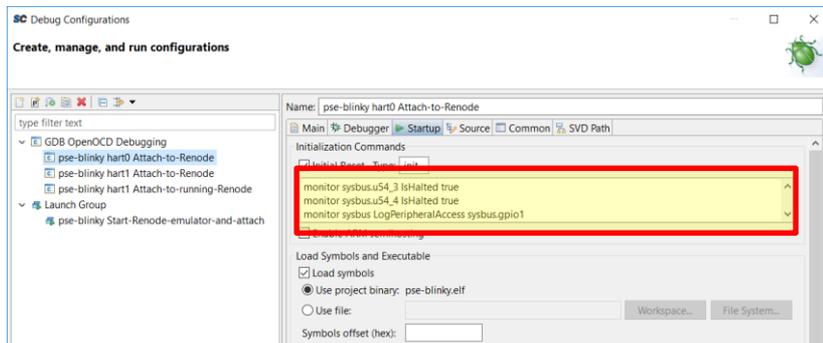
# Debugging a model using logs

---

- **There are several logging levels available:**
  - -1 Noisy
  - 0 Debug
  - 1 Info
  - 2 Warning
  - 3 Error
  
- **Each can capture different levels of data**
  
- **Peripherals will produce a different level of log data depending on what has caused the log (e.g. uart print vs fatal error)**

# Debugging a model using logs

- Logging (and other commands) can be set up automatically by passing the commands during start up or including them in the launch script
- E.G (pse-blinky debug configuration):
  - monitor sysbus LogPeripheralAccess sysbus.gpio1
  - monitor logLevel -1 sysbus.gpio1
- E.G (pse-blinky launch script):
  - logLevel 3 sysbus.e51



# Debugging a model using logs

---

| Log Level  | Log Peripheral Access   |
|--|---|
| Produces different levels of content depending on log level selected | Creates a log entry every time the CPU tries to access a peripheral |
| Logging commands included in the peripheral model                    | Can be set individually per peripheral                              |
| Variable values can be included in the log                           | Log level must be set to 0 (debug)                                  |
|  | Peripheral must be on system bus                                    |



# Debugging a model using logs

PolarFire-SoC-Renode-emulation-platform [Program] C:\Microsemi\SoftConsole\_v6.0\renode\bin\Renode.exe

```
15:36:32.5299 [INFO] Including script: C:\Microsemi\SoftConsole_v6.0\renode\scripts\single-node\polarfire-soc-multiple-servers.resc
15:36:34.8709 [ERROR] Script: Renode has been started successfully and is ready for a gdb connection. (This is not an error)
15:36:36.3250 [ERROR] u54_1: CPU abort [PC=0x1000]: Trying to execute code outside RAM or ROM at 0x000000000001000.
15:36:36.3250 [ERROR] u54_2: CPU abort [PC=0x1000]: Trying to execute code outside RAM or ROM at 0x000000000001000.
```

Log level 3  
(error)

PolarFire-SoC-Renode-emulation-platform [Program] C:\Microsemi\SoftConsole\_v6.0\renode\bin\Renode.exe

```
15:42:20.5119 [INFO] Including script: C:\Microsemi\SoftConsole_v6.0\renode\scripts\single-node\polarfire-soc-multiple-servers.resc
15:42:22.9525 [ERROR] Script: Renode has been started successfully and is ready for a gdb connection. (This is not an error)
15:42:25.3827 [ERROR] u54_2: CPU abort [PC=0x1000]: Trying to execute code outside RAM or ROM at 0x000000000001000.
15:42:25.3847 [ERROR] u54_1: CPU abort [PC=0x1000]: Trying to execute code outside RAM or ROM at 0x000000000001000.
15:42:32.2031 [WARNING] gpio1: Unhandled write to offset 0x0. Unhandled bits: [2] when writing value 0x5. Tags: OutputBufferEnable (0x1).
15:42:32.2036 [WARNING] gpio1: Unhandled write to offset 0x4. Unhandled bits: [2] when writing value 0x5. Tags: OutputBufferEnable (0x1).
15:42:32.2036 [WARNING] gpio1: Unhandled write to offset 0x8. Unhandled bits: [2] when writing value 0x5. Tags: OutputBufferEnable (0x1).
```

Log level 2  
(warning)

Log level 1  
(info)

Log level 0  
(Debug)

# Debugging a model using logs

```
PSE_Watchdog.cs
PSE_Watchdog DefineRegisters()
47     this.Log(LogLevel.Debug, "Starting watchdog.");
48     internalTimer.Enabled = true;
49     SetState(State.ForbiddenRegion);
50 }
51 else if(state == State.RefreshRegion && value == WatchdogReset)
52 {
53     this.Log(LogLevel.Noisy, "Refreshing watchdog.");
54     SetState(State.ForbiddenRegion);
55 }
56 else if(state == State.ForbiddenRegion && forbiddenRangeEnabled.Value)
57 {
58     this.Log(LogLevel.Warning, "Watchdog refreshed in forbidden region, triggering NMI.");
59     SetState(State.AfterTrigger);
60 }
61 }, valueProviderCallback: _ => GetCurrentTimerValue(), name: "REFRESH")
62 .WithWriteCallback( (_, __) => locked.Value = true);
63 ;
```

E.G you could add the model as JIT and add more logging as needed to the file and re-launch Renode to add the increased logging

# Debugging a model using logs

```
PolarFire-SoC-Renode-emulation-platform [Program] C:\Microsemi\SoftConsole_v6.0\renode\bin\Renode.exe
15:42:20.5119 [INFO] Including script: C:\Microsemi\SoftConsole_v6.0\renode\scripts\single-node\polarfire-soc-multiple-servers.resc
15:42:22.9525 [ERROR] Script: Renode has been started successfully and is ready for a gdb connection. (This is not an error)
15:42:25.3827 [ERROR] u54_2: CPU abort [PC=0x1000]: Trying to execute code outside RAM or ROM at 0x0000000000001000.
15:42:25.3847 [ERROR] u54_1: CPU abort [PC=0x1000]: Trying to execute code outside RAM or ROM at 0x0000000000001000.
15:42:32.2031 [WARNING] gpio1: Unhandled write to offset 0x0. Unhandled bits: [2] when writing value 0x5. Tags: OutputBufferEnable (0x1).
15:42:32.2036 [WARNING] gpio1: Unhandled write to offset 0x4. Unhandled bits: [2] when writing value 0x5. Tags: OutputBufferEnable (0x1).
15:42:32.2036 [WARNING] gpio1: Unhandled write to offset 0x8. Unhandled bits: [2] when writing value 0x5. Tags: OutputBufferEnable (0x1).
```

## Log level 0 (Debug) + LogPeripheralAccess

```
PolarFire-SoC-Renode-emulation-platform [Program] C:\Microsemi\SoftConsole_v6.0\renode\bin\Renode.exe
15:54:05.3206 [INFO] Including script: C:\Microsemi\SoftConsole_v6.0\renode\scripts\single-node\polarfire-soc-multiple-servers.resc
15:54:08.4482 [ERROR] Script: Renode has been started successfully and is ready for a gdb connection. (This is not an error)
15:54:10.9811 [ERROR] u54_1: CPU abort [PC=0x1000]: Trying to execute code outside RAM or ROM at 0x0000000000001000.
15:54:10.9811 [ERROR] u54_2: CPU abort [PC=0x1000]: Trying to execute code outside RAM or ROM at 0x0000000000001000.
15:54:55.7182 [DEBUG] gpio1: WriteUInt32 to 0x0 (unknown), value 0x5.
15:54:55.7336 [WARNING] gpio1: Unhandled write to offset 0x0. Unhandled bits: [2] when writing value 0x5. Tags: OutputBufferEnable (0x1).
15:54:55.7351 [DEBUG] gpio1: WriteUInt32 to 0x4 (unknown), value 0x5.
15:54:55.7356 [WARNING] gpio1: Unhandled write to offset 0x4. Unhandled bits: [2] when writing value 0x5. Tags: OutputBufferEnable (0x1).
15:54:55.7356 [DEBUG] gpio1: WriteUInt32 to 0x8 (unknown), value 0x5.
15:54:55.7356 [WARNING] gpio1: Unhandled write to offset 0x8. Unhandled bits: [2] when writing value 0x5. Tags: OutputBufferEnable (0x1).
15:54:55.7361 [DEBUG] gpio1: WriteUInt32 to 0x88 (OutputRegister), value 0x0.
15:54:55.7589 [DEBUG] gpio1: WriteUInt32 to 0xA4 (SetRegister), value 0x1.
15:54:55.7599 [DEBUG] gpio1: WriteUInt32 to 0xA4 (SetRegister), value 0x2.
15:54:55.7604 [DEBUG] gpio1: WriteUInt32 to 0xA4 (SetRegister), value 0x4.
15:54:55.7624 [DEBUG] gpio1: WriteUInt32 to 0xA0 (ClearRegister), value 0x1.
15:54:55.7624 [DEBUG] gpio1: WriteUInt32 to 0xA0 (ClearRegister), value 0x2.
15:54:55.7629 [DEBUG] gpio1: WriteUInt32 to 0xA0 (ClearRegister), value 0x4.
15:54:55.7649 [DEBUG] gpio1: WriteUInt32 to 0xA4 (SetRegister), value 0x1.
```

# Summary

---

- **Where are models in Renode**
- **Ways to add models**
- **How to add a Just In Time (JIT) compiled model**
- **Debugging a model**



# First Thursdays

---

May 2 - Webinar 1: Discover Renode for PolarFire® SoC Design and Debug

June 6 - Webinar 2: How to Get Started with Renode for PolarFire SoC

July 4 - Webinar 3: Learn to Debug a Bare-Metal PolarFire SoC Application with Renode

Aug. 1 - Webinar 4: Tips and Tricks for Even Easier PolarFire SoC Debug with Renode

Sept. 5 - Webinar 5: Add and Debug PolarFire SoC models with Renode

Oct. 3 - Webinar 6: Add and Debug and Pre-Existing model in PolarFire SoC

**Nov. 7 - Webinar 7: How to write custom models – filters, offloading, acceleration etc**

**Dec. 5 - Webinar 8: Handling Binaries**

**Contd.**



# Second Thursdays

---

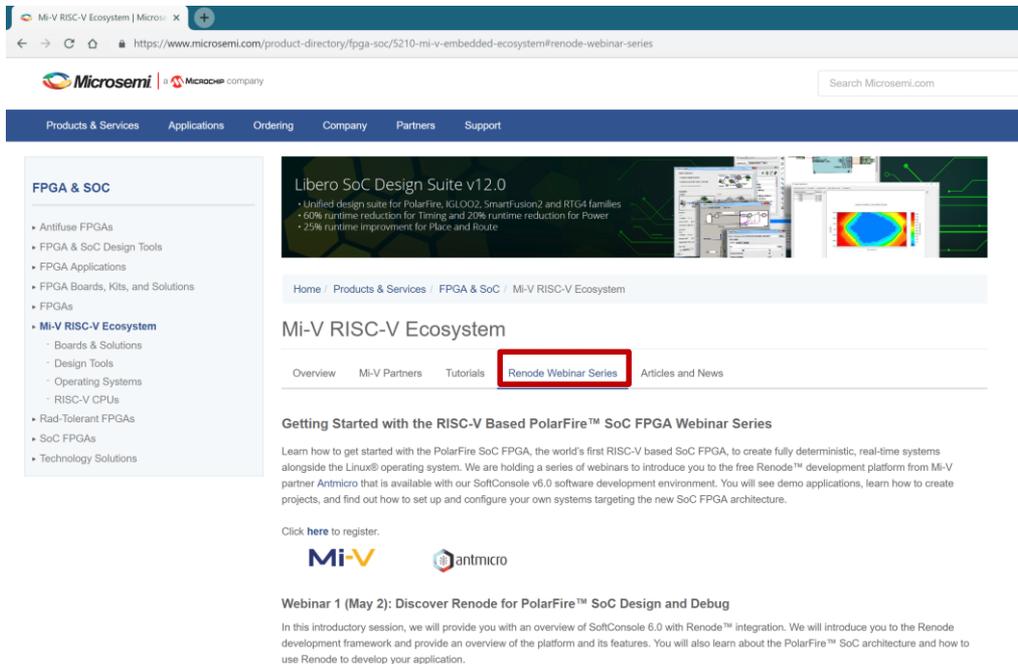
**Jan. 9 - Webinar 9: Run Linux on Renode (PolarFire SoC Model as a Quad-core SMP) – this is not a Linux / Buildroot tutorial**

**Feb. 13 - Webinar 10: Build applications for Linux on PolarFire SoC**

**Mar. 12 - Webinar 11: Introduction to PolarFire SoC MSS Configuration and Software Flow**

**Apr. 9 - Webinar 12: Two baremetal Applications on PolarFire SoC**

**May 14 - Webinar 13: Linux + Real-Time (AMP Mode) on PolarFire SoC**



The screenshot shows a web browser window with the URL <https://www.microsemi.com/product-directory/fpga-soc/5210-mi-v-embedded-ecosystem#renode-webinar-series>. The page features a navigation menu with 'Products & Services', 'Applications', 'Ordering', 'Company', 'Partners', and 'Support'. A sidebar on the left lists 'FPGA & SOC' categories, including 'Mi-V RISC-V Ecosystem'. The main content area displays 'Libero SoC Design Suite v12.0' and a breadcrumb trail: 'Home / Products & Services / FPGA & SoC / Mi-V RISC-V Ecosystem'. Below this, the 'Mi-V RISC-V Ecosystem' page is shown with a navigation bar containing 'Overview', 'Mi-V Partners', 'Tutorials', 'Renode Webinar Series' (highlighted with a red box), and 'Articles and News'. The 'Getting Started with the RISC-V Based PolarFire™ SoC FPGA Webinar Series' section includes a paragraph about the PolarFire SoC FPGA and a link to register, accompanied by the Mi-V and antmicro logos.

[www.microsemi.com/Mi-V](https://www.microsemi.com/Mi-V) “Renode Webinar Series”



**MICROCHIP**

**Thank You**

