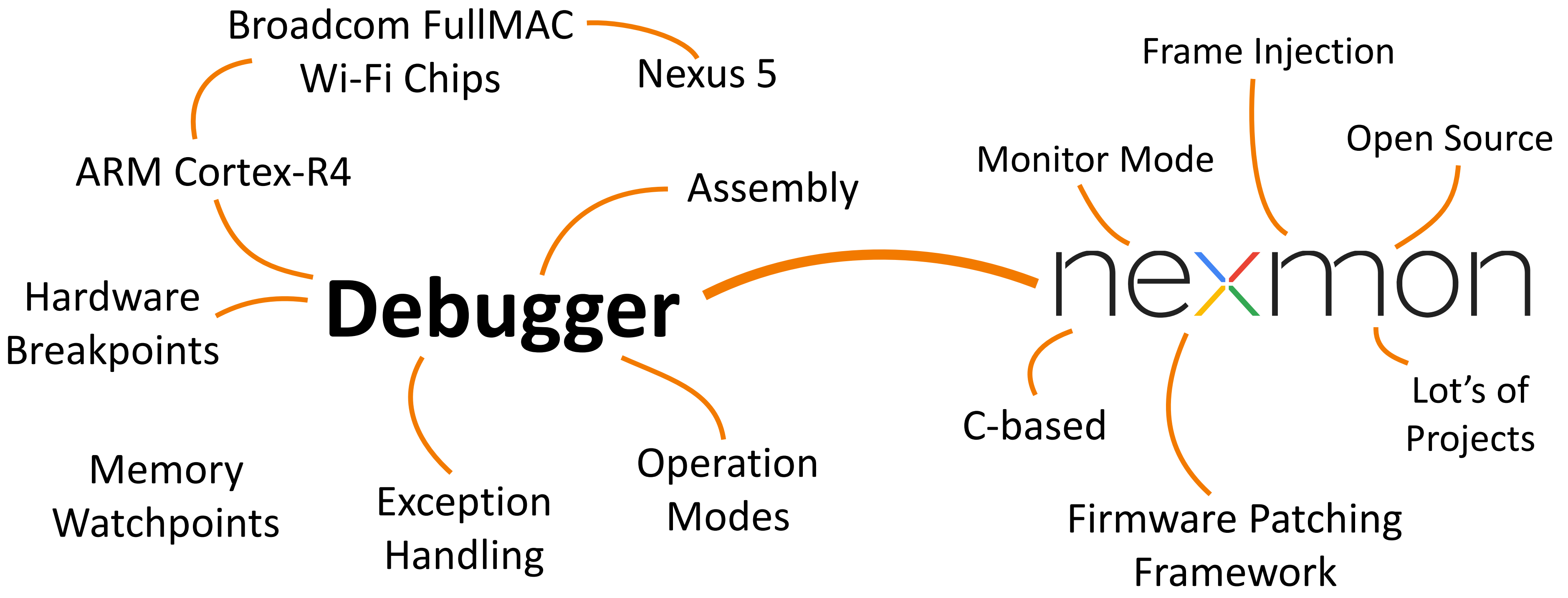
DIY ARM Debugger for Wi-Fi Chips

Using Nexmon to Perform Single-Step Debugging and More on Proprietary Wi-Fi Chips

Matthias Schulz

TECHNISCHE
UNIVERSITÄT
DARMSTADT



Broadcom FullMAC Wi-Fi Chips

Nexus 5

ARM Cortex-R4

Assembly

Hardware Breakpoints

Debugger

Memory Watchpoints

Exception Handling

Operation Modes

Frame Injection

Monitor Mode

Open Source

nexmon

C-based

Firmware Patching Framework

Lot's of Projects

Powered by:

nexmon

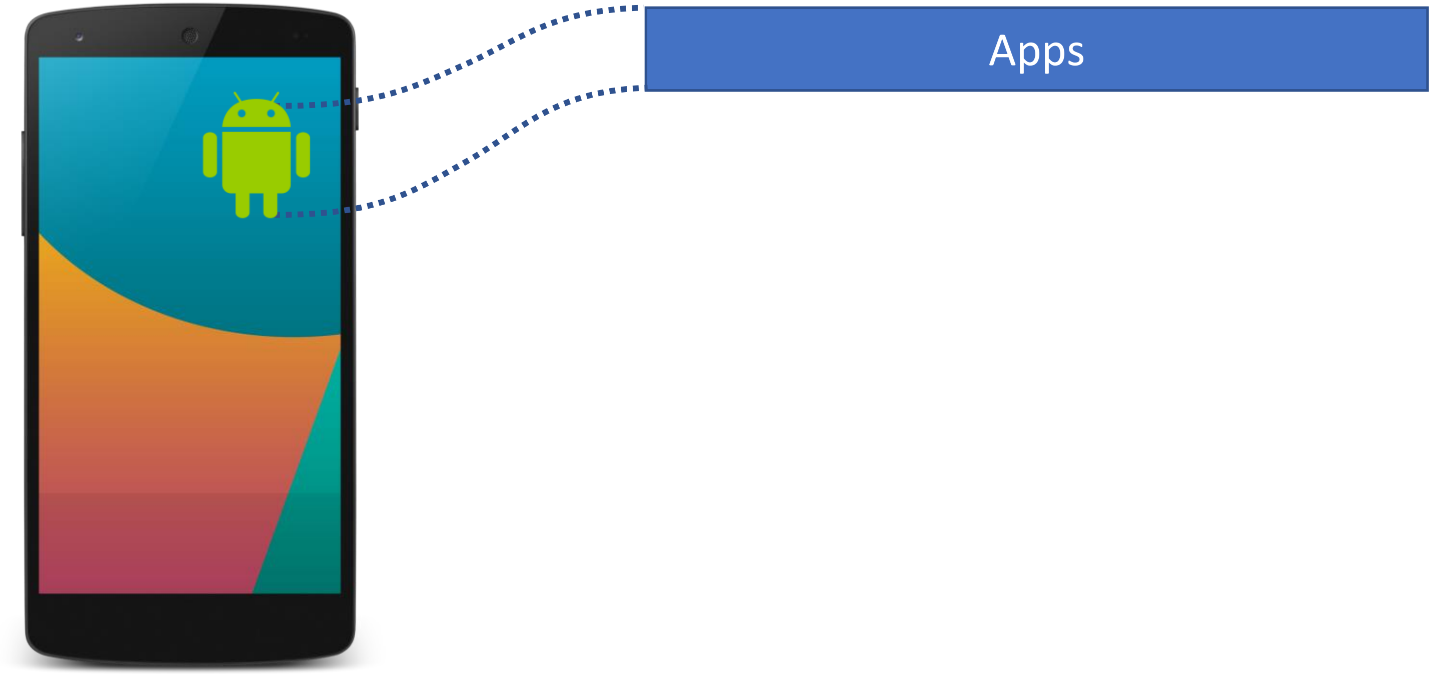
SEEMOO
SECURE MOBILE NETWORKING

NICER

Wi-Fi Chips in Smartphones

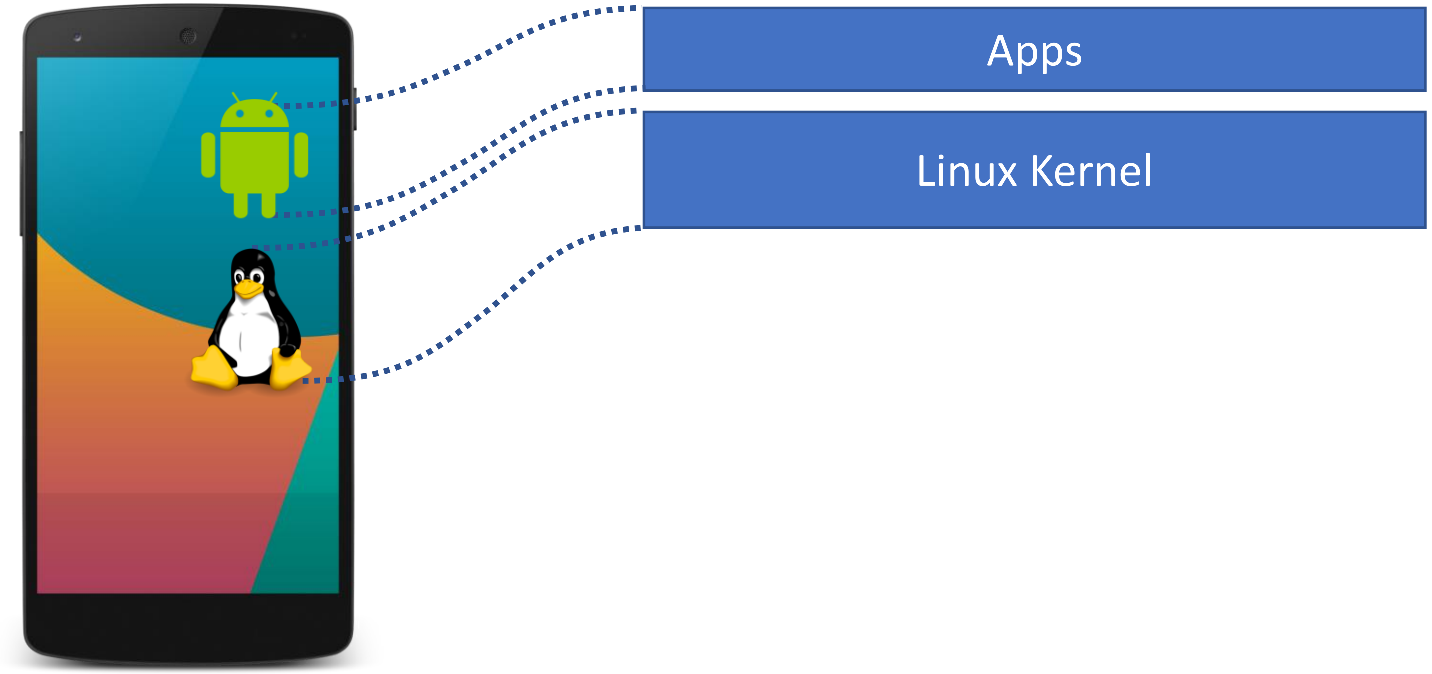
Nexus 5

Wi-Fi Chips in Smartphones



Nexus 5

Wi-Fi Chips in Smartphones

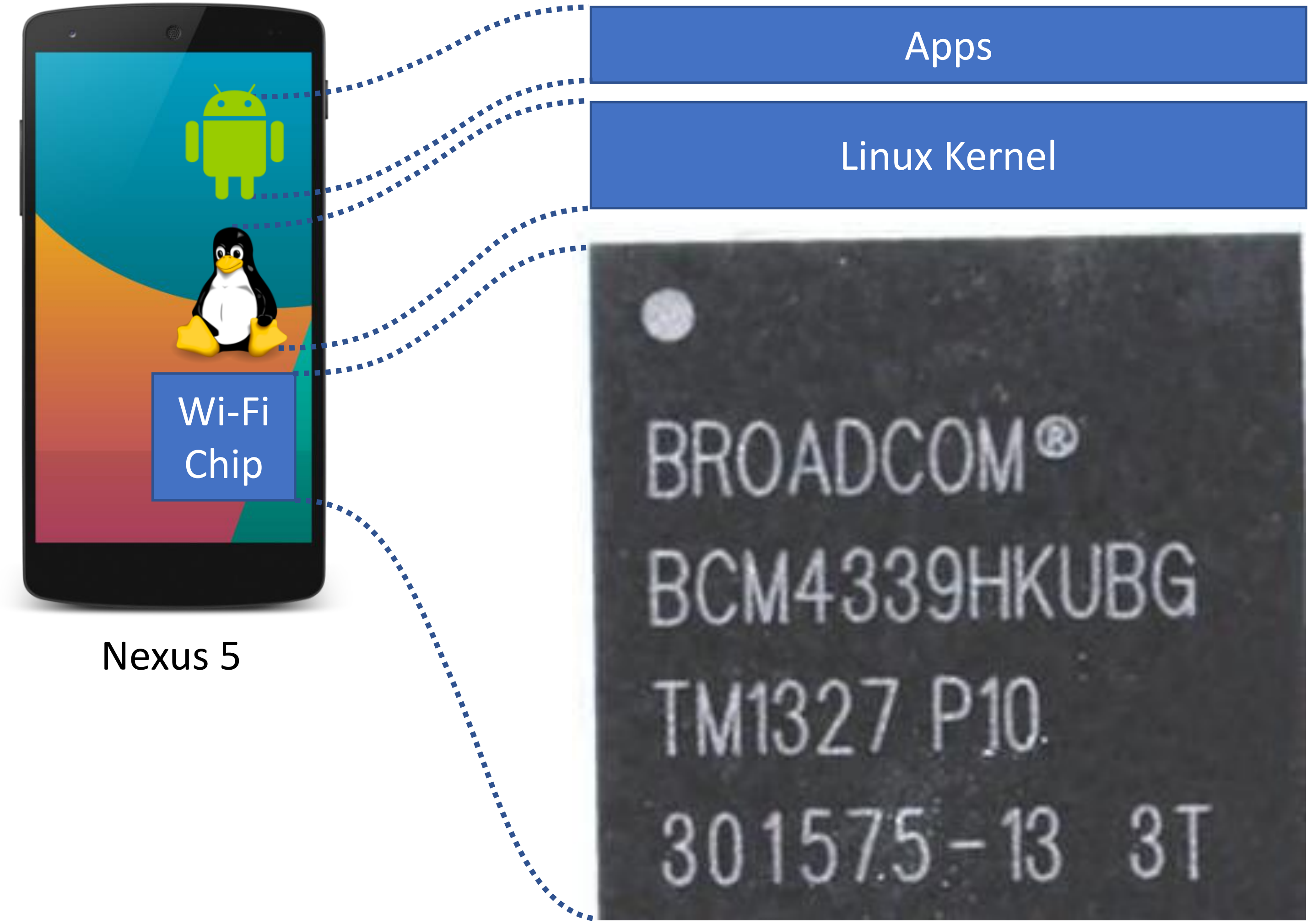


Apps

Linux Kernel

Nexus 5

Wi-Fi Chips in Smartphones



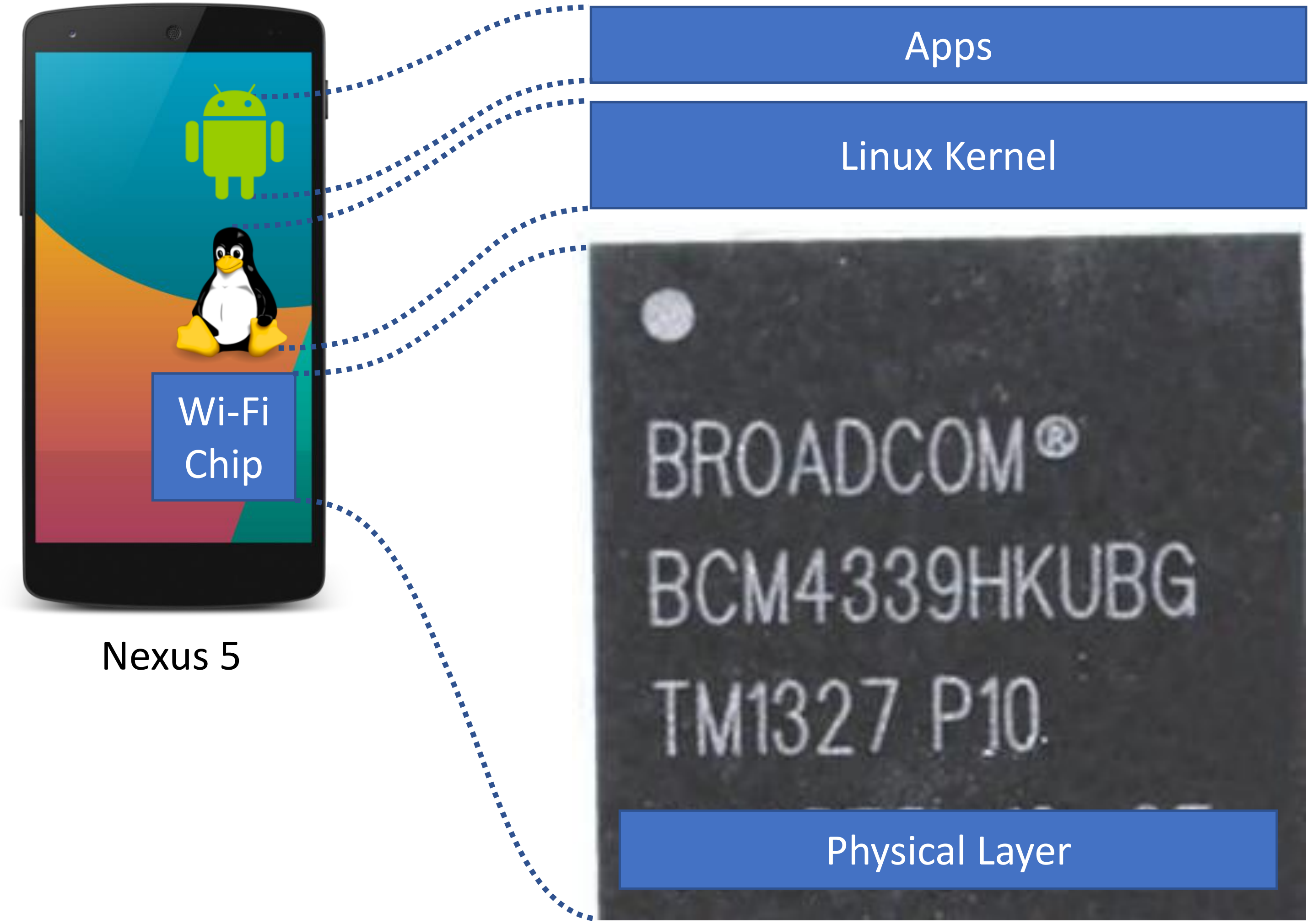
Apps

Linux Kernel

Wi-Fi Chip

Nexus 5

Wi-Fi Chips in Smartphones



Apps

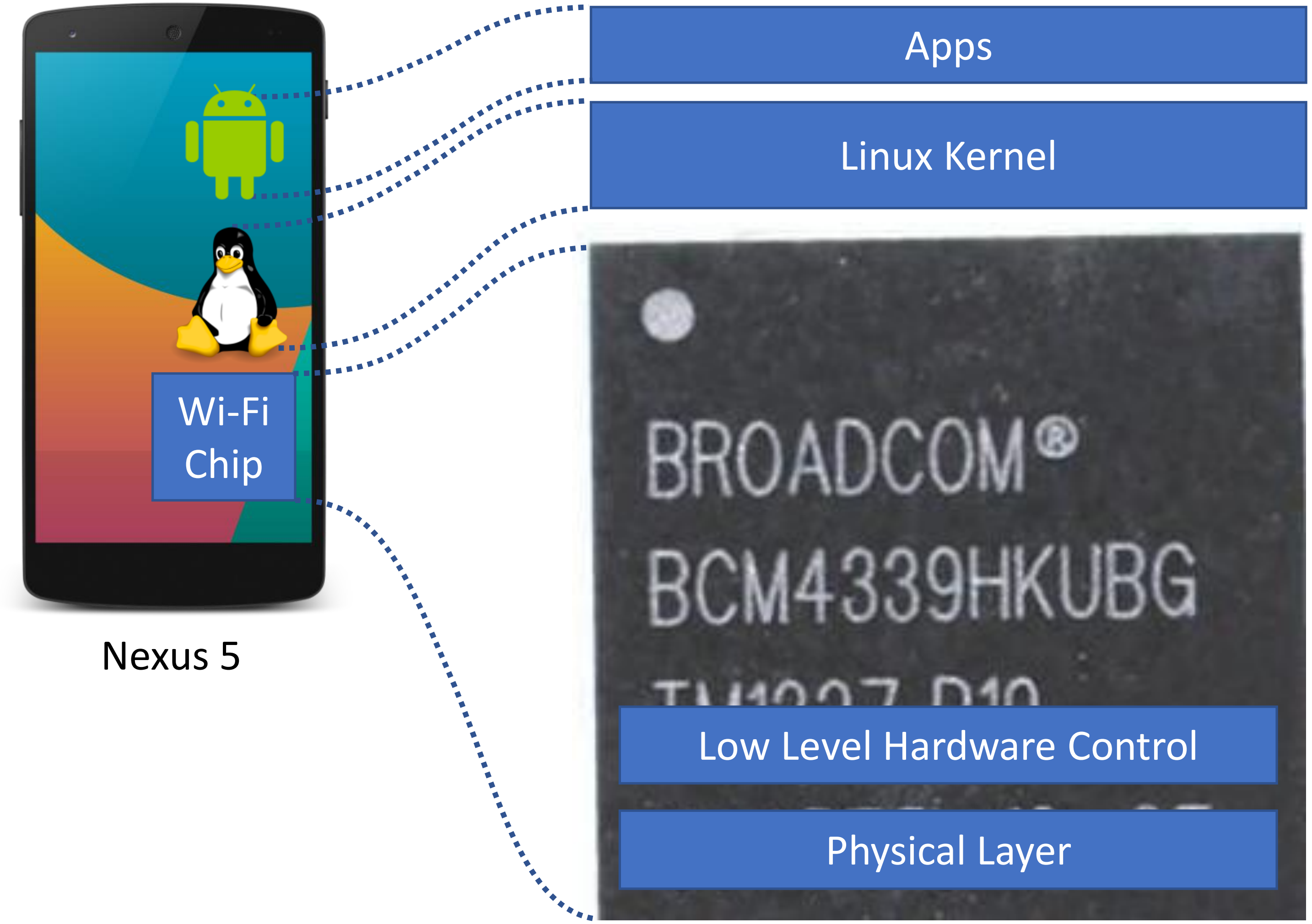
Linux Kernel

Physical Layer

Wi-Fi Chip

Nexus 5

Wi-Fi Chips in Smartphones



Apps

Linux Kernel

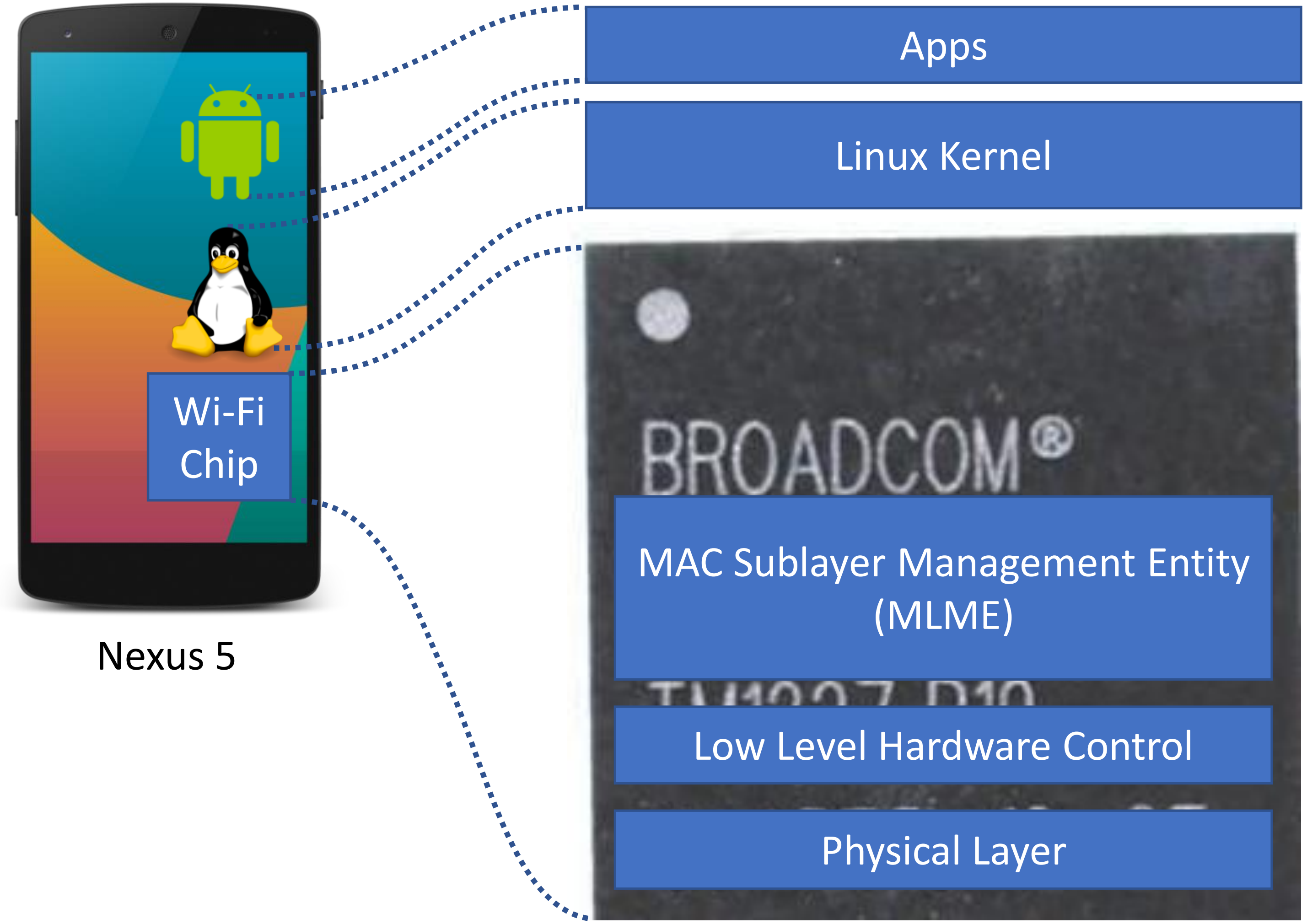
Low Level Hardware Control

Physical Layer

Wi-Fi Chip

Nexus 5

Wi-Fi Chips in Smartphones



Wi-Fi Chip

Nexus 5

Apps

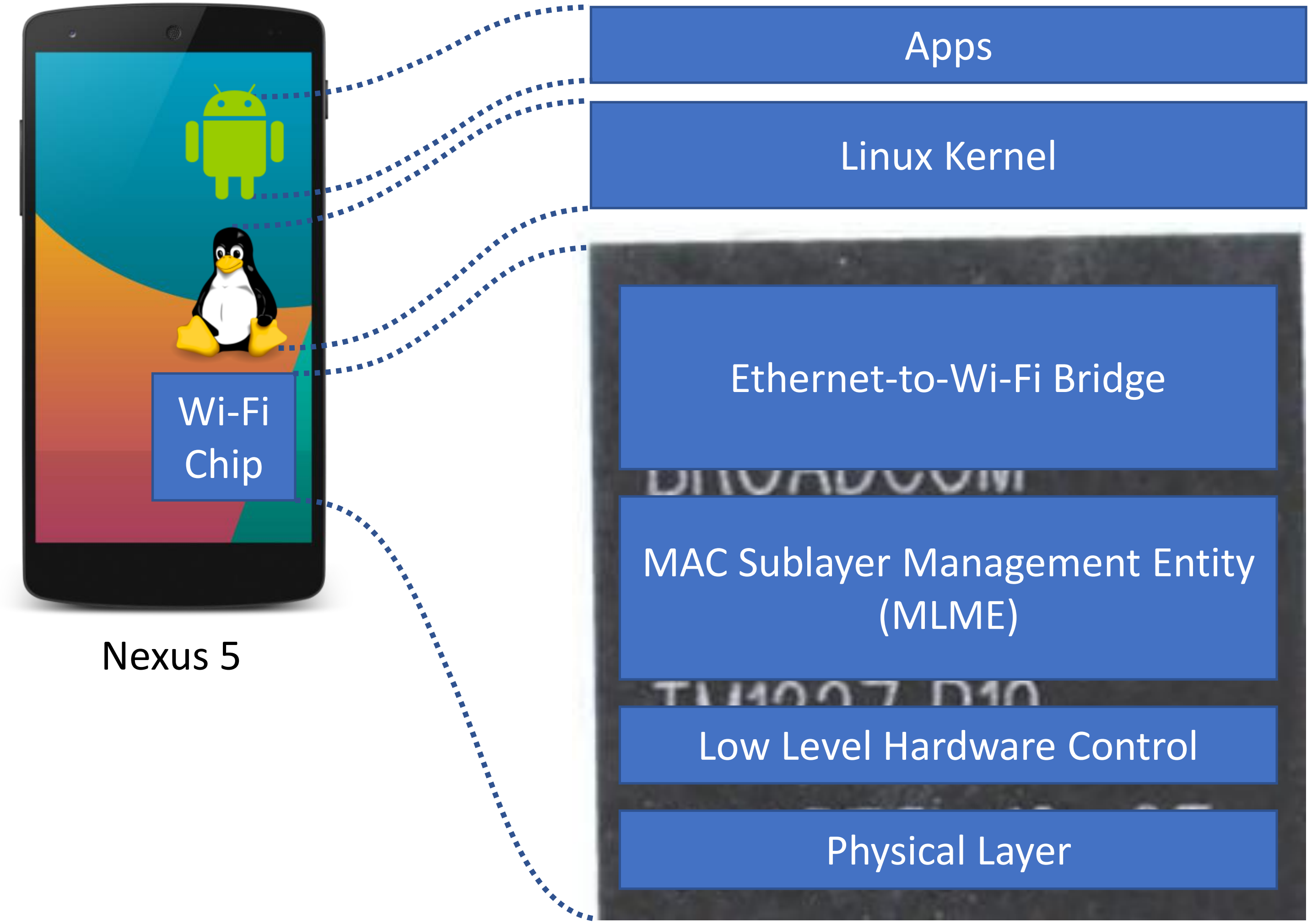
Linux Kernel

MAC Sublayer Management Entity (MLME)

Low Level Hardware Control

Physical Layer

Wi-Fi Chips in Smartphones



Wi-Fi Chip

Nexus 5

Apps

Linux Kernel

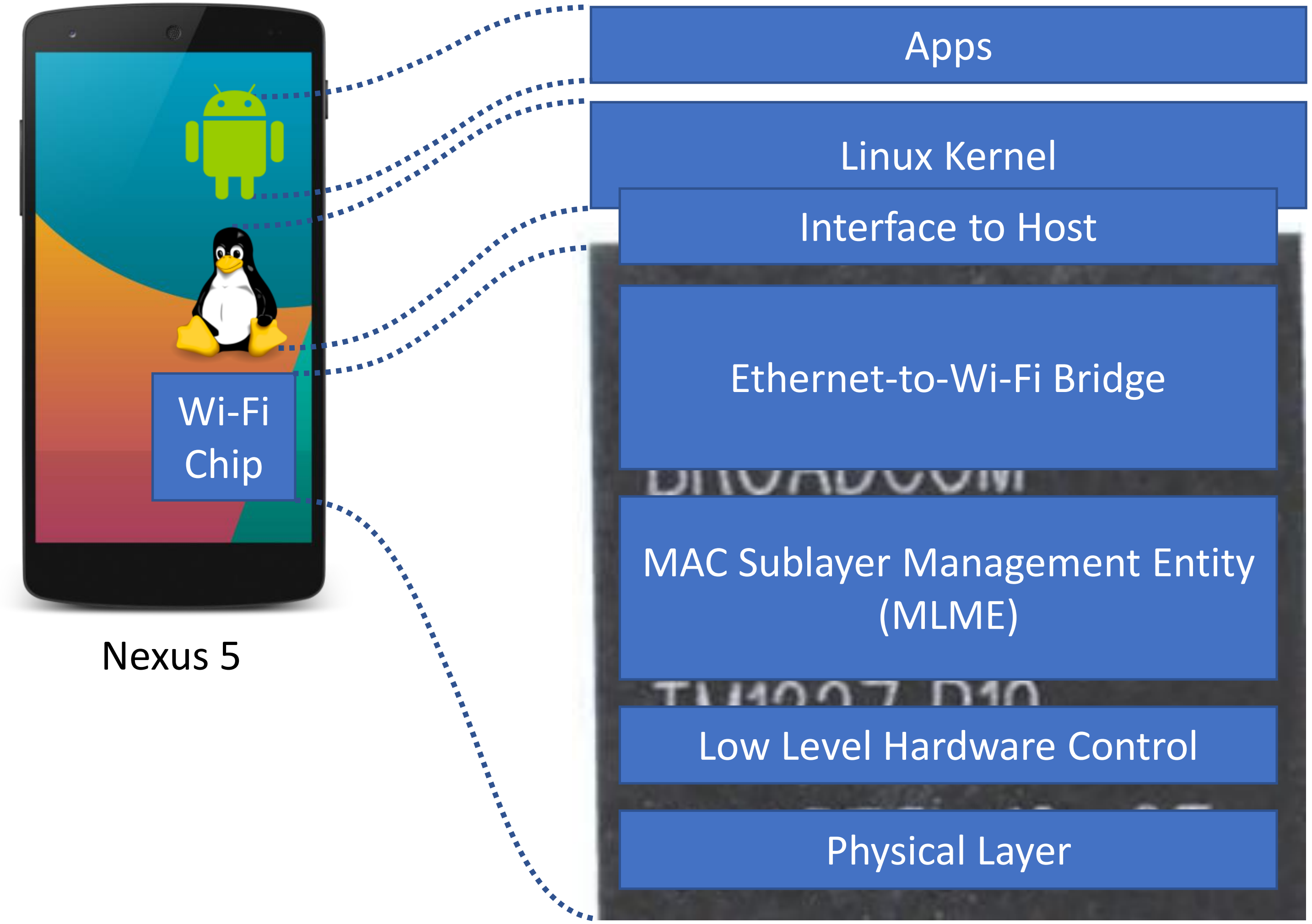
Ethernet-to-Wi-Fi Bridge

MAC Sublayer Management Entity (MLME)

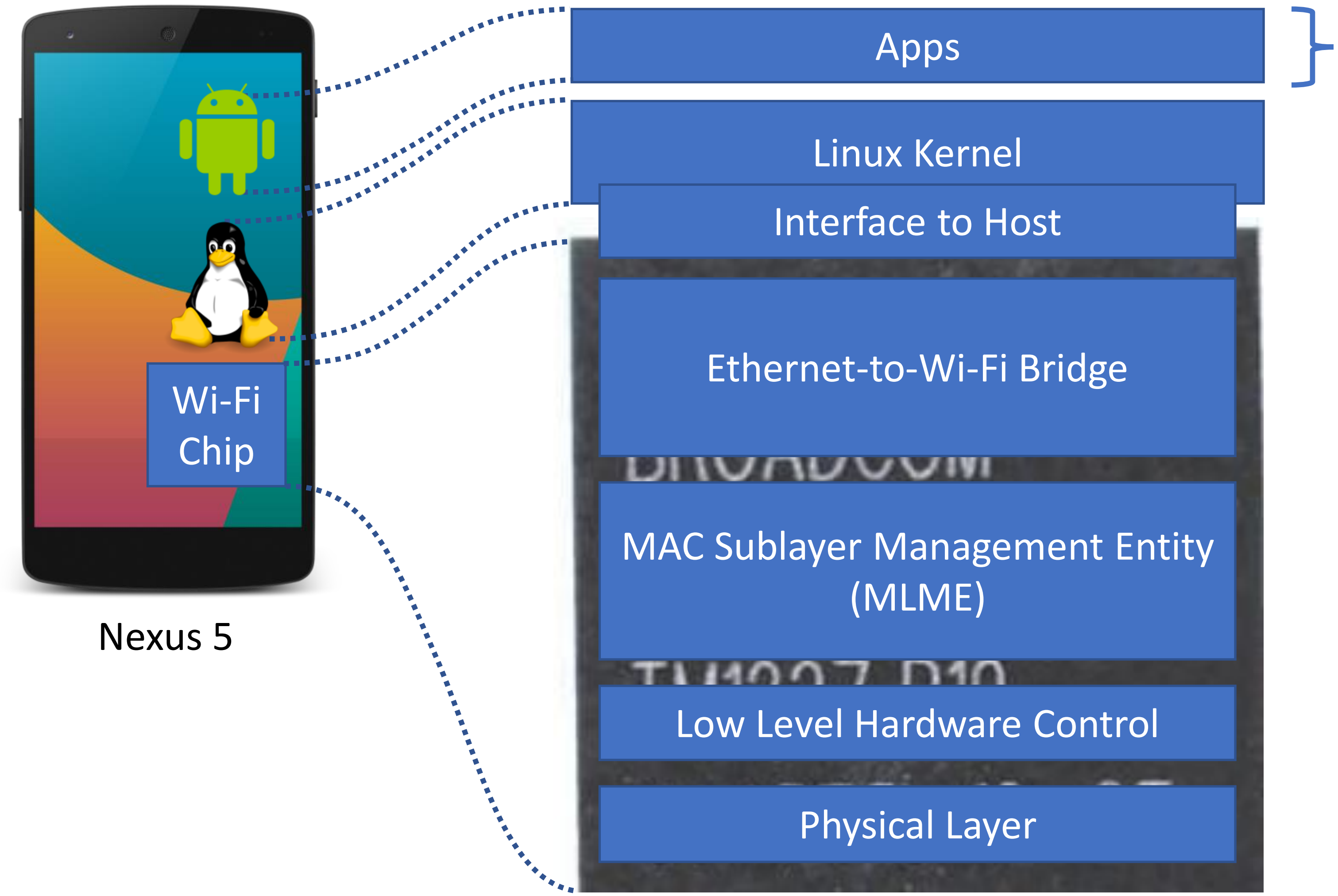
Low Level Hardware Control

Physical Layer

Wi-Fi Chips in Smartphones



Wi-Fi Chips in Smartphones



Wi-Fi Chip

Nexus 5

Apps

Generate TCP/UDP traffic

Linux Kernel

Interface to Host

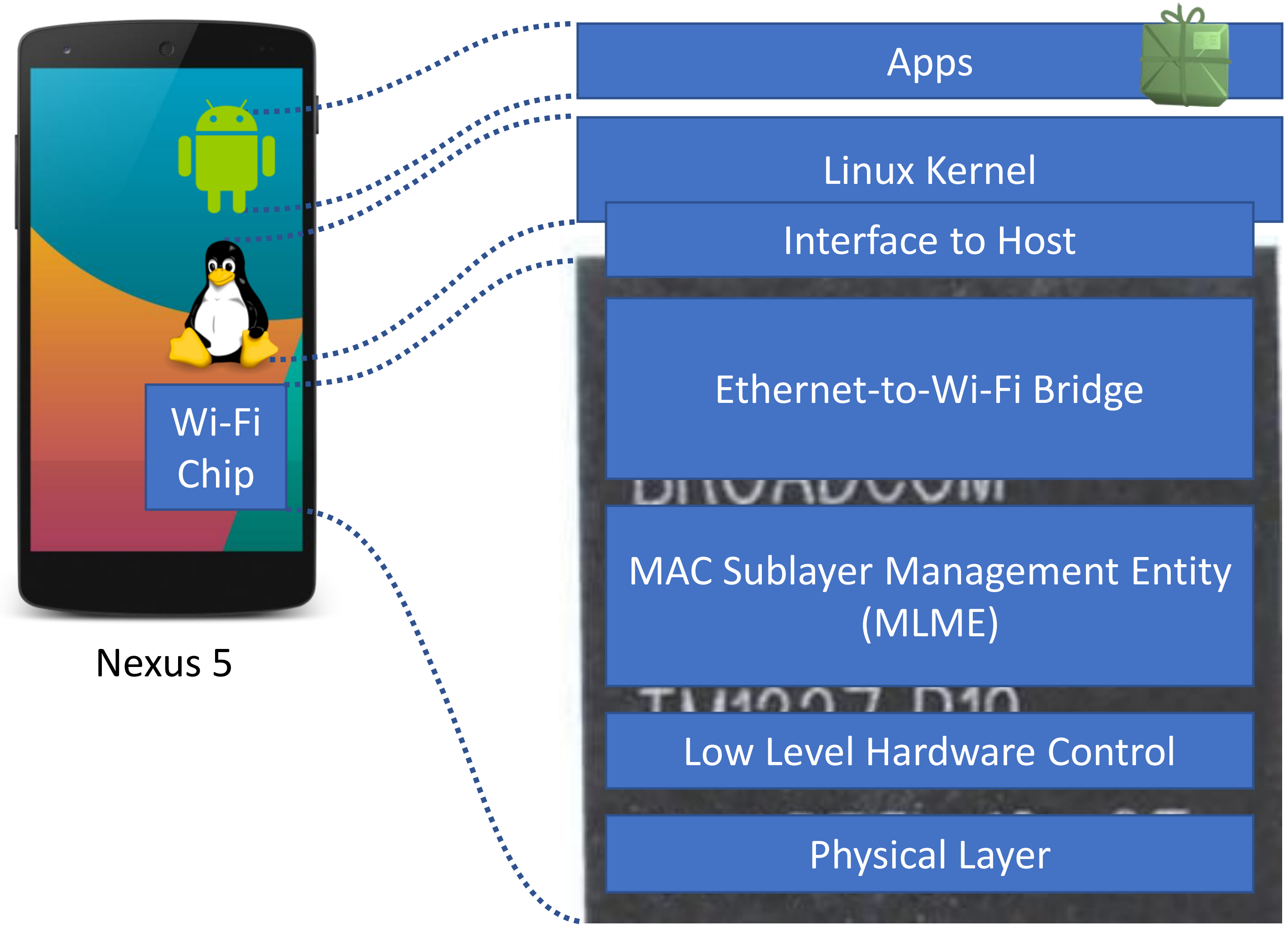
Ethernet-to-Wi-Fi Bridge

MAC Sublayer Management Entity (MLME)

Low Level Hardware Control

Physical Layer

Wi-Fi Chips in Smartphones



Nexus 5

Apps

Linux Kernel

Interface to Host

Ethernet-to-Wi-Fi Bridge

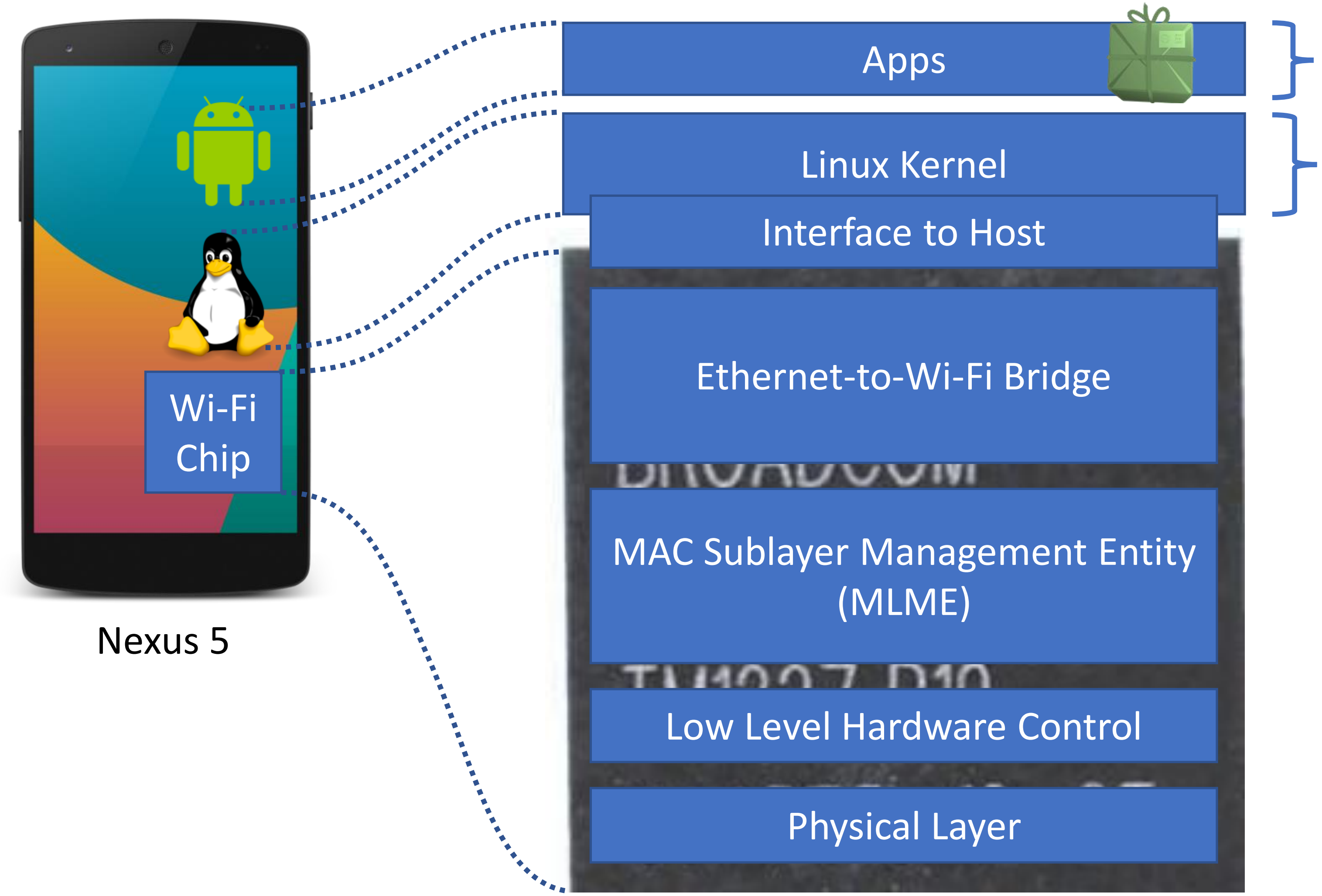
MAC Sublayer Management Entity (MLME)

Low Level Hardware Control

Physical Layer

Generate TCP/UDP traffic

Wi-Fi Chips in Smartphones

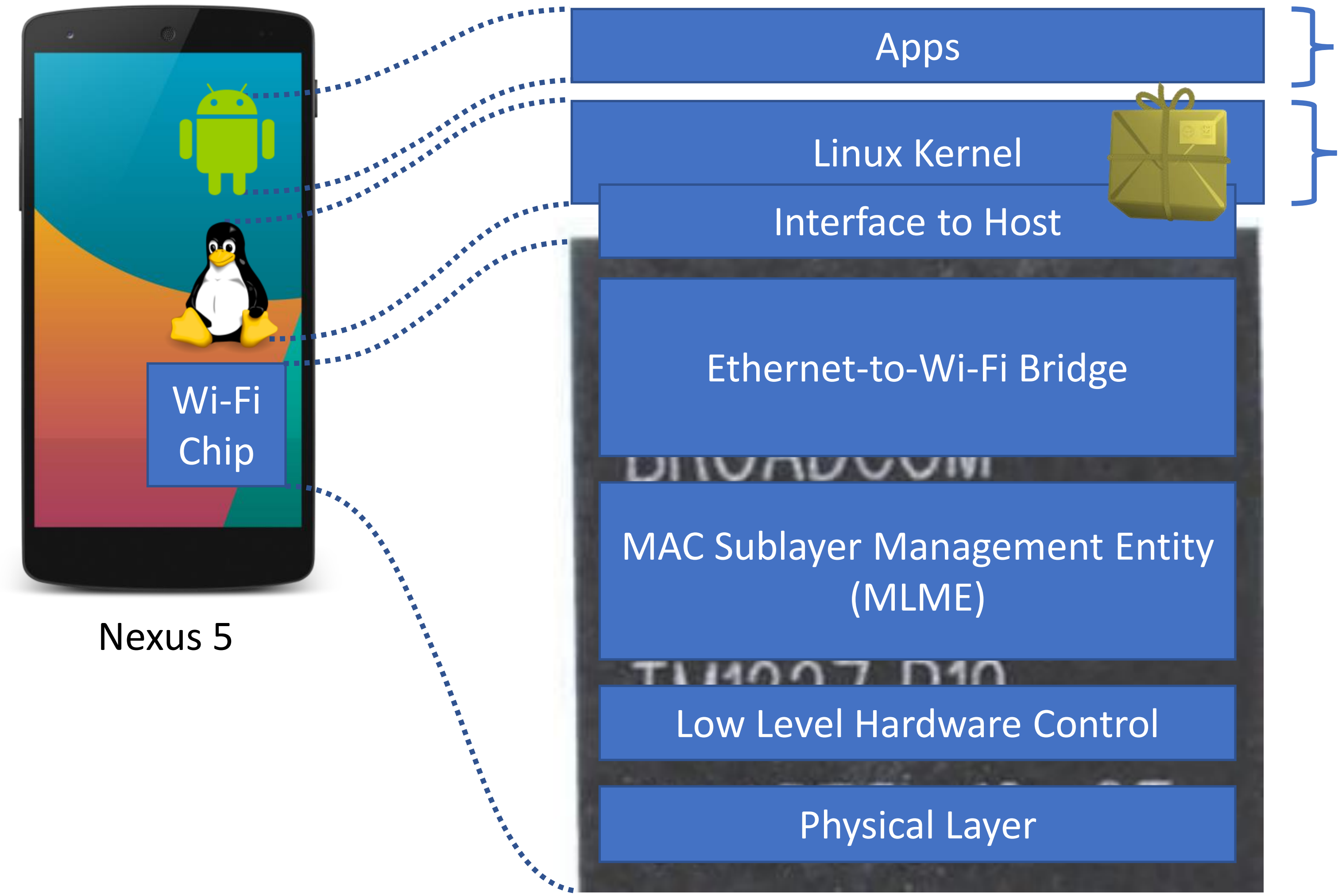


Nexus 5

Generate TCP/UDP traffic

Packs data in Ethernet frames

Wi-Fi Chips in Smartphones



Nexus 5

Wi-Fi Chip

Apps

Generate TCP/UDP traffic

Linux Kernel

Packs data in Ethernet frames

Interface to Host

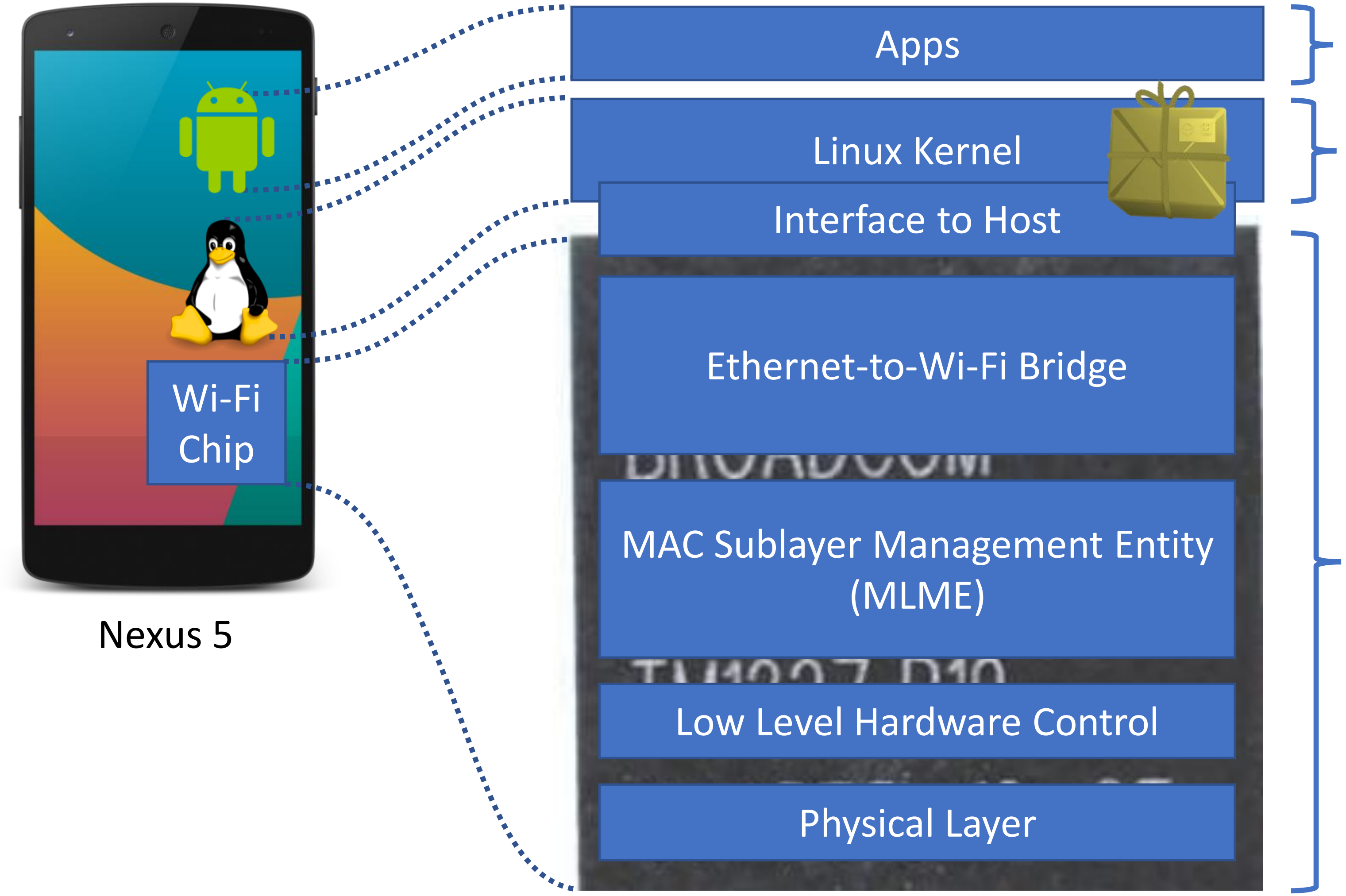
Ethernet-to-Wi-Fi Bridge

MAC Sublayer Management Entity (MLME)

Low Level Hardware Control

Physical Layer

Wi-Fi Chips in Smartphones



Nexus 5

Wi-Fi Chip

Apps

Linux Kernel

Interface to Host

Ethernet-to-Wi-Fi Bridge

MAC Sublayer Management Entity (MLME)

Low Level Hardware Control

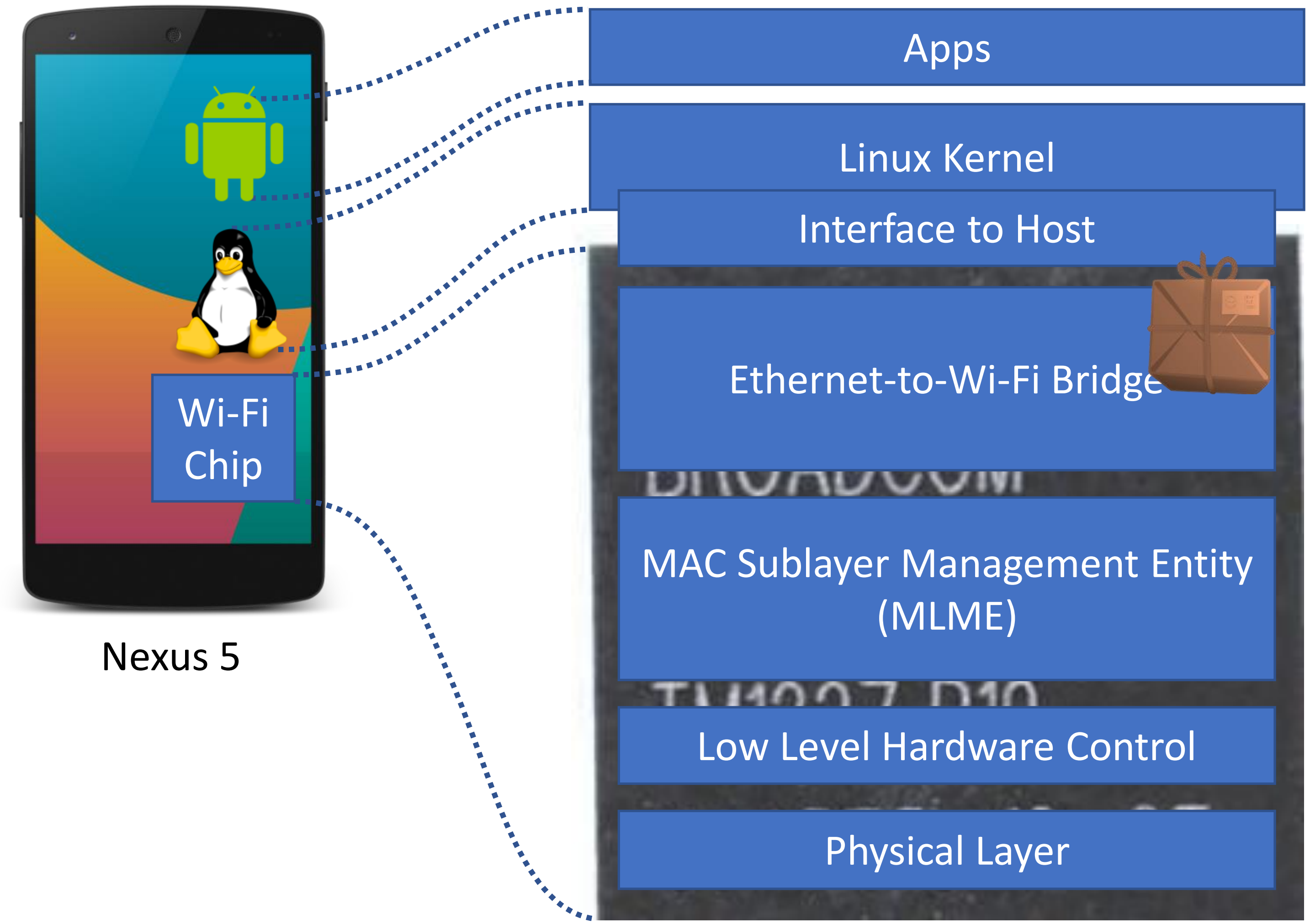
Physical Layer

Generate TCP/UDP traffic

Packs data in Ethernet frames

FullMAC firmware handles Wi-Fi connection and replaces Ethernet headers by Wi-Fi headers

Wi-Fi Chips in Smartphones



Nexus 5

Apps

Linux Kernel

Interface to Host

Ethernet-to-Wi-Fi Bridge

MAC Sublayer Management Entity (MLME)

Low Level Hardware Control

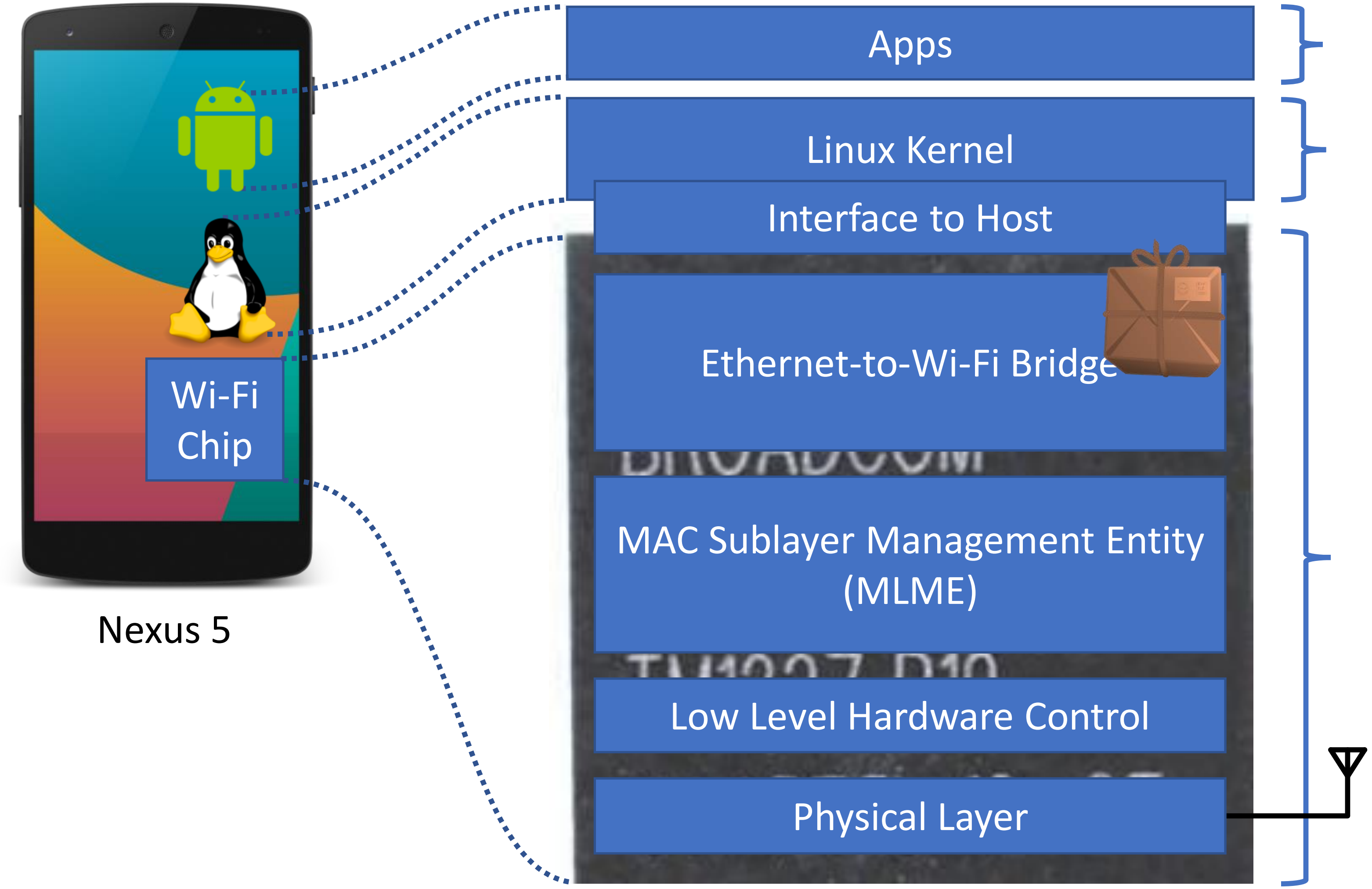
Physical Layer

Generate TCP/UDP traffic

Packs data in Ethernet frames

FullMAC firmware handles Wi-Fi connection and replaces Ethernet headers by Wi-Fi headers

Wi-Fi Chips in Smartphones



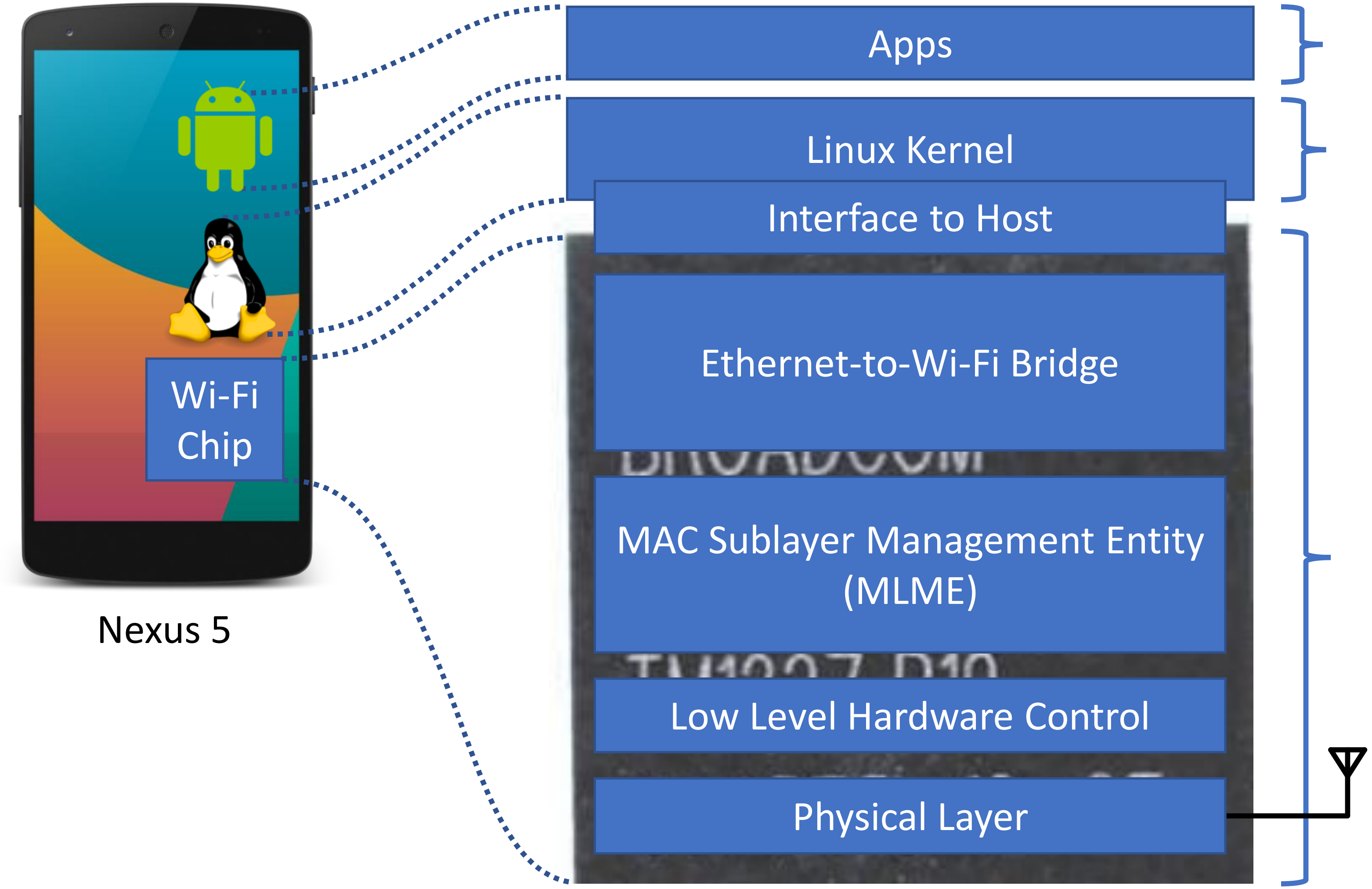
Nexus 5

Generate TCP/UDP traffic

Packs data in Ethernet frames

FullMAC firmware handles Wi-Fi connection and replaces Ethernet headers by Wi-Fi headers

Wi-Fi Chips in Smartphones

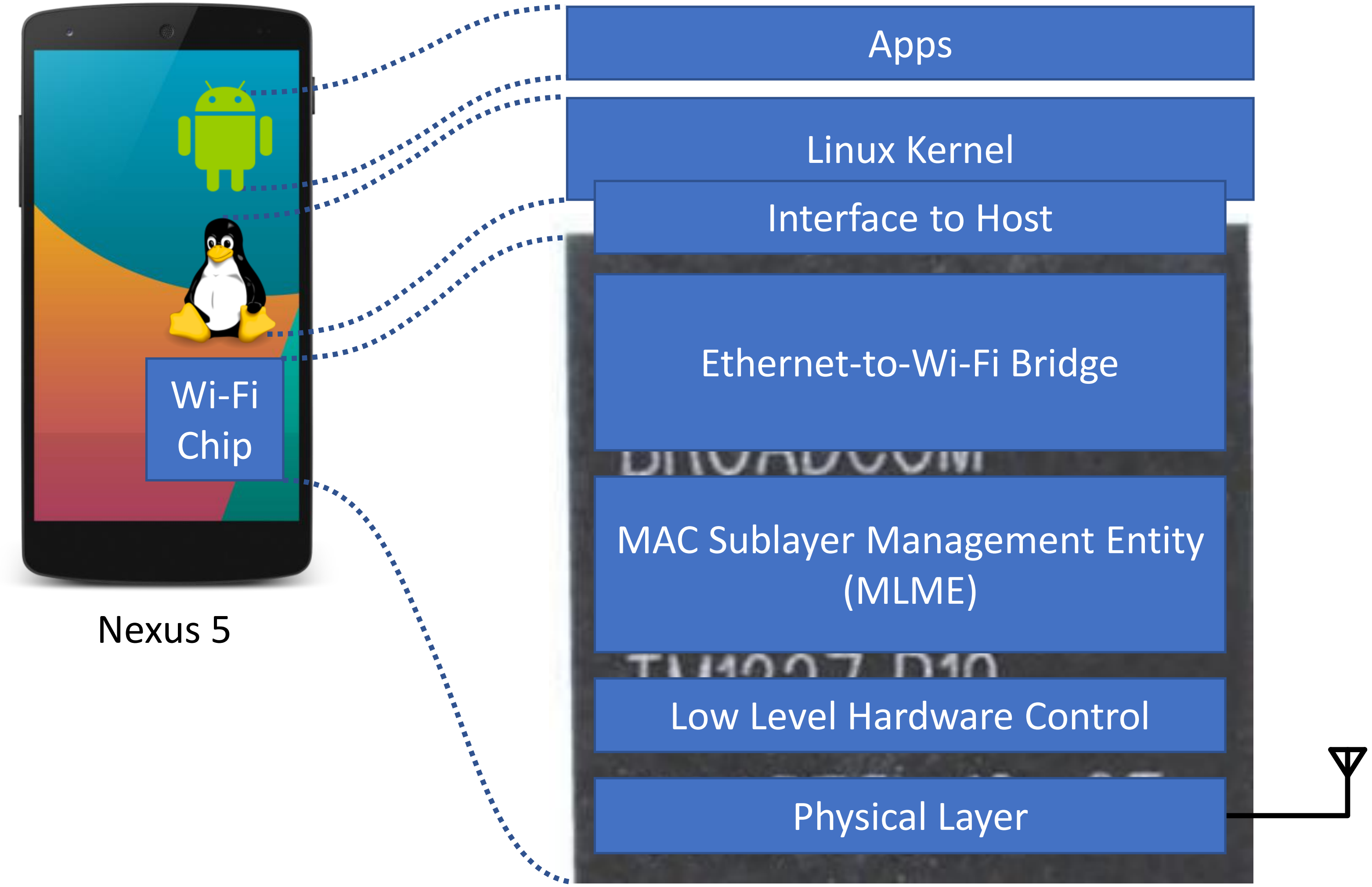


Generate TCP/UDP traffic

Packs data in Ethernet frames

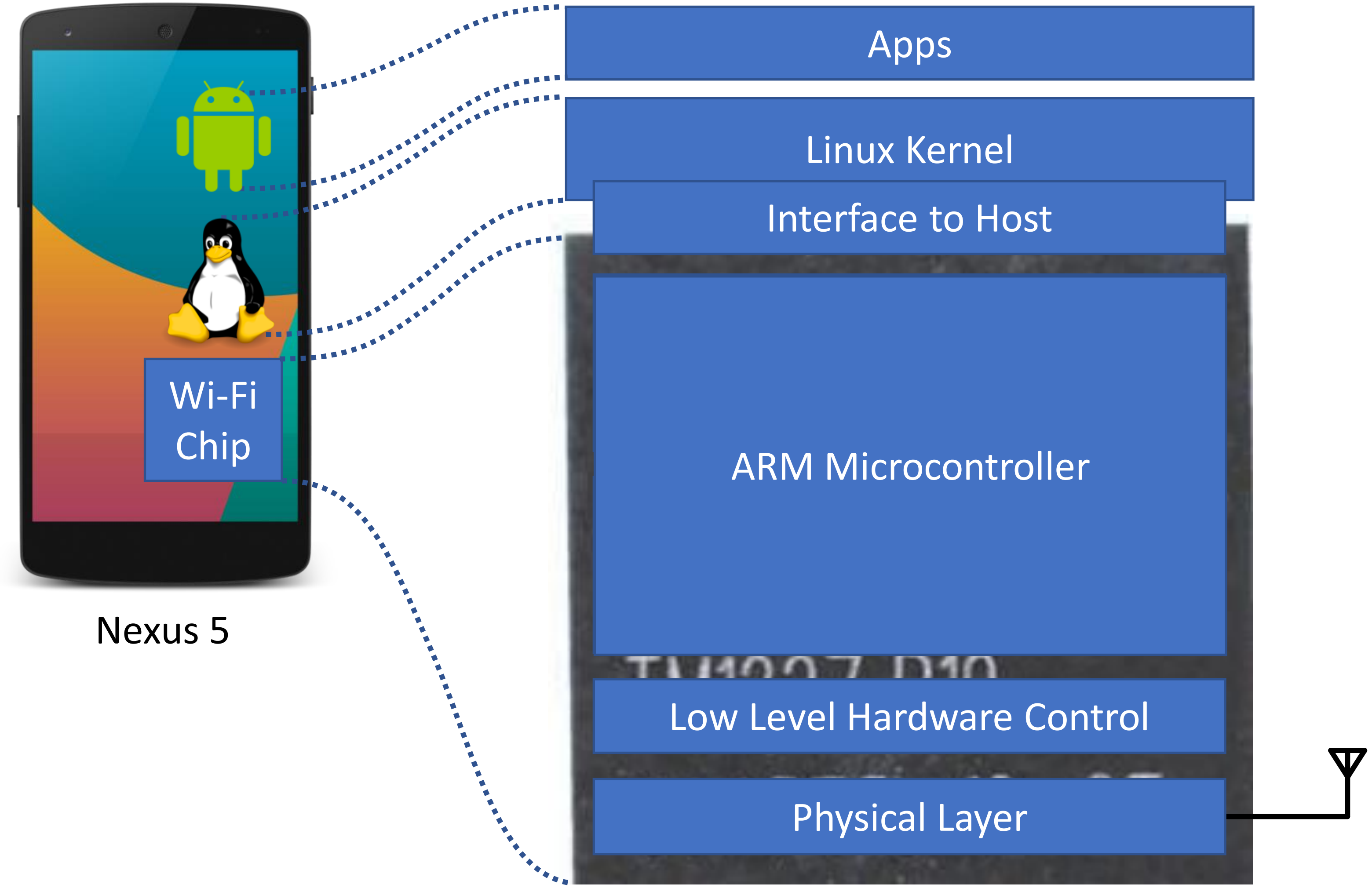
FullMAC firmware handles Wi-Fi connection and replaces Ethernet headers by Wi-Fi headers

Wi-Fi Firmware Handling

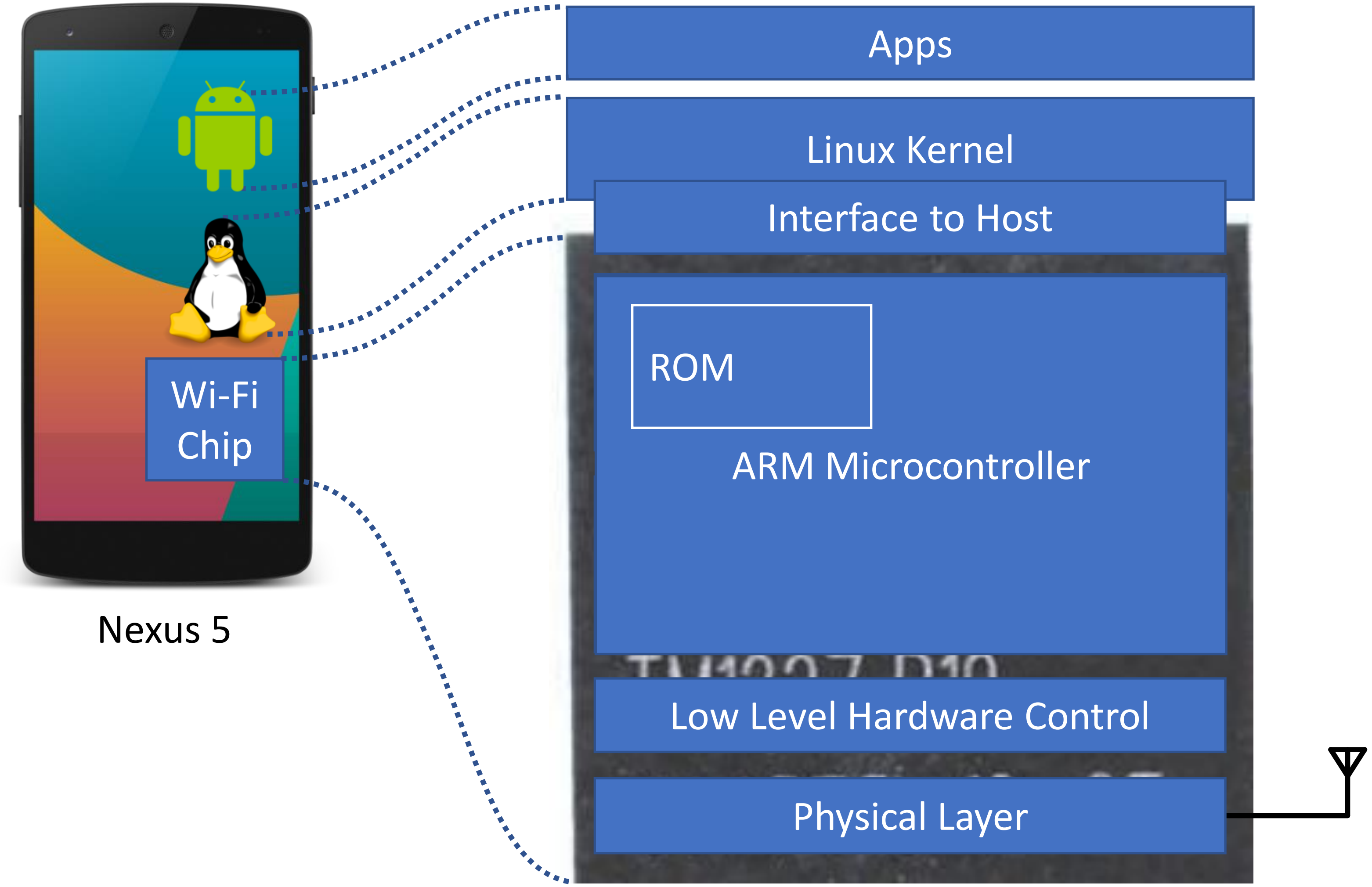


Nexus 5

Wi-Fi Firmware Handling



Wi-Fi Firmware Handling



Apps

Linux Kernel

Interface to Host

ROM

ARM Microcontroller

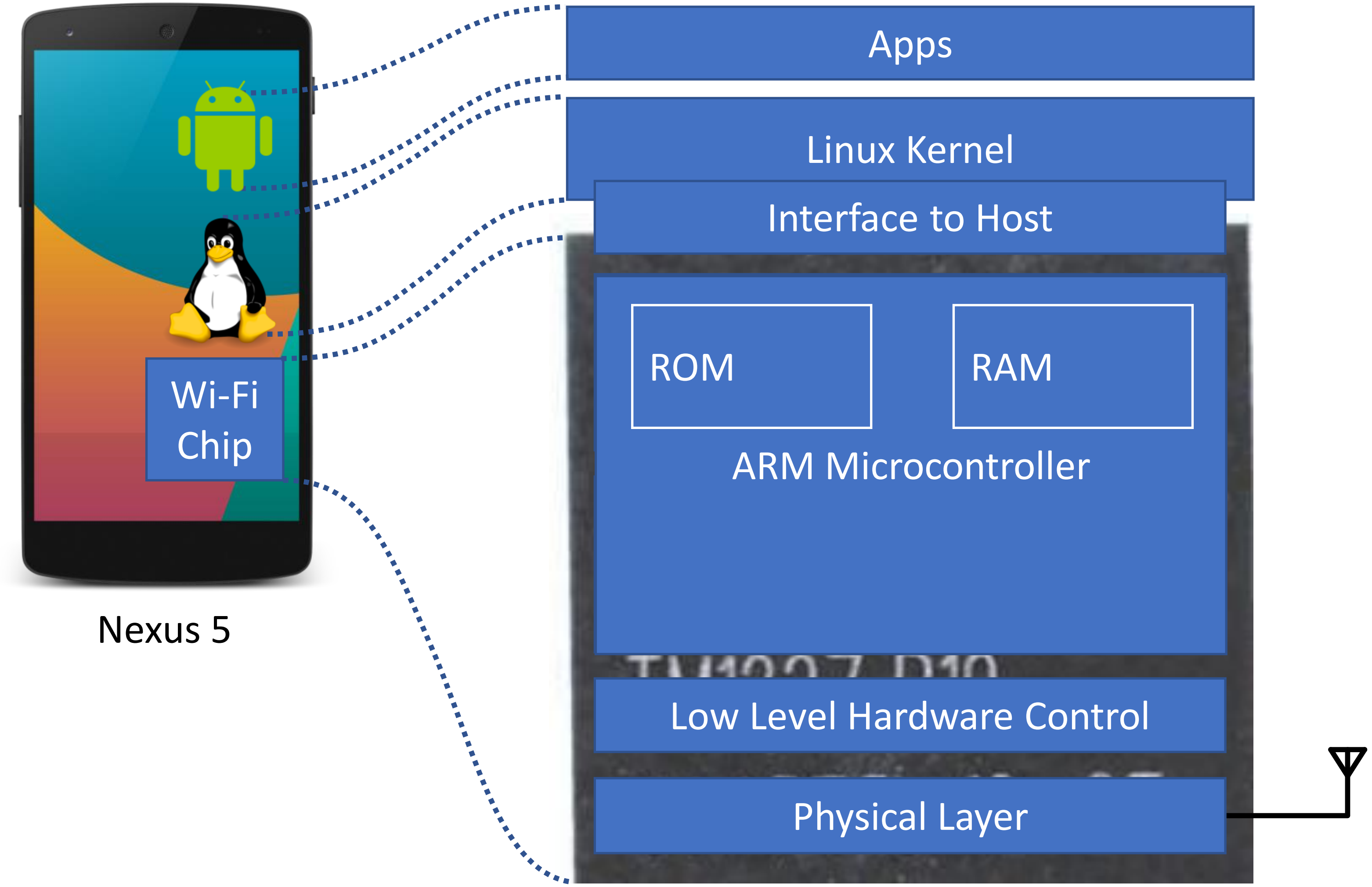
Low Level Hardware Control

Physical Layer

Wi-Fi Chip

Nexus 5

Wi-Fi Firmware Handling



Apps

Linux Kernel

Interface to Host

ROM

RAM

ARM Microcontroller

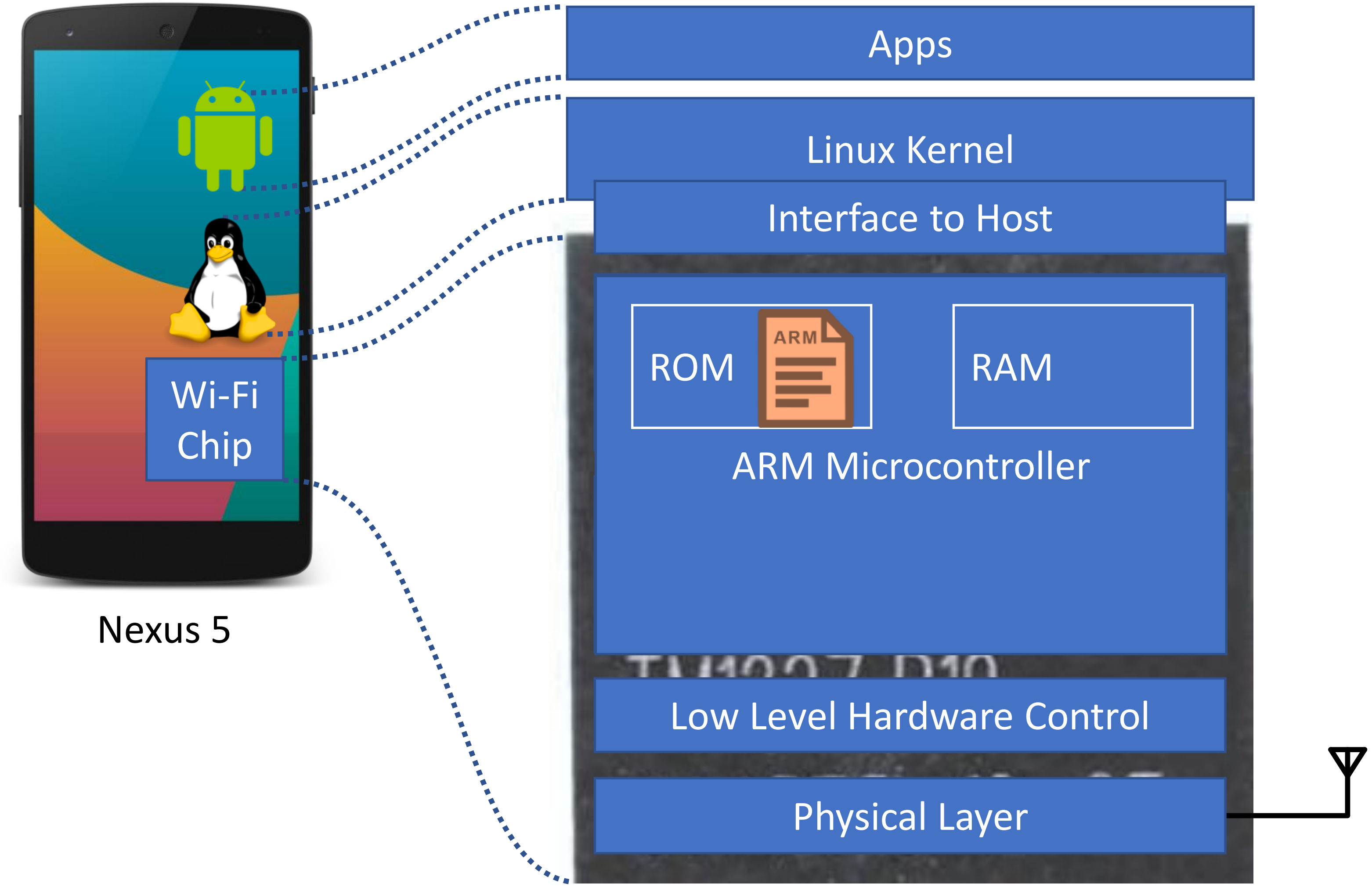
Low Level Hardware Control

Physical Layer

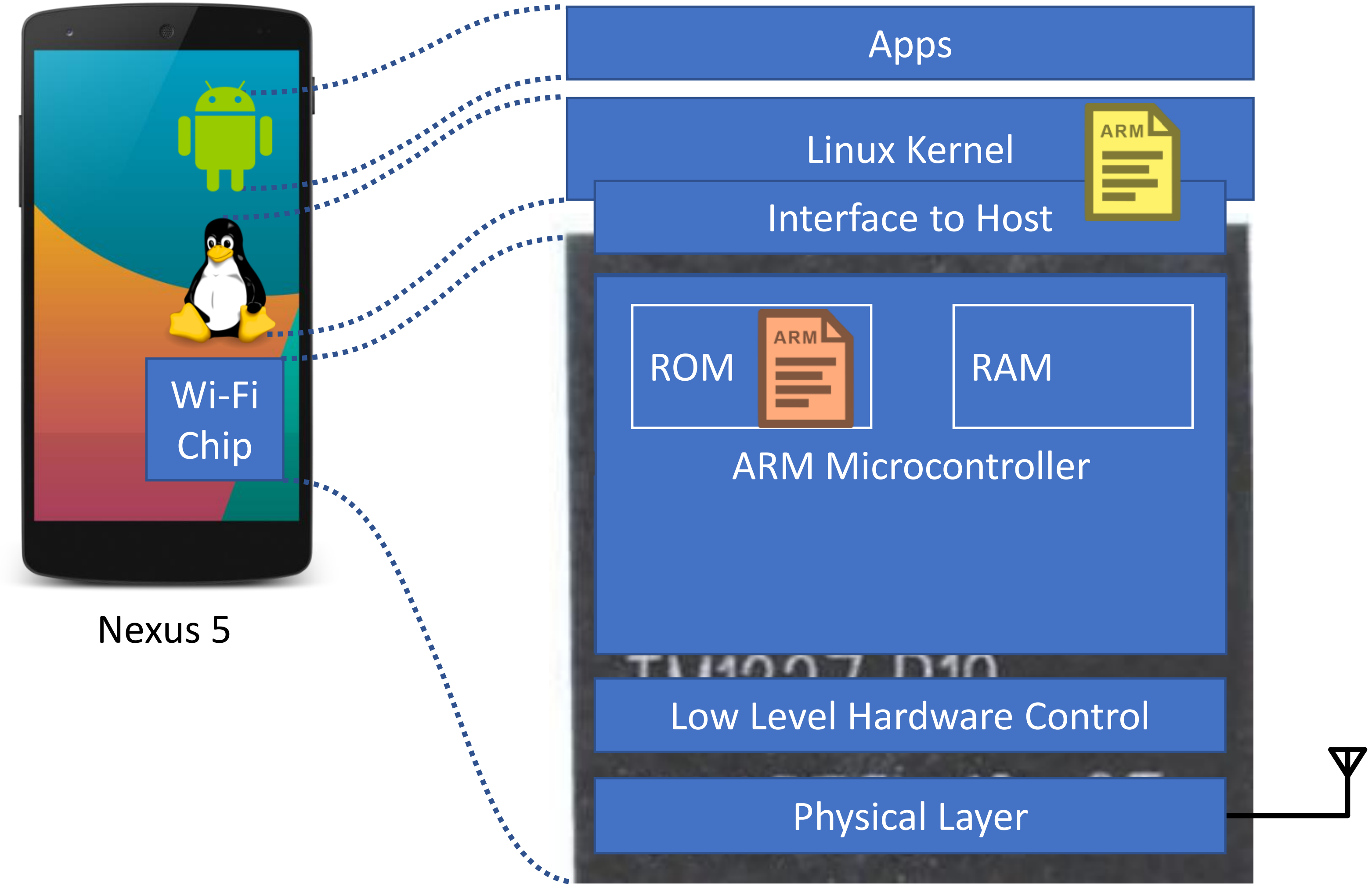
Wi-Fi Chip

Nexus 5

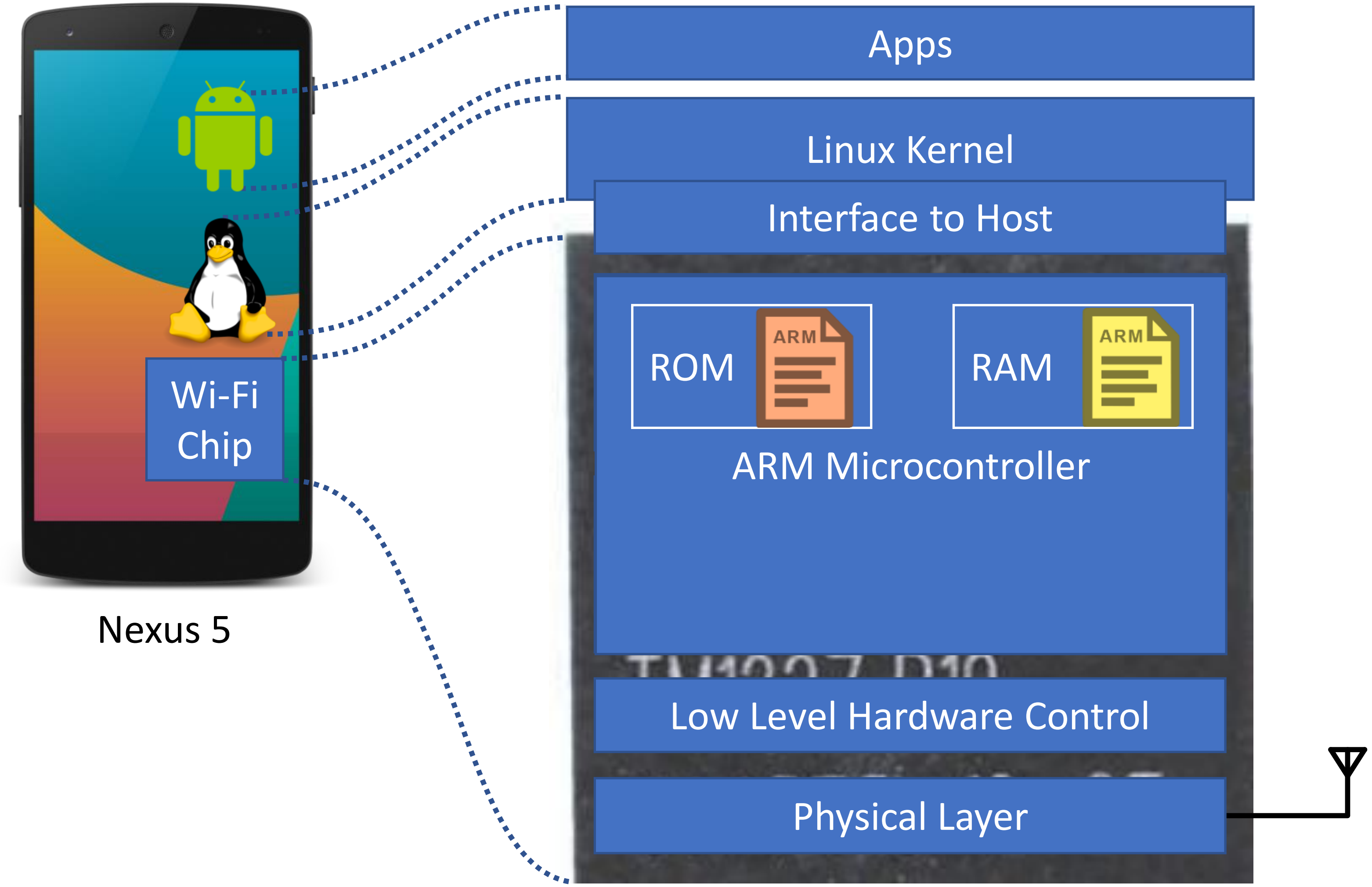
Wi-Fi Firmware Handling



Wi-Fi Firmware Handling



Firmware Analysis and Patching



Apps

Linux Kernel

Interface to Host

ROM

RAM

ARM Microcontroller

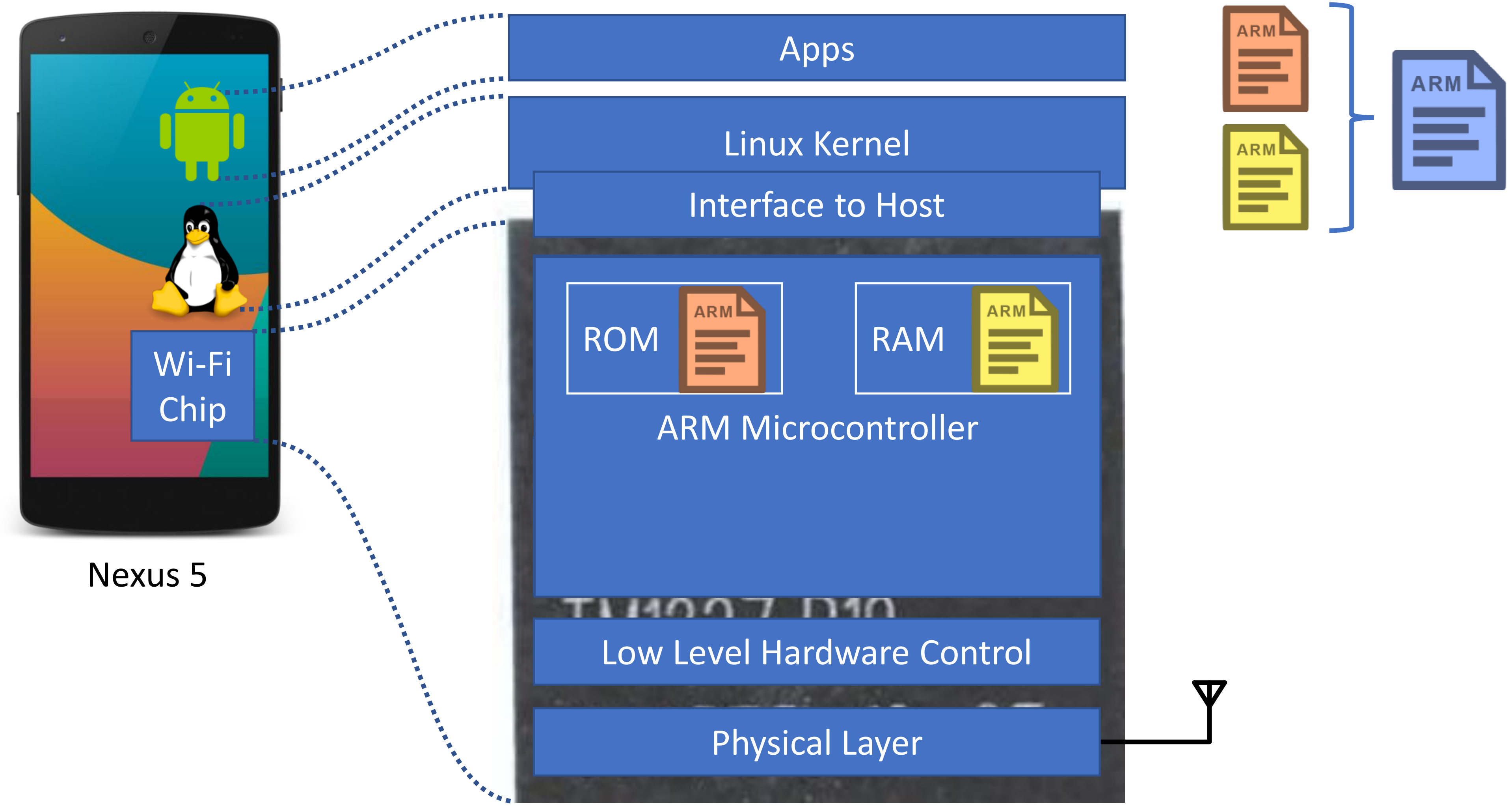
Low Level Hardware Control

Physical Layer

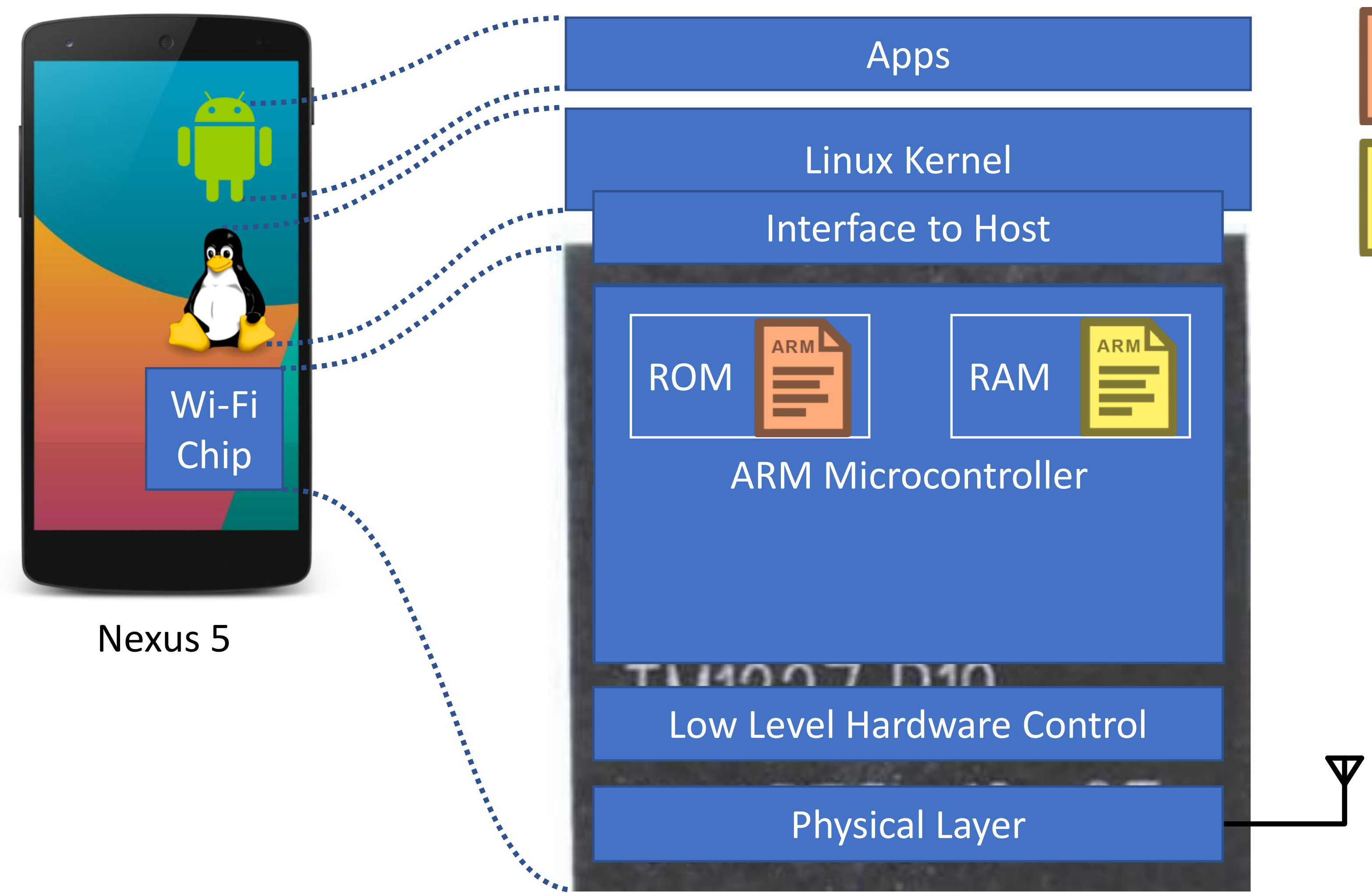
Wi-Fi Chip

Nexus 5

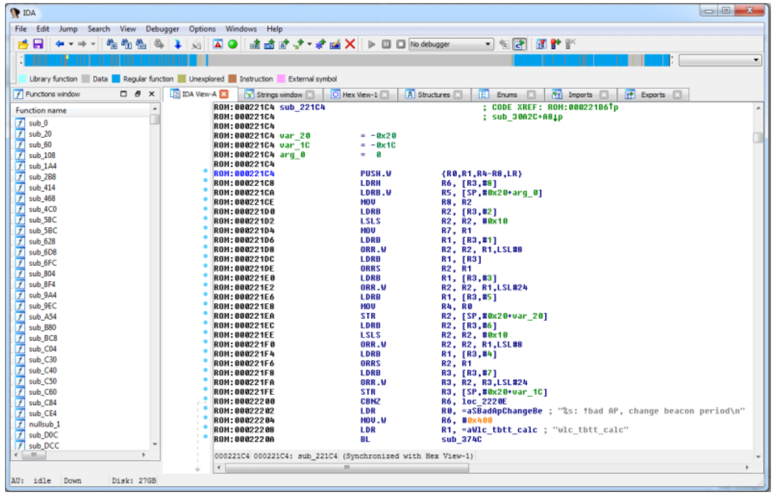
Firmware Analysis and Patching



Firmware Analysis and Patching

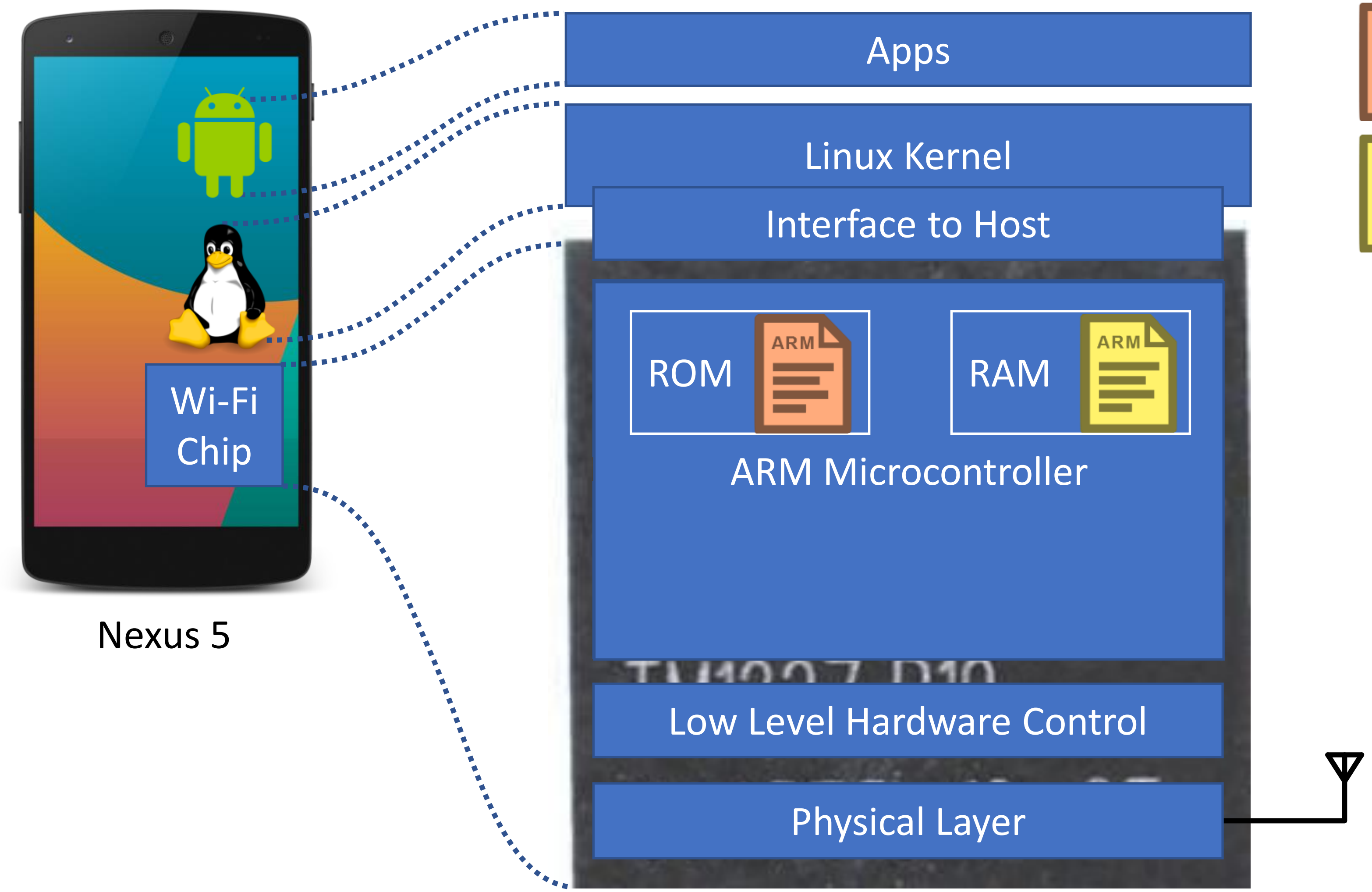


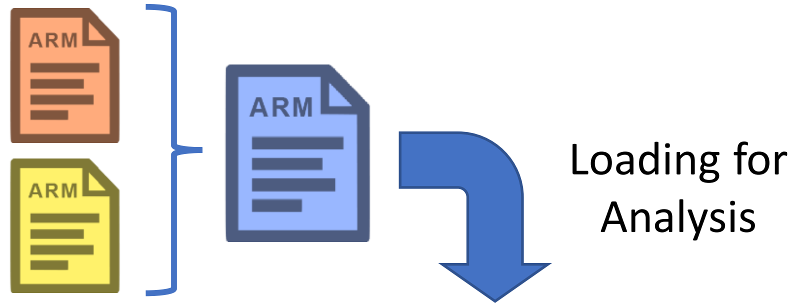
Loading for Analysis

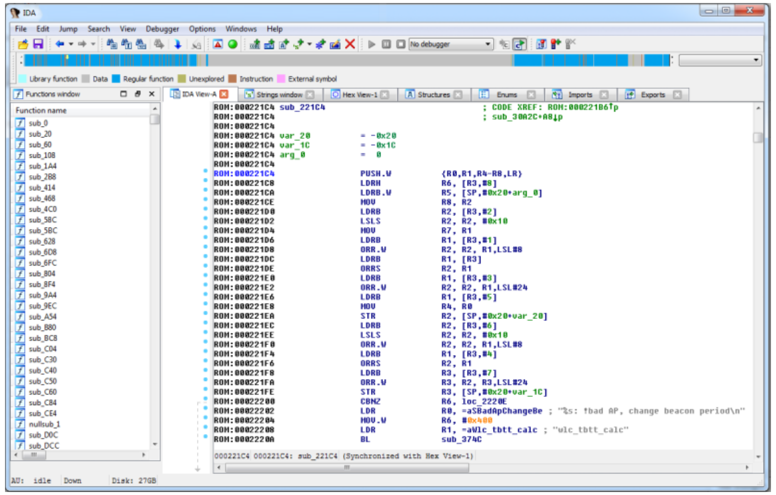


```
ROM:000221C4 sub_221C4                               ; CODE XREF: ROM:000221B6↑p
ROM:000221C4                                         ; sub_30A2C+A8↓p
ROM:000221C4
ROM:000221C4 var_20          = -0x20
ROM:000221C4 var_1C          = -0x1C
ROM:000221C4 arg_0           =  0
ROM:000221C4
ROM:000221C4                 PUSH.W          {R0,R1,R4-R8,LR}
ROM:000221C8                 LDRH            R6, [R3,#8]
ROM:000221CA                 LDRB.W          R5, [SP,#0x20+arg_0]
ROM:000221CE                 MOV             R8, R2
ROM:000221D0                 LDRB            R2, [R3,#2]
ROM:000221D2                 LSLS            R2, R2, #0x10
ROM:000221D4                 MOV             R7, R1
ROM:000221D6                 LDRB            R1, [R3,#1]
ROM:000221D8                 ORR.W           R2, R2, R1,LSL#8
ROM:000221DC                 LDRB            R1, [R3]
ROM:000221DE                 ORRS            R2, R1
ROM:000221E0                 LDRB            R1, [R3,#3]
ROM:000221E2                 ORR.W           R2, R2, R1,LSL#24
ROM:000221E6                 LDRB            R1, [R3,#5]
ROM:000221E8                 MOV             R4, R0
ROM:000221EA                 STR             R2, [SP,#0x20+var_20]
ROM:000221EC                 LDRB            R2, [R3,#6]
ROM:000221EE                 LSLS            R2, R2, #0x10
ROM:000221F0                 ORR.W           R2, R2, R1,LSL#8
ROM:000221F4                 LDRB            R1, [R3,#4]
ROM:000221F6                 ORRS            R2, R1
ROM:000221F8                 LDRB            R3, [R3,#7]
ROM:000221FA                 ORR.W           R3, R2, R3,LSL#24
ROM:000221FE                 STR             R3, [SP,#0x20+var_1C]
ROM:00022200                 CBNZ            R6, loc_2220E
ROM:00022202                 LDR             R0, =aSBadApChangeBe ; "%s: !bad AP, change beacon period\n"
ROM:00022204                 MOV.W           R6, #0x400
ROM:00022208                 LDR             R1, =aWlc_tbtt_calc ; "wlc_tbtt_calc"
ROM:0002220A                 BL              sub_374C
000221C4 000221C4: sub_221C4 (Synchronized with Hex View-1)
```

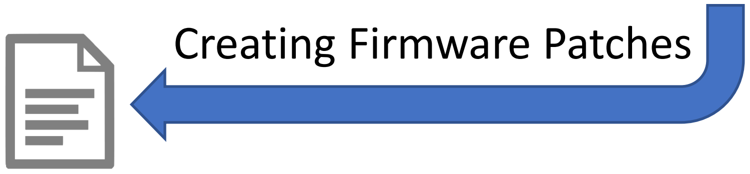
Firmware Analysis and Patching



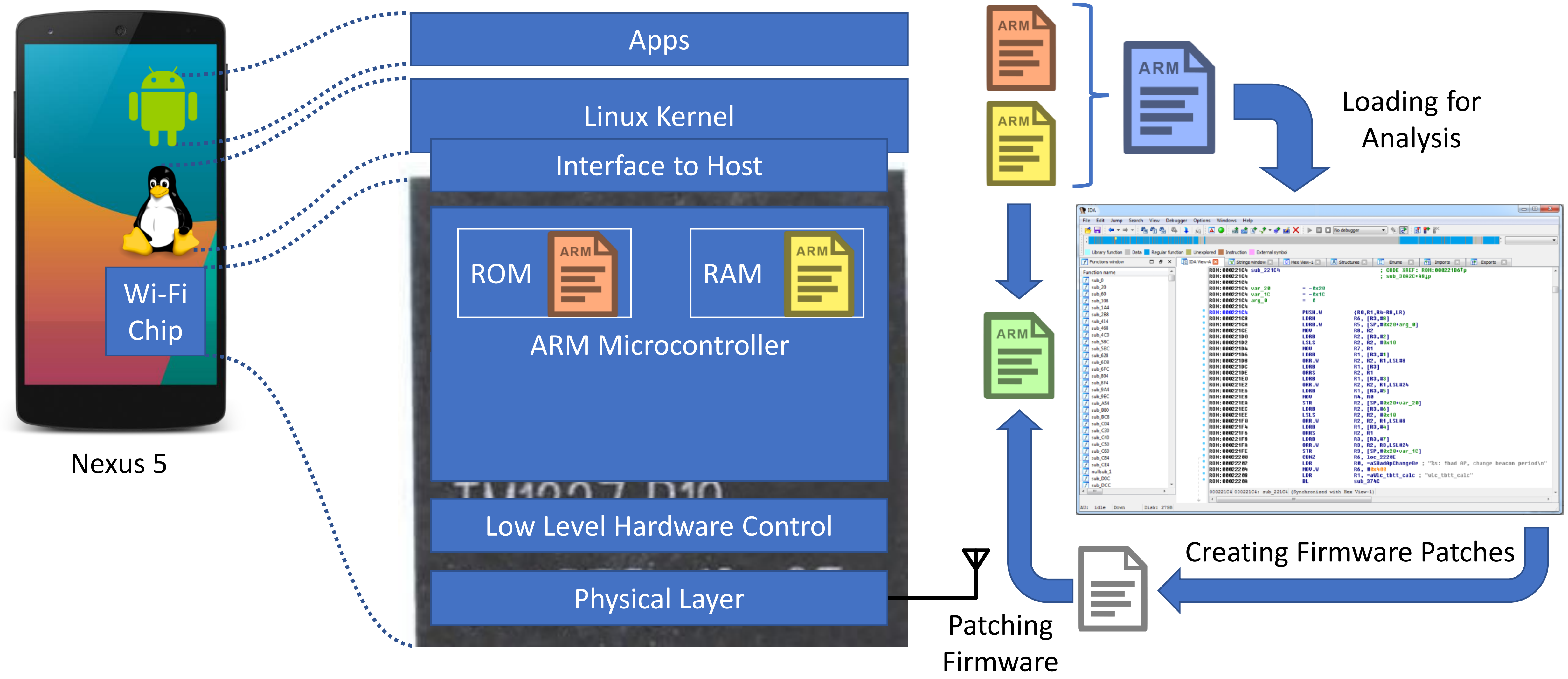




```
ROM:000221C4 sub_221C4                                  ; CODE XREF: ROM:000221B6↑p
ROM:000221C4                                            ; sub_30A2C+A8↓p
ROM:000221C4
ROM:000221C4 var_20          = -0x20
ROM:000221C4 var_1C          = -0x1C
ROM:000221C4 arg_0           =  0
ROM:000221C4
ROM:000221C4                 PUSH.W          {R0,R1,R4-R8,LR}
ROM:000221C8                 LDRH            R6, [R3,#8]
ROM:000221CA                 LDRB.W          R5, [SP,#0x20+arg_0]
ROM:000221CE                 MOV             R8, R2
ROM:000221D0                 LDRB            R2, [R3,#2]
ROM:000221D2                 LSLS            R2, R2, #0x10
ROM:000221D4                 MOV             R7, R1
ROM:000221D6                 LDRB            R1, [R3,#1]
ROM:000221D8                 ORR.W           R2, R2, R1,LSL#8
ROM:000221DC                 LDRB            R1, [R3]
ROM:000221DE                 ORRS            R2, R1
ROM:000221E0                 LDRB            R1, [R3,#3]
ROM:000221E2                 ORR.W           R2, R2, R1,LSL#24
ROM:000221E6                 LDRB            R1, [R3,#5]
ROM:000221E8                 MOV             R4, R0
ROM:000221EA                 STR             R2, [SP,#0x20+var_20]
ROM:000221EC                 LDRB            R2, [R3,#6]
ROM:000221EE                 LSLS            R2, R2, #0x10
ROM:000221F0                 ORR.W           R2, R2, R1,LSL#8
ROM:000221F4                 LDRB            R1, [R3,#4]
ROM:000221F6                 ORRS            R2, R1
ROM:000221F8                 LDRB            R3, [R3,#7]
ROM:000221FA                 ORR.W           R3, R2, R3,LSL#24
ROM:000221FE                 STR             R3, [SP,#0x20+var_1C]
ROM:00022200                 CBNZ            R6, loc_2220E
ROM:00022202                 LDR             R0, =aSBadApChangeBe ; "%s: !bad AP, change beacon period\n"
ROM:00022204                 MOV.W           R6, #0x400
ROM:00022208                 LDR             R1, =aWlc_tbtt_calc ; "wlc_tbtt_calc"
ROM:0002220A                 BL              sub_374C
000221C4 000221C4: sub_221C4 (Synchronized with Hex View-1)
```



Firmware Analysis and Patching



Nexus 5

Wi-Fi Chip

Apps

Linux Kernel

Interface to Host

ROM

ARM

RAM

ARM

ARM Microcontroller

Low Level Hardware Control

Physical Layer

ARM

ARM

ARM

Loading for Analysis

ARM

Creating Firmware Patches

Patching Firmware

Firmware Analysis and Patching

Apps

Related Work

One firmware to monitor 'em all

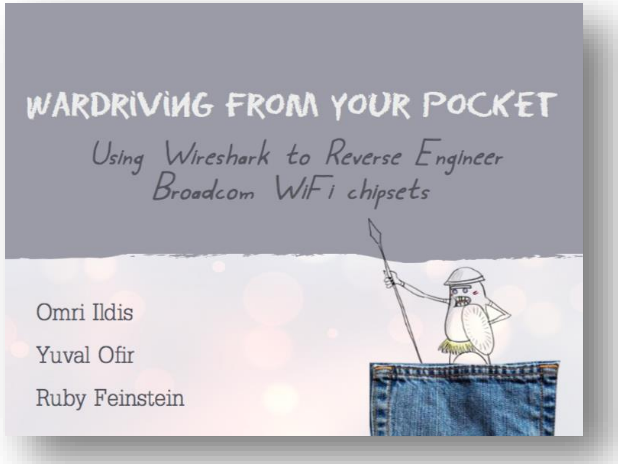
Andrés Blanco - Matías Eissler

CORE SECURITY

HACK.LU

Thursday, October 25, 12

monmob



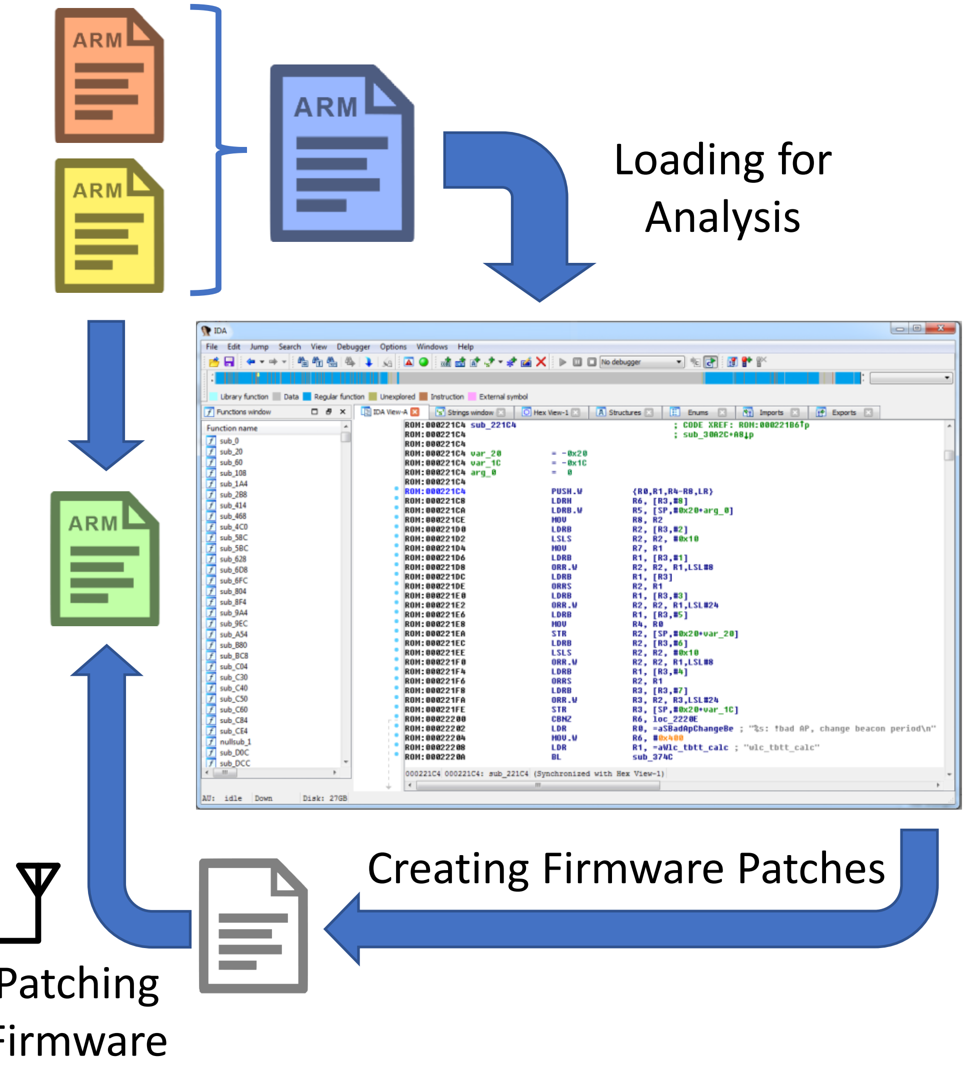
WARDRIVING FROM YOUR POCKET

Using Wireshark to Reverse Engineer Broadcom WiFi chipsets

Omri Ildis
Yuval Ofir
Ruby Feinstein

bcmon

Physical Layer

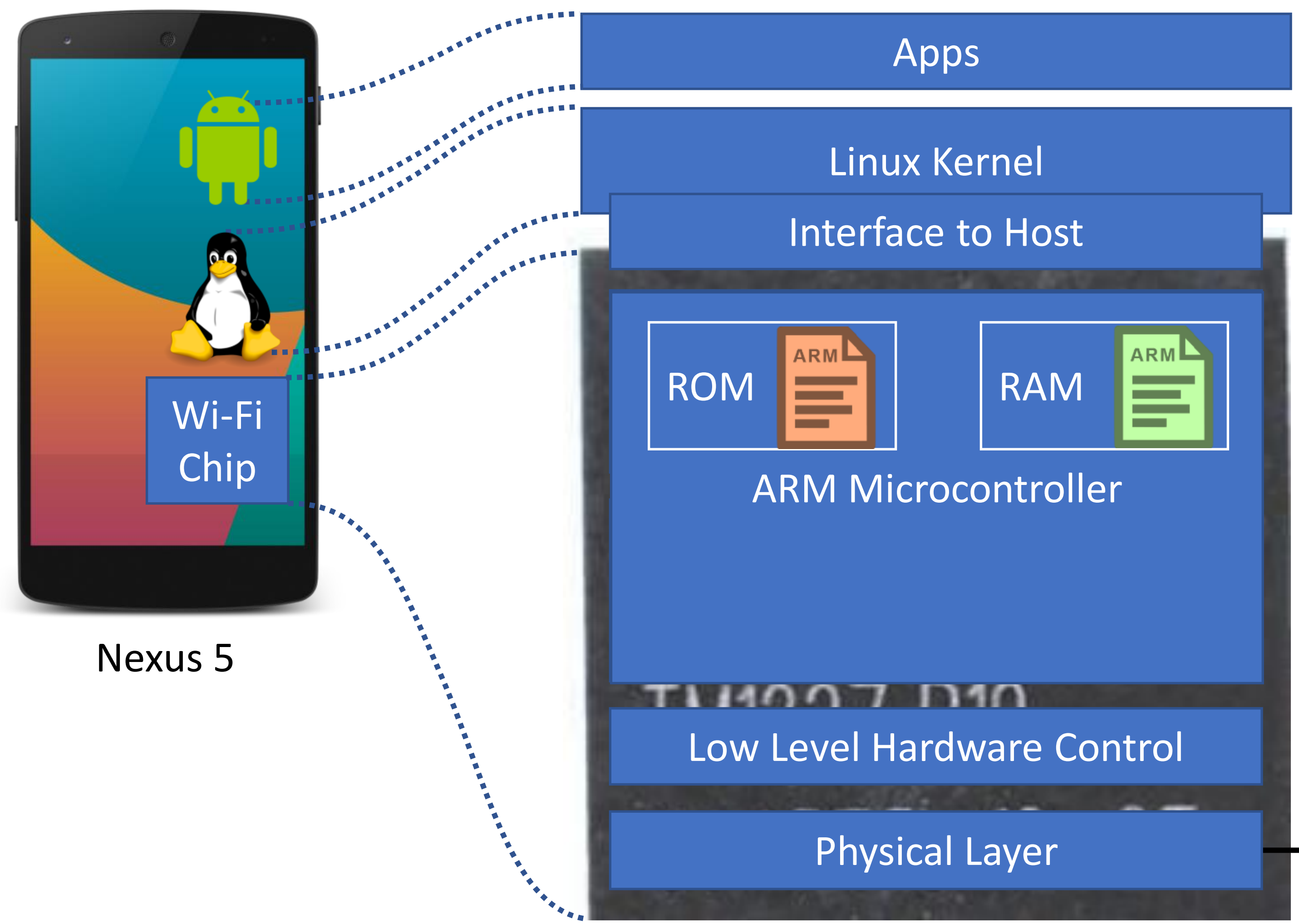


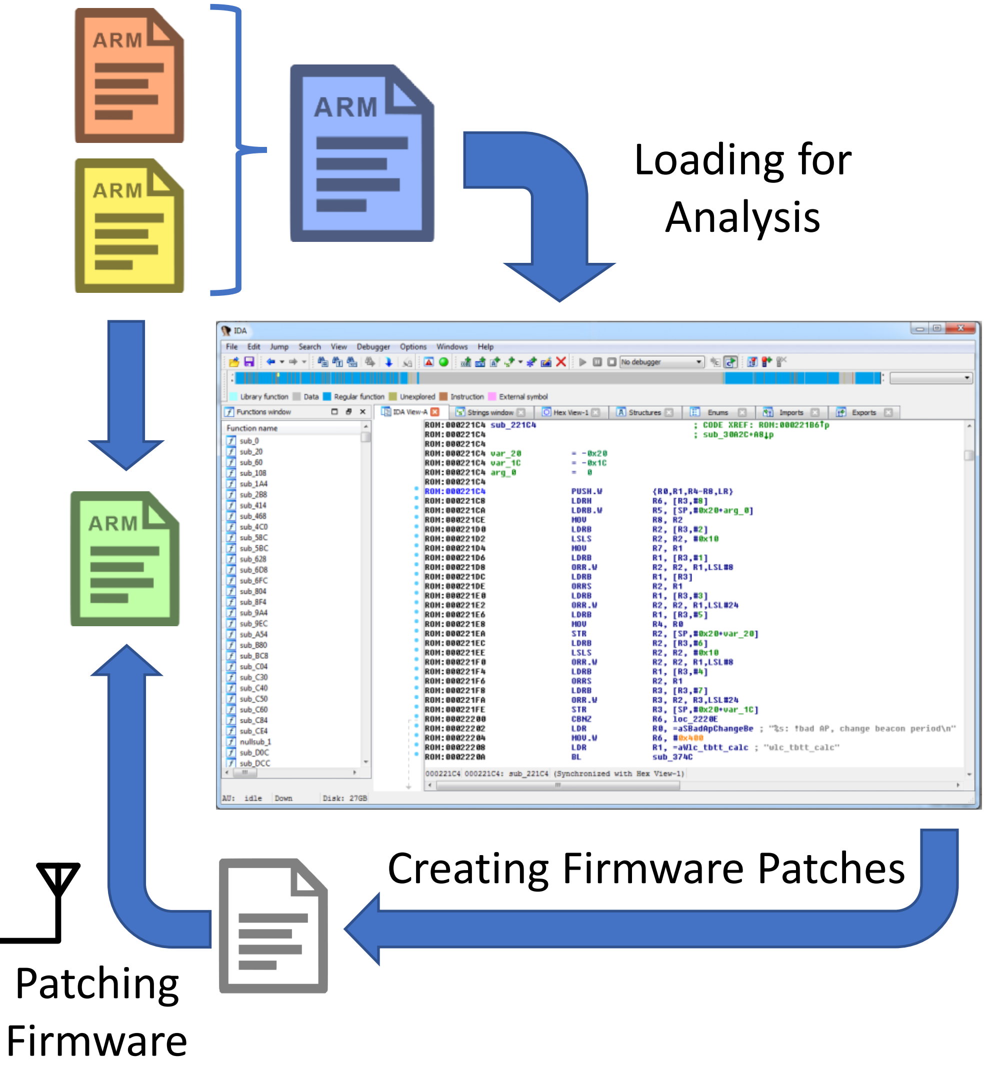
Loading for Analysis

Creating Firmware Patches

Patching Firmware

Firmware Analysis and Patching



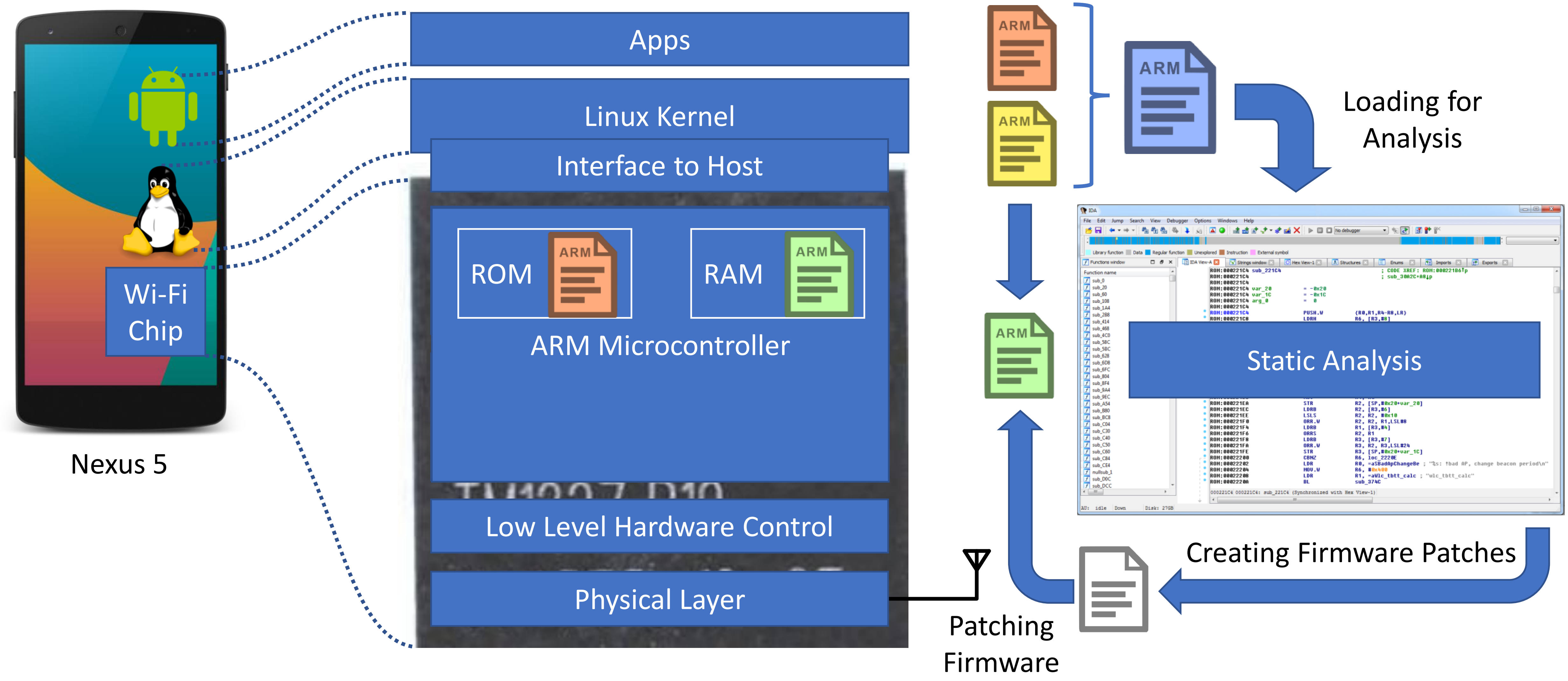


```
sub_0
sub_20
sub_60
sub_108
sub_1A4
sub_2B8
sub_414
sub_468
sub_4C0
sub_58C
sub_5BC
sub_628
sub_6D8
sub_6FC
sub_804
sub_8F4
sub_9A4
sub_9EC
sub_A54
sub_B80
sub_BC8
sub_C04
sub_C30
sub_C40
sub_C50
sub_C60
sub_C84
sub_CE4
nullsub_1
sub_D0C
sub_DCC
```

```
ROM:000221C4 sub_221C4                             ; CODE XREF: ROM:000221B6↑p
ROM:000221C4                                       ; sub_30A2C+A8↓p
ROM:000221C4
ROM:000221C4 var_20          = -0x20
ROM:000221C4 var_1C          = -0x1C
ROM:000221C4 arg_0           =  0
ROM:000221C4
ROM:000221C4                 PUSH.W          {R0,R1,R4-R8,LR}
ROM:000221C8                 LDRH            R6, [R3,#8]
ROM:000221CA                 LDRB.W          R5, [SP,#0x20+arg_0]
ROM:000221CE                 MOV             R8, R2
ROM:000221D0                 LDRB            R2, [R3,#2]
ROM:000221D2                 LSLS            R2, R2, #0x10
ROM:000221D4                 MOV             R7, R1
ROM:000221D6                 LDRB            R1, [R3,#1]
ROM:000221D8                 ORR.W           R2, R2, R1,LSL#8
ROM:000221DC                 LDRB            R1, [R3]
ROM:000221DE                 ORRS            R2, R1
ROM:000221E0                 LDRB            R1, [R3,#3]
ROM:000221E2                 ORR.W           R2, R2, R1,LSL#24
ROM:000221E6                 LDRB            R1, [R3,#5]
ROM:000221E8                 MOV             R4, R0
ROM:000221EA                 STR             R2, [SP,#0x20+var_20]
ROM:000221EC                 LDRB            R2, [R3,#6]
ROM:000221EE                 LSLS            R2, R2, #0x10
ROM:000221F0                 ORR.W           R2, R2, R1,LSL#8
ROM:000221F4                 LDRB            R1, [R3,#4]
ROM:000221F6                 ORRS            R2, R1
ROM:000221F8                 LDRB            R3, [R3,#7]
ROM:000221FA                 ORR.W           R3, R2, R3,LSL#24
ROM:000221FE                 STR             R3, [SP,#0x20+var_1C]
ROM:00022200                 CBNZ            R6, loc_2220E
ROM:00022202                 LDR             R0, =aSBadApChangeBe ; "%s: !bad AP, change beacon period\n"
ROM:00022204                 MOV.W           R6, #0x400
ROM:00022208                 LDR             R1, =aWlc_tbtt_calc ; "wlc_tbtt_calc"
ROM:0002220A                 BL              sub_374C
000221C4 000221C4: sub_221C4 (Synchronized with Hex View-1)
```

```
AU:  idle   Down      Disk: 27GB
```

Static vs. Dynamic Code Analysis



Nexus 5

Wi-Fi Chip

Apps

Linux Kernel

Interface to Host

ROM

RAM

ARM Microcontroller

Low Level Hardware Control

Physical Layer

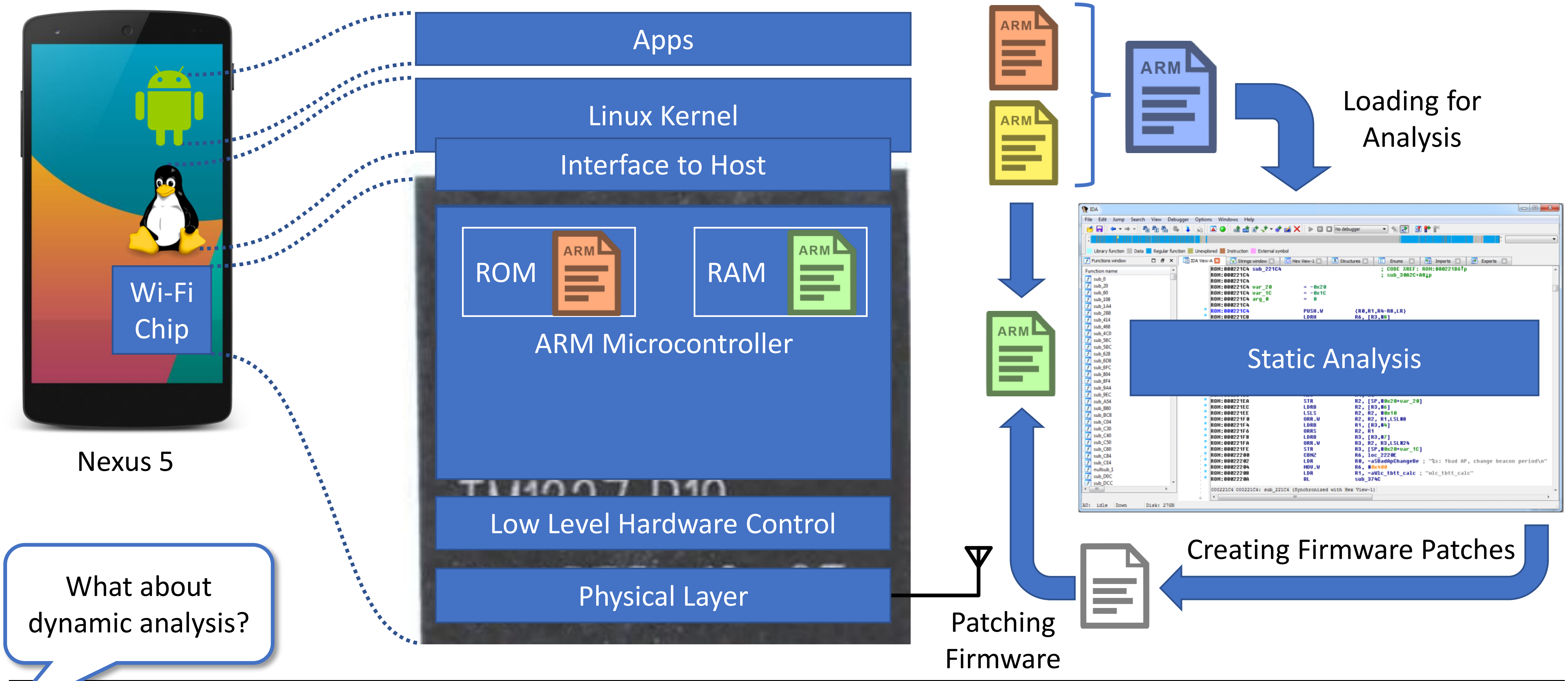
Loading for Analysis

Static Analysis

Creating Firmware Patches

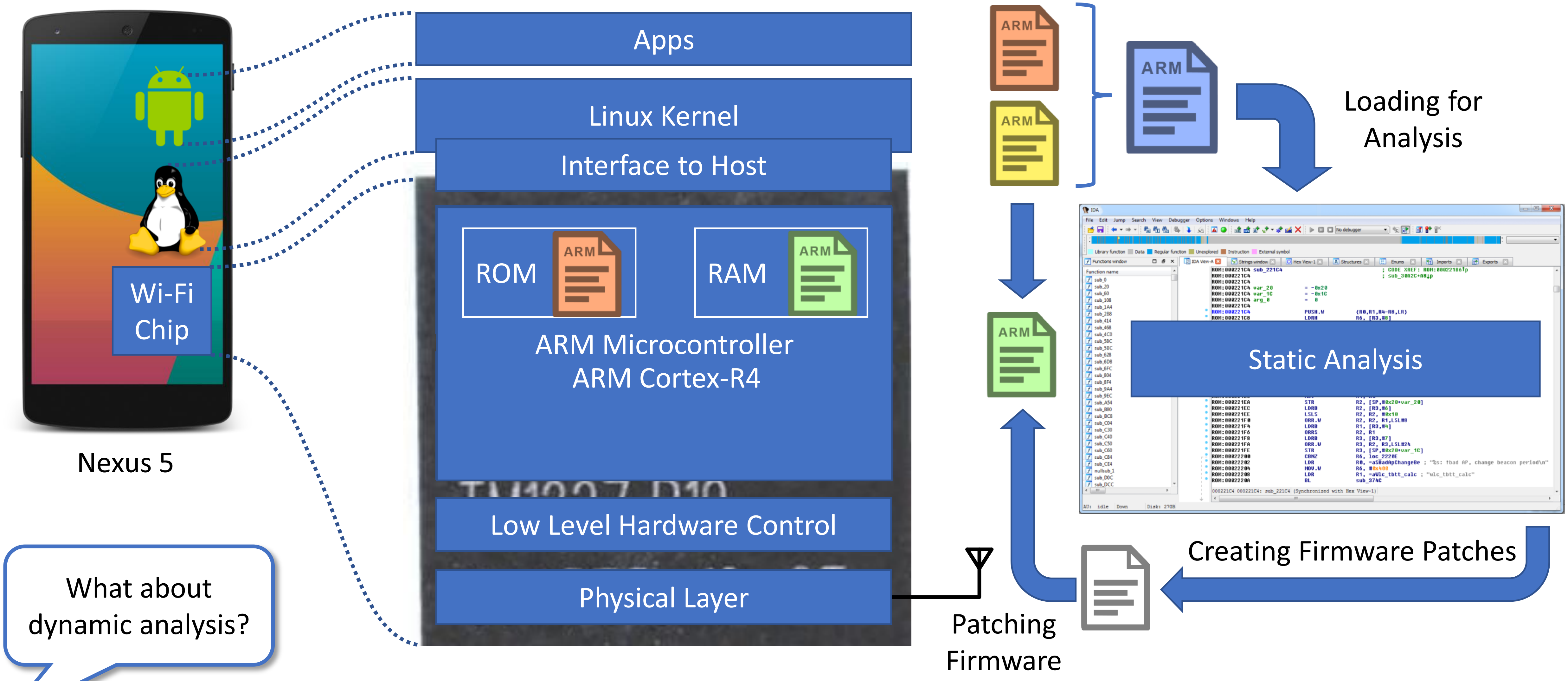
Patching Firmware

Static vs. Dynamic Code Analysis



What about dynamic analysis?

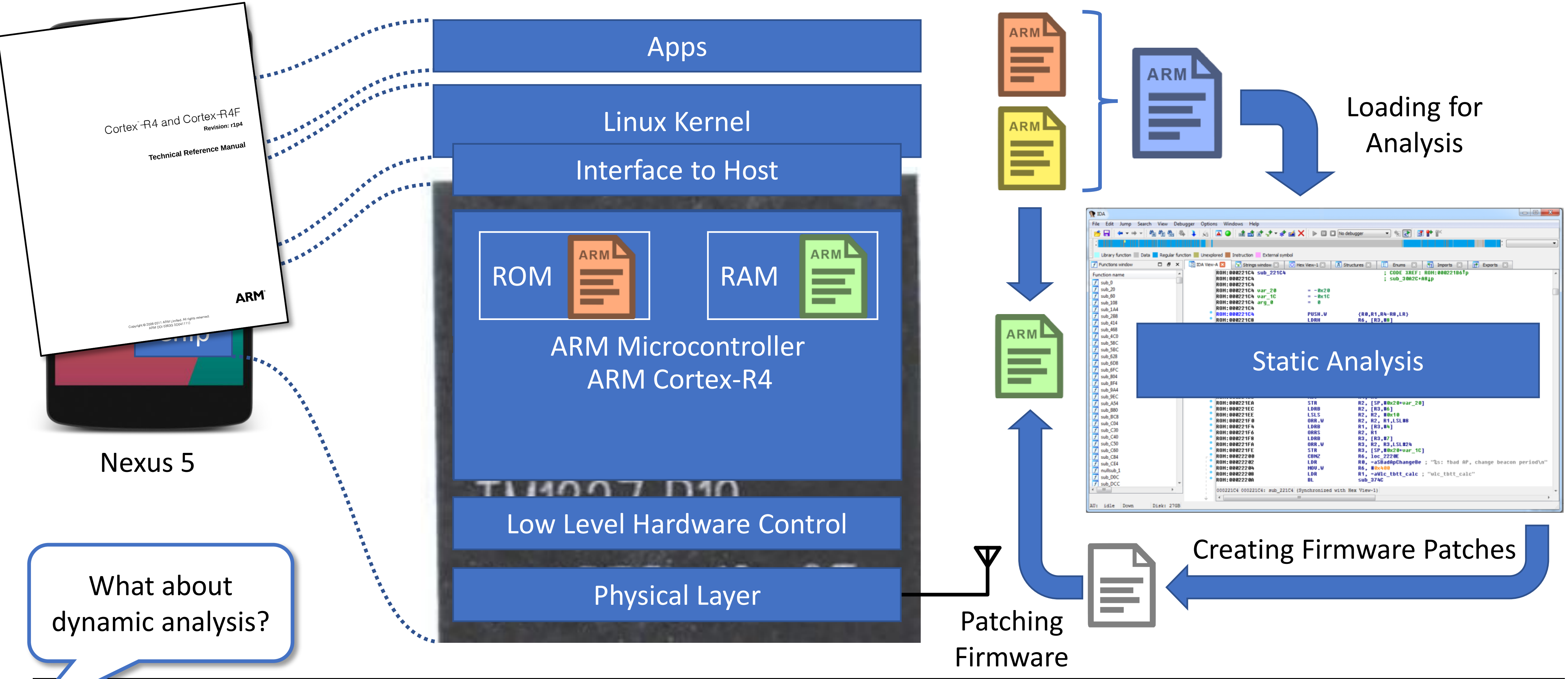
Static vs. Dynamic Code Analysis



```
ROM:000221C4 sub_221C4                              ; CODE XREF: ROM:000221B6↑p
ROM:000221C4                                        ; sub_30A2C+A8↓p
ROM:000221C4
ROM:000221C4 var_20          = -0x20
ROM:000221C4 var_1C          = -0x1C
ROM:000221C4 arg_0           =  0
ROM:000221C4
ROM:000221C4                 PUSH.W          {R0,R1,R4-R8,LR}
ROM:000221C8                 LDRH            R6, [R3,#8]
ROM:000221EA                 STR             R2, [SP,#0x20+var_20]
ROM:000221EC                 LDRB            R2, [R3,#6]
ROM:000221EE                 LSLS            R2, R2, #0x10
ROM:000221F0                 ORR.W           R2, R2, R1,LSL#8
ROM:000221F4                 LDRB            R1, [R3,#4]
ROM:000221F6                 ORRS            R2, R1
ROM:000221F8                 LDRB            R3, [R3,#7]
ROM:000221FA                 ORR.W           R3, R2, R3,LSL#24
ROM:000221FE                 STR             R3, [SP,#0x20+var_1C]
ROM:00022200                 CBNZ            R6, loc_2220E
ROM:00022202                 LDR             R0, =aSBadApChangeBe ; "%s: !bad AP, change beacon period\n"
ROM:00022204                 MOV.W           R6, #0x400
ROM:00022208                 LDR             R1, =aWlc_tbtt_calc ; "wlc_tbtt_calc"
ROM:0002220A                 BL              sub_374C
000221C4 000221C4: sub_221C4 (Synchronized with Hex View-1)
```

What about dynamic analysis?

Static vs. Dynamic Code Analysis



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

Nexus 5

Apps

Linux Kernel

Interface to Host

ROM ARM

RAM ARM

ARM Microcontroller
ARM Cortex-R4

Low Level Hardware Control

Physical Layer

ARM

ARM

ARM

Loading for Analysis

ARM

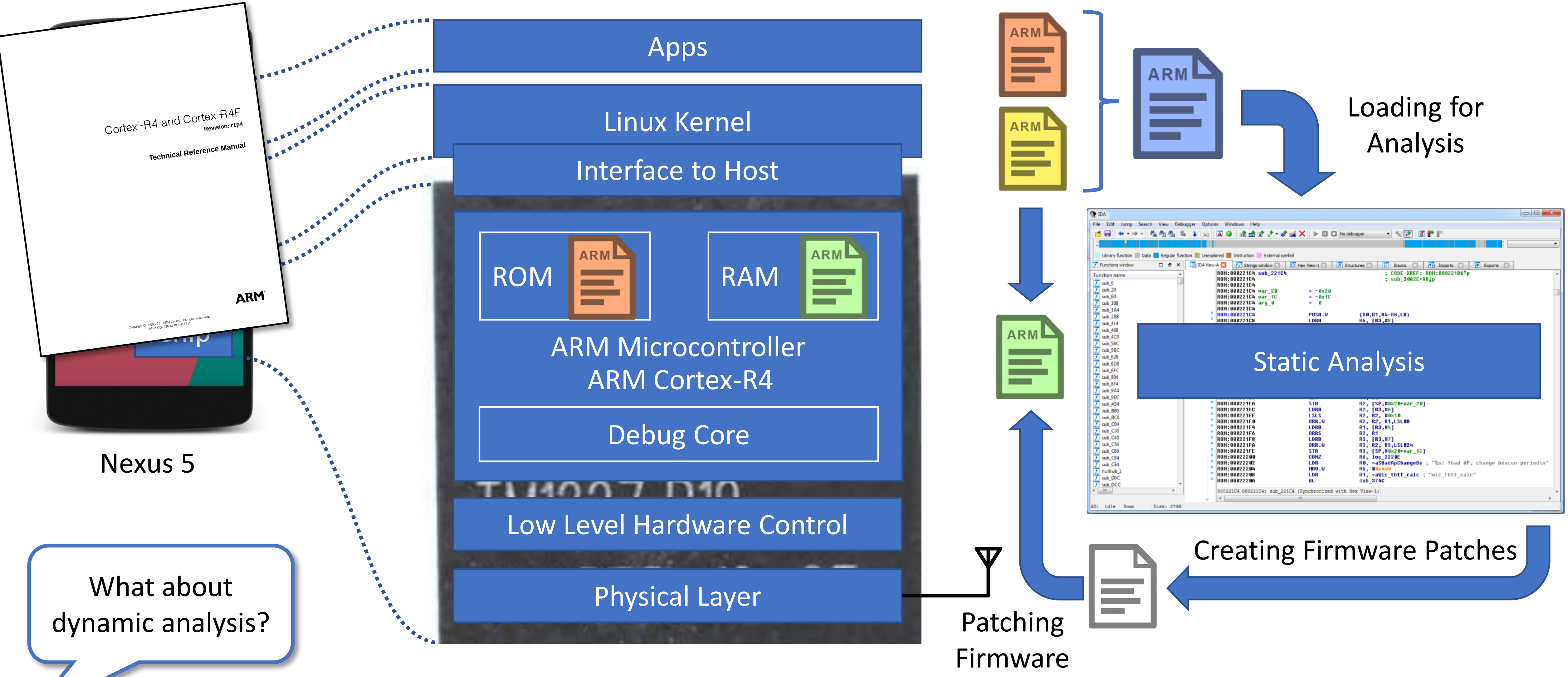
Static Analysis

Creating Firmware Patches

Patching Firmware

What about dynamic analysis?

Static vs. Dynamic Code Analysis

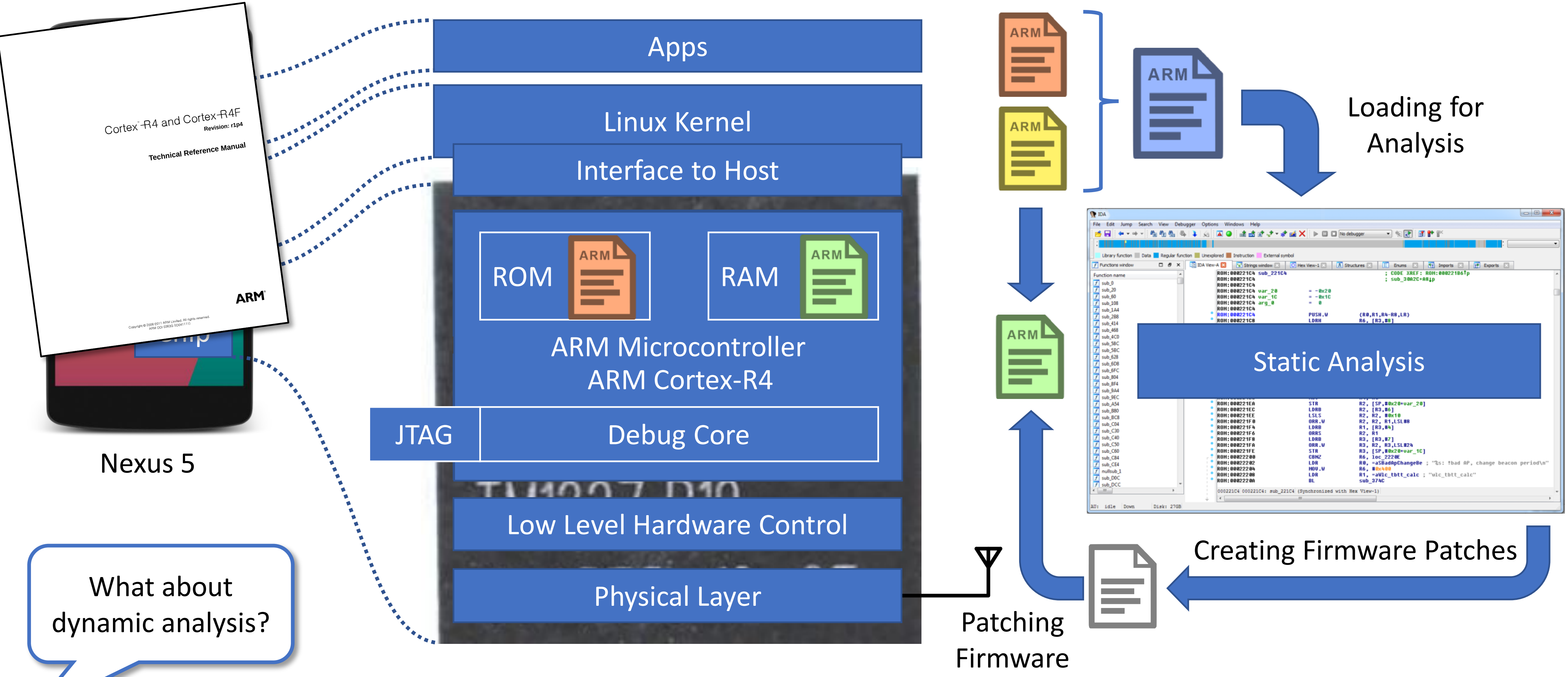


```
ROM:000221C4 sub_221C4                          ; CODE XREF: ROM:000221B6↑p
ROM:000221C4                                    ; sub_30A2C+A8↓p
ROM:000221C4
ROM:000221C4 var_20          = -0x20
ROM:000221C4 var_1C          = -0x1C
ROM:000221C4 arg_0           =  0
ROM:000221C4
ROM:000221C4                 PUSH.W          {R0,R1,R4-R8,LR}
ROM:000221C8                 LDRH            R6, [R3,#8]
```

```
ROM:000221EA                 STR             R2, [SP,#0x20+var_20]
ROM:000221EC                 LDRB            R2, [R3,#6]
ROM:000221EE                 LSLS            R2, R2, #0x10
ROM:000221F0                 ORR.W           R2, R2, R1,LSL#8
ROM:000221F4                 LDRB            R1, [R3,#4]
ROM:000221F6                 ORRS            R2, R1
ROM:000221F8                 LDRB            R3, [R3,#7]
ROM:000221FA                 ORR.W           R3, R2, R3,LSL#24
ROM:000221FE                 STR             R3, [SP,#0x20+var_1C]
ROM:00022200                 CBNZ            R6, loc_2220E
ROM:00022202                 LDR             R0, =aSBadApChangeBe ; "%s: !bad AP, change beacon period\n"
ROM:00022204                 MOV.W           R6, #0x400
ROM:00022208                 LDR             R1, =aWlc_tbtt_calc ; "wlc_tbtt_calc"
ROM:0002220A                 BL              sub_374C
000221C4 000221C4: sub_221C4 (Synchronized with Hex View-1)
AU:  idle   Down      Disk: 27GB
```

What about dynamic analysis?

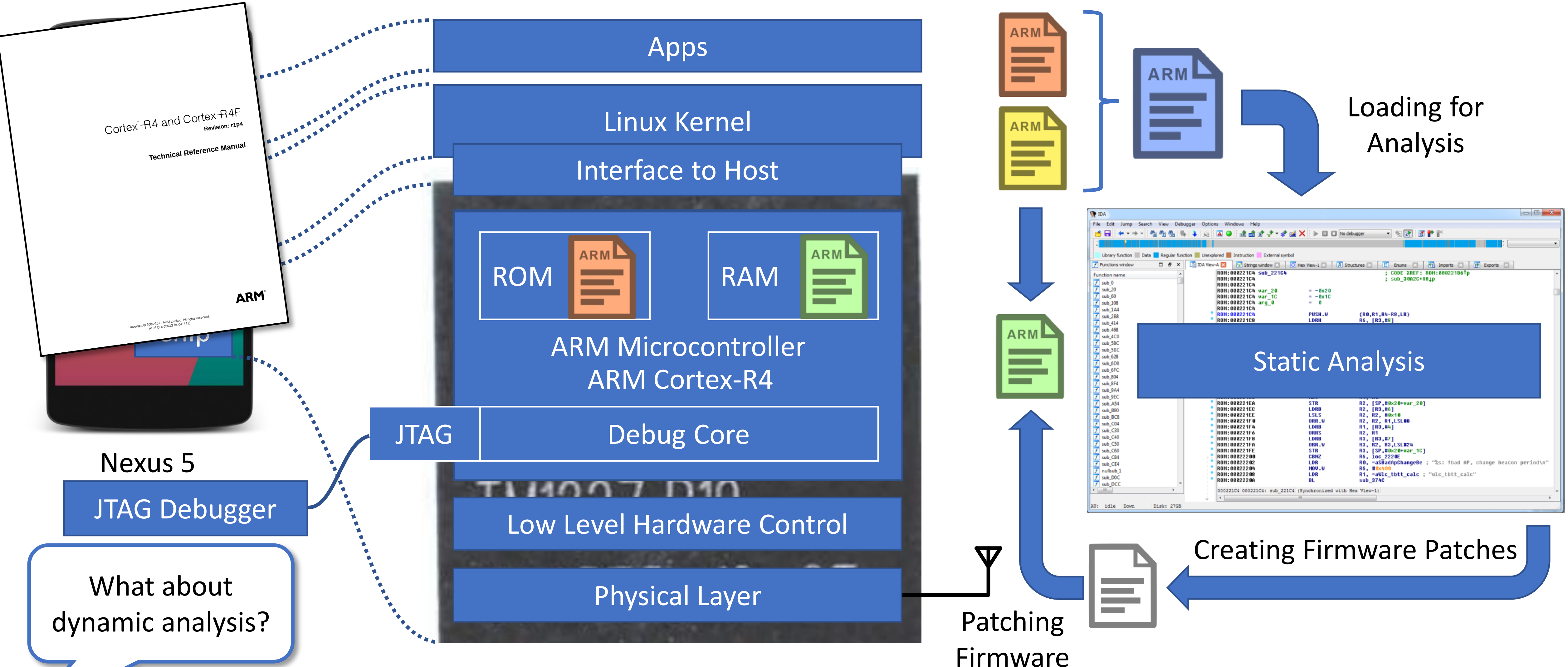
Static vs. Dynamic Code Analysis



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual

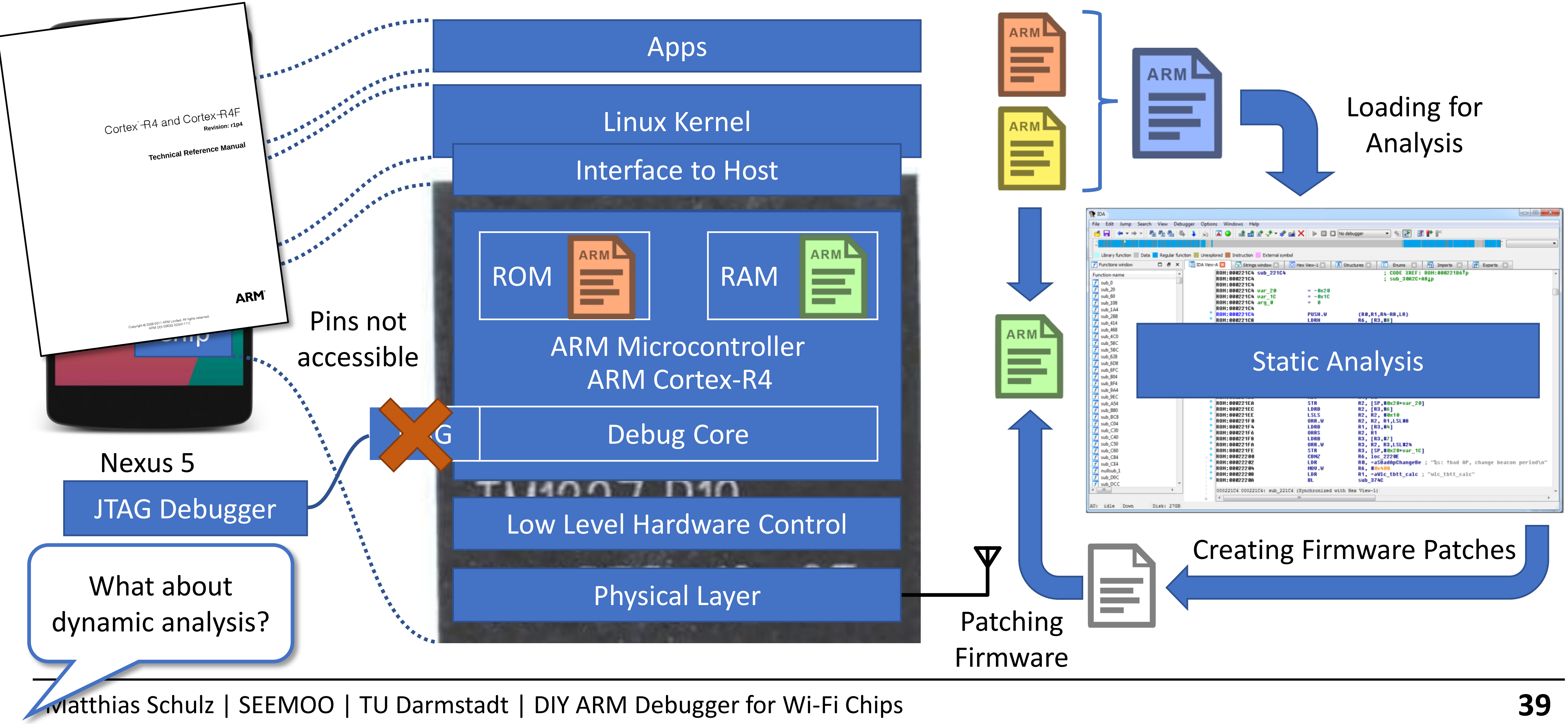
What about dynamic analysis?

Static vs. Dynamic Code Analysis



What about dynamic analysis?

Static vs. Dynamic Code Analysis



Apps

Linux Kernel

Interface to Host

ROM

RAM

ARM Microcontroller
ARM Cortex-R4

G

Debug Core

Low Level Hardware Control

Physical Layer

Pins not accessible

Nexus 5

JTAG Debugger

What about dynamic analysis?

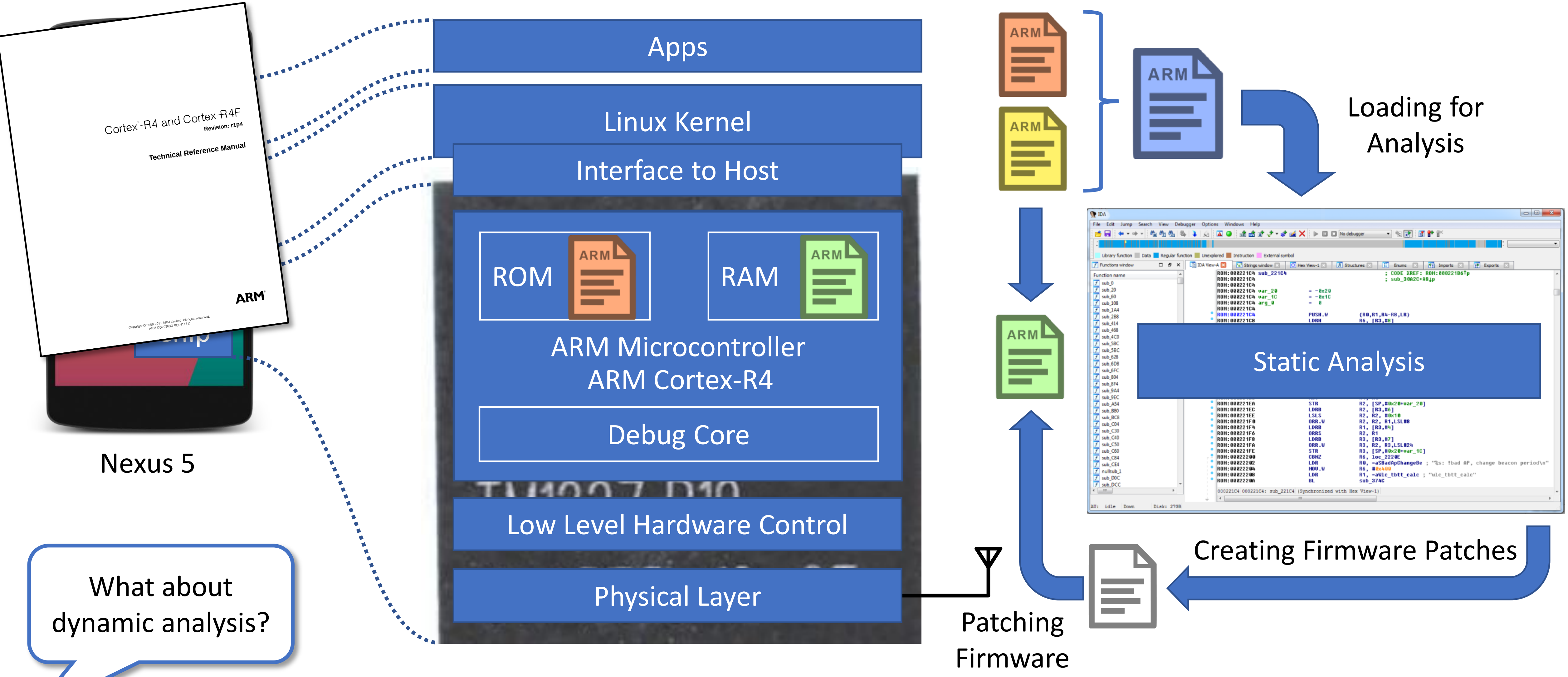
Loading for Analysis

Static Analysis

Creating Firmware Patches

Patching Firmware

Static vs. Dynamic Code Analysis



Cortex™-R4 and Cortex-R4F
Revision: r1p4

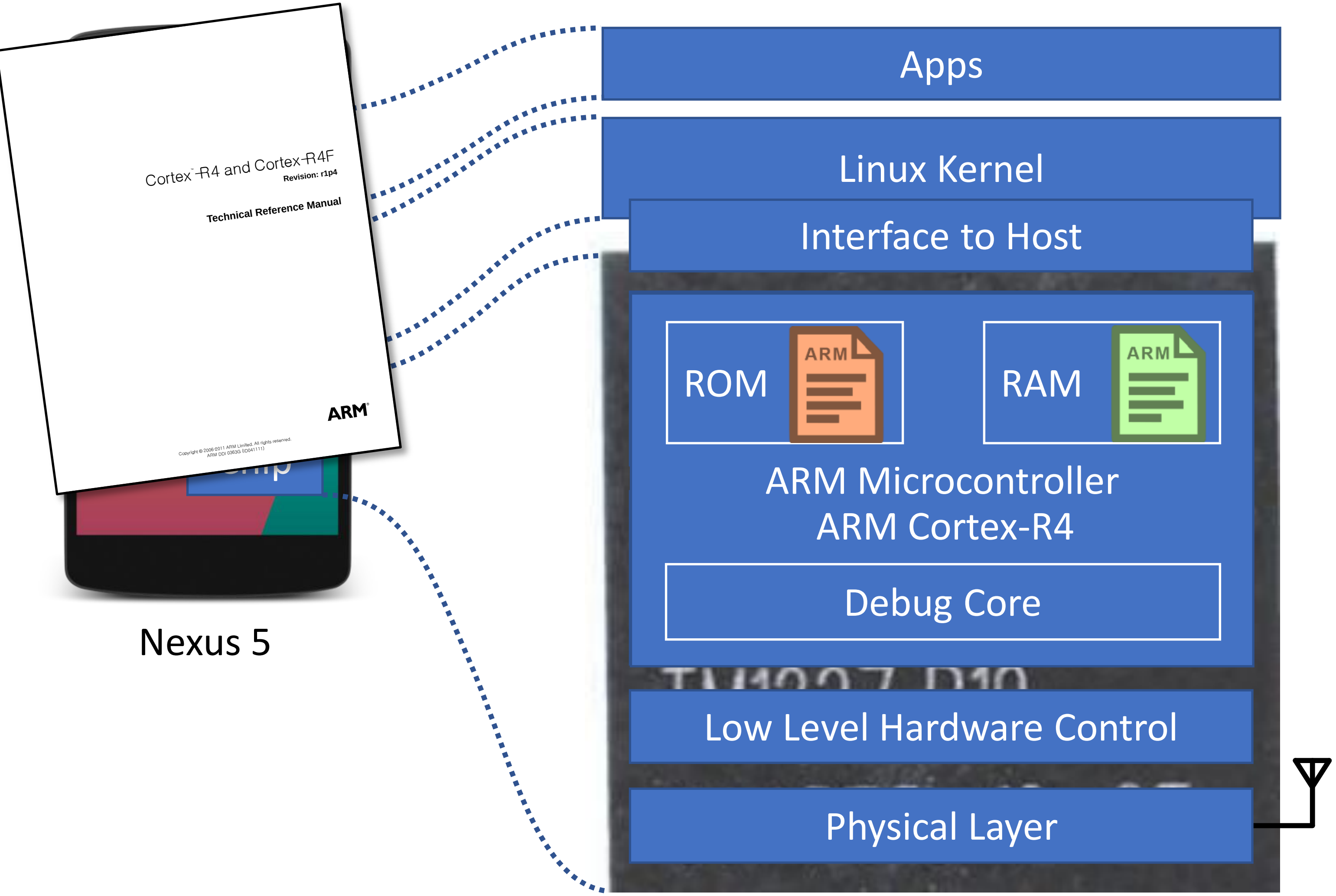
Technical Reference Manual

ARM®

Copyright © 2006-2011 ARM Limited. All rights reserved.
ARM DDI 0363G (ID041111)

What about dynamic analysis?

Static vs. Dynamic Code Analysis



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

Nexus 5

Apps

Linux Kernel

Interface to Host

ROM ARM

RAM ARM

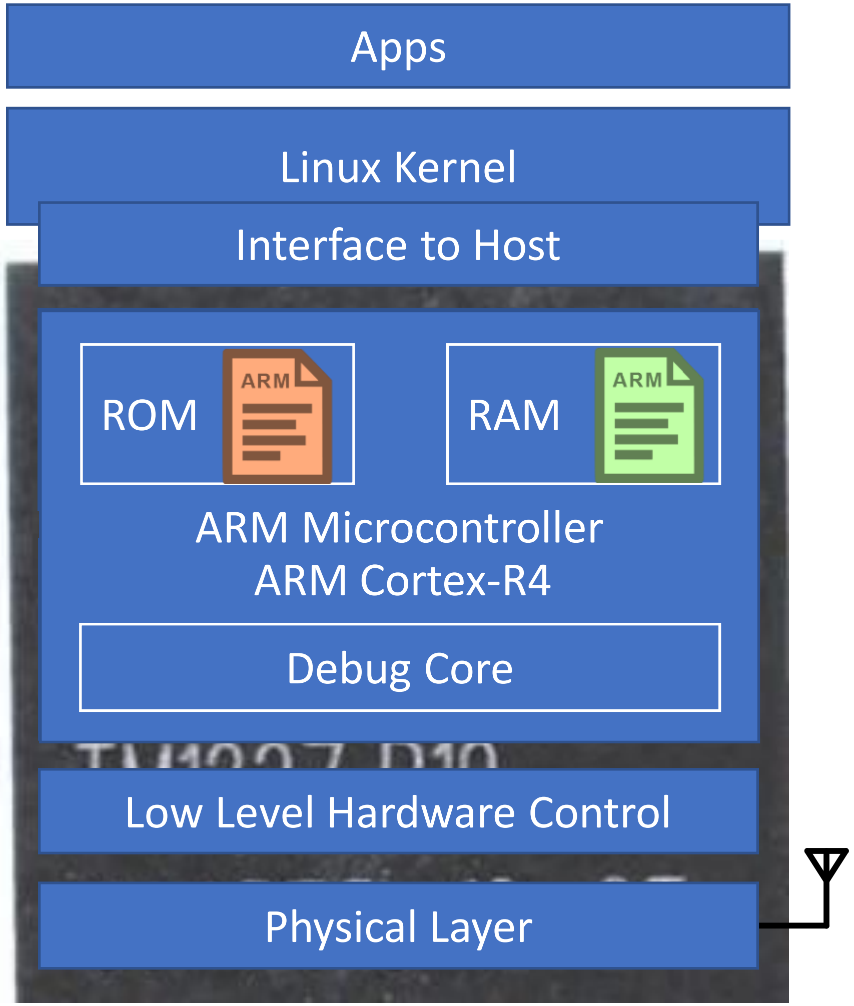
ARM Microcontroller
ARM Cortex-R4

Debug Core

Low Level Hardware Control

Physical Layer

Halting vs. Monitor Debug-Mode



Apps

Linux Kernel

Interface to Host

ROM

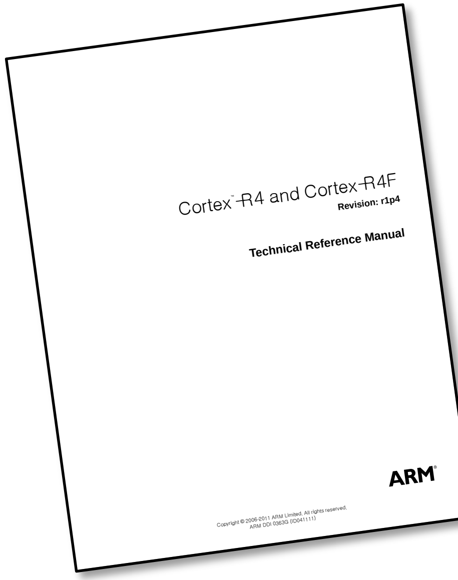
RAM

ARM Microcontroller
ARM Cortex-R4

Debug Core

Low Level Hardware Control

Physical Layer

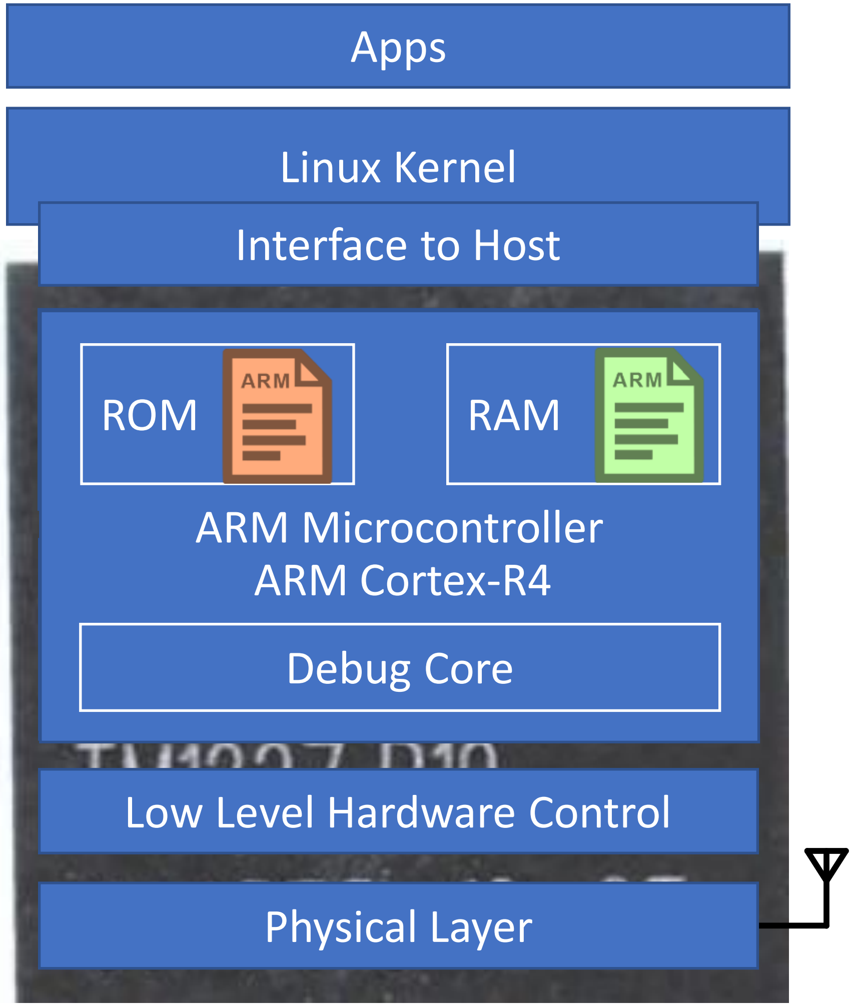


Cortex™-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM®

Halting vs. Monitor Debug-Mode

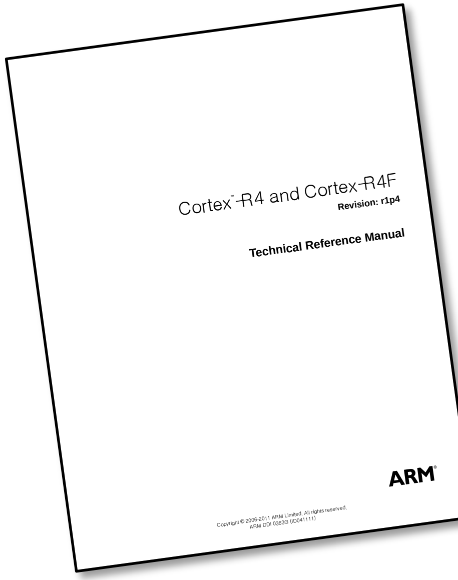


Example Program

```
➡ 0x50: MOV R0, #10
  0x52: MOV R1, #3
  0x54: ADD R0, R0, R1
  0x56: B 0x50
```

Registers

```
R0 = UNDEF
R1 = UNDEF
...
PC = 0x50
```

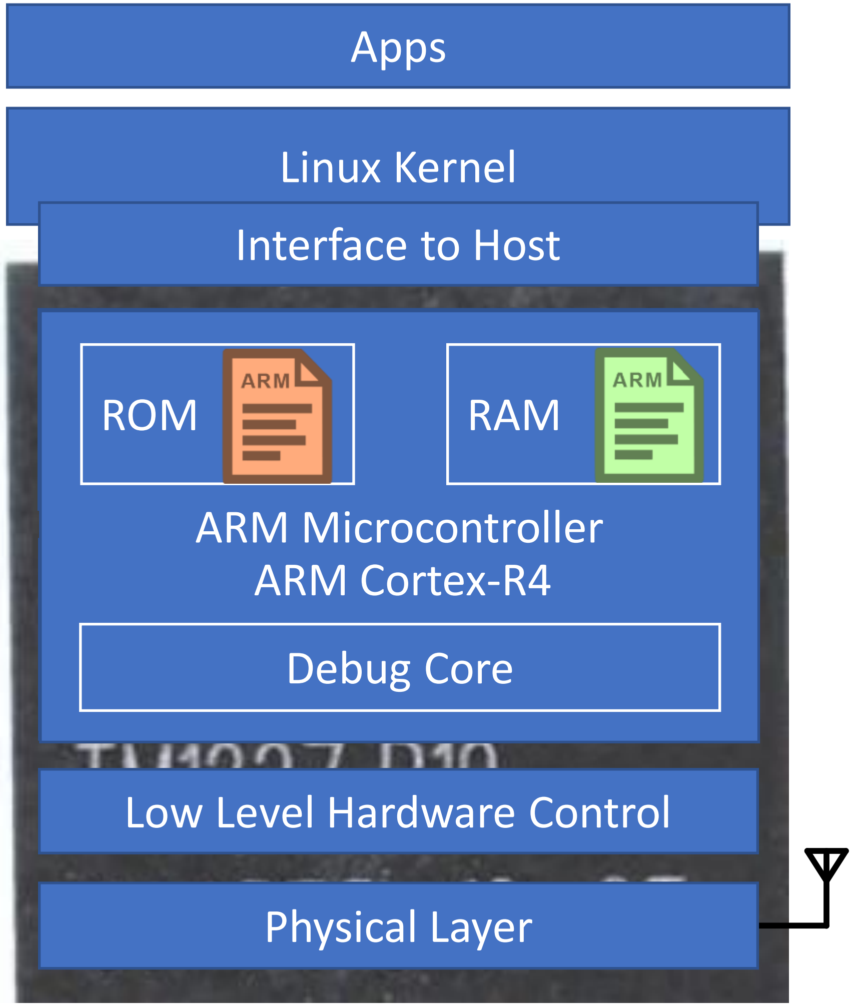


Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

Halting vs. Monitor Debug-Mode



Apps

Linux Kernel

Interface to Host

ROM

RAM

ARM Microcontroller
ARM Cortex-R4

Debug Core

Low Level Hardware Control

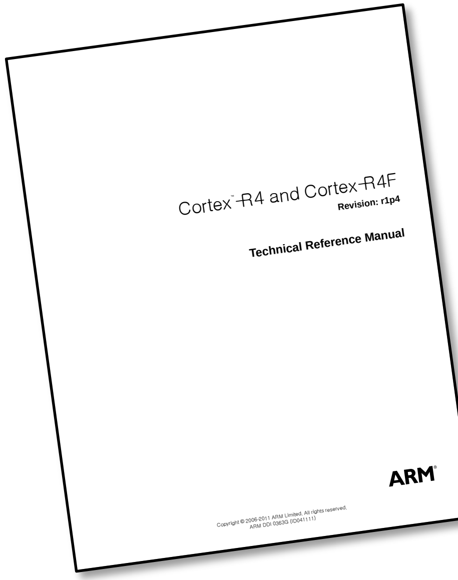
Physical Layer

Example Program

```
   0x50: MOV R0, #10
➡ 0x52: MOV R1, #3
   0x54: ADD R0, R0, R1
   0x56: B 0x50
```

Registers

```
R0 = 10
R1 = UNDEF
...
PC = 0x52
```

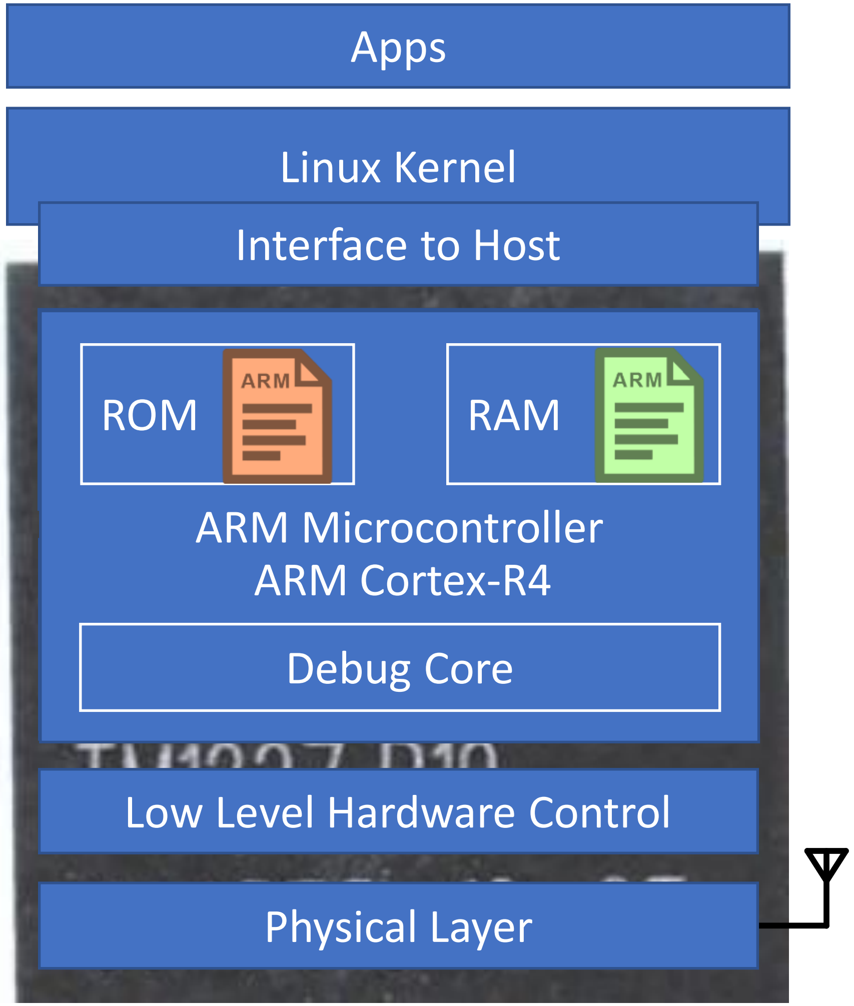


Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

Halting vs. Monitor Debug-Mode

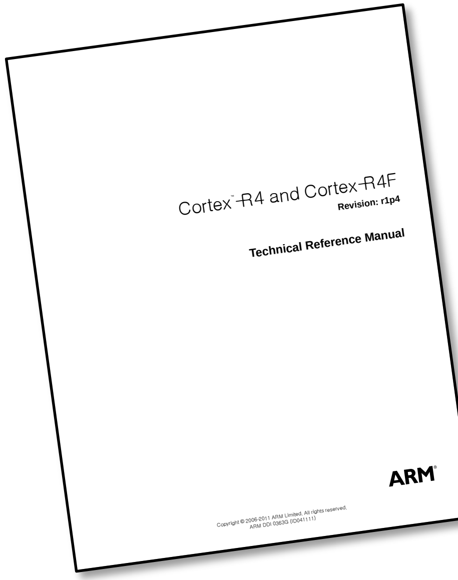


Example Program

```
   0x50: MOV R0, #10
   0x52: MOV R1, #3
➡ 0x54: ADD R0, R0, R1
   0x56: B 0x50
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x54
```

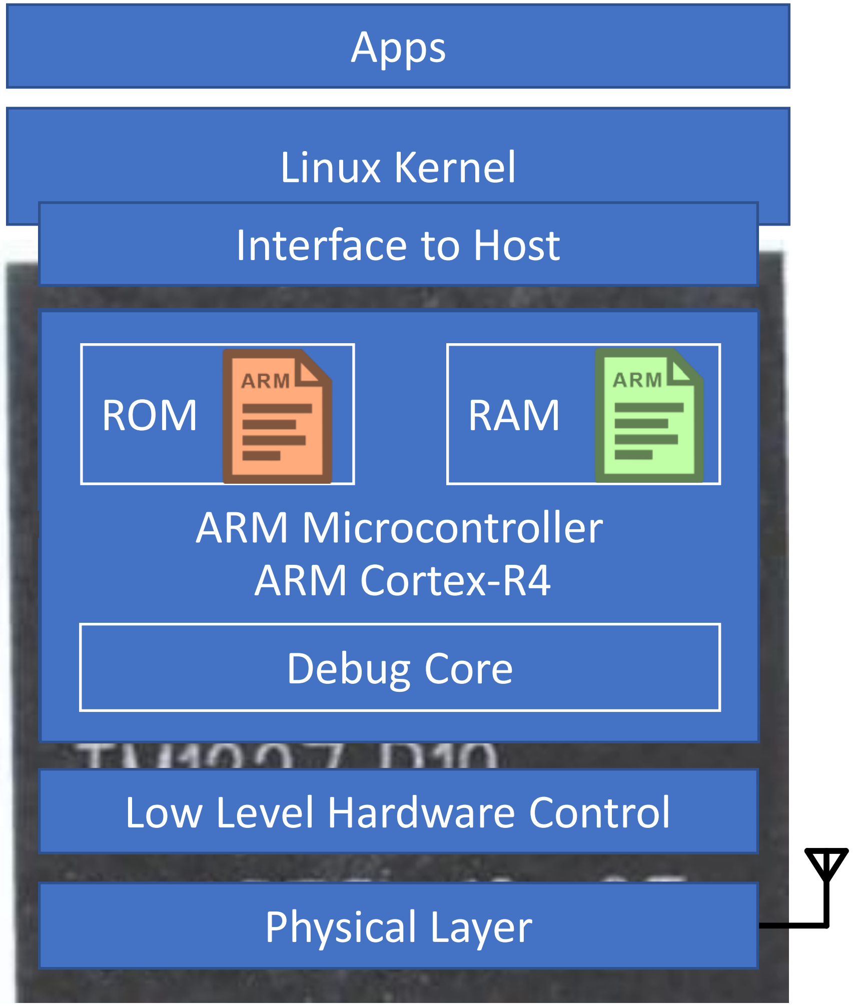


Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

Halting vs. Monitor Debug-Mode

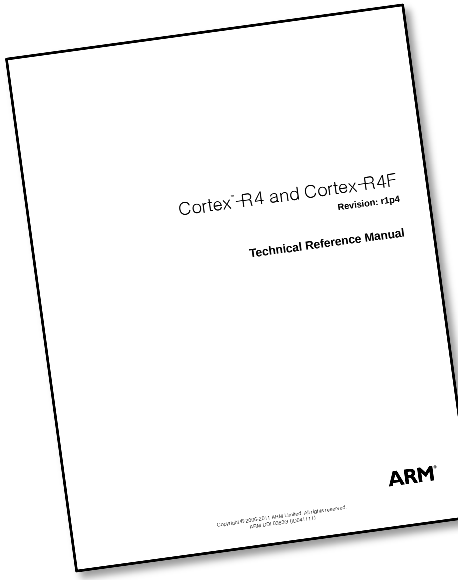


Example Program

```
   0x50: MOV R0, #10
   0x52: MOV R1, #3
   0x54: ADD R0, R0, R1
➡  0x56: B 0x50
```

Registers

```
R0 = 13
R1 = 3
...
PC = 0x56
```

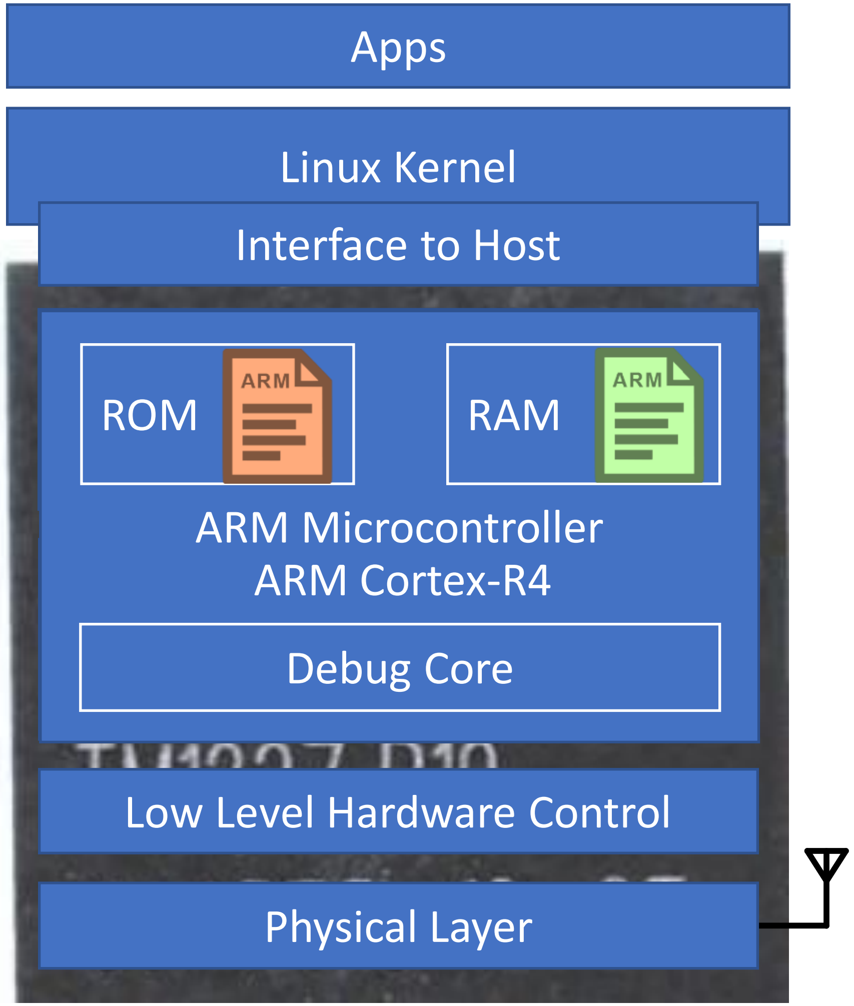


Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

Halting vs. Monitor Debug-Mode


Apps

Linux Kernel

Interface to Host

ROM

RAM

ARM Microcontroller
ARM Cortex-R4

Debug Core

Low Level Hardware Control

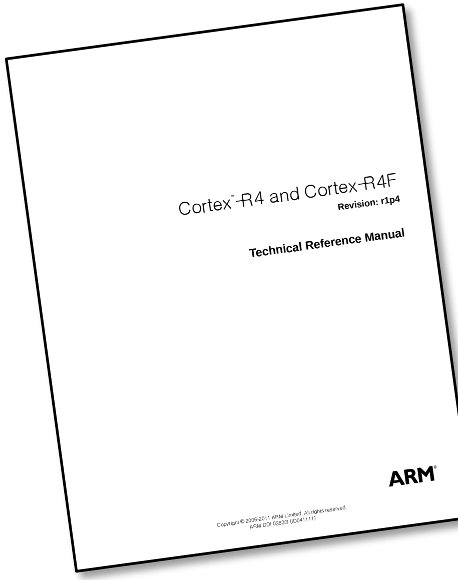
Physical Layer

Example Program

```
➡ 0x50: MOV R0, #10
  0x52: MOV R1, #3
  0x54: ADD R0, R0, R1
  0x56: B 0x50
```

Registers

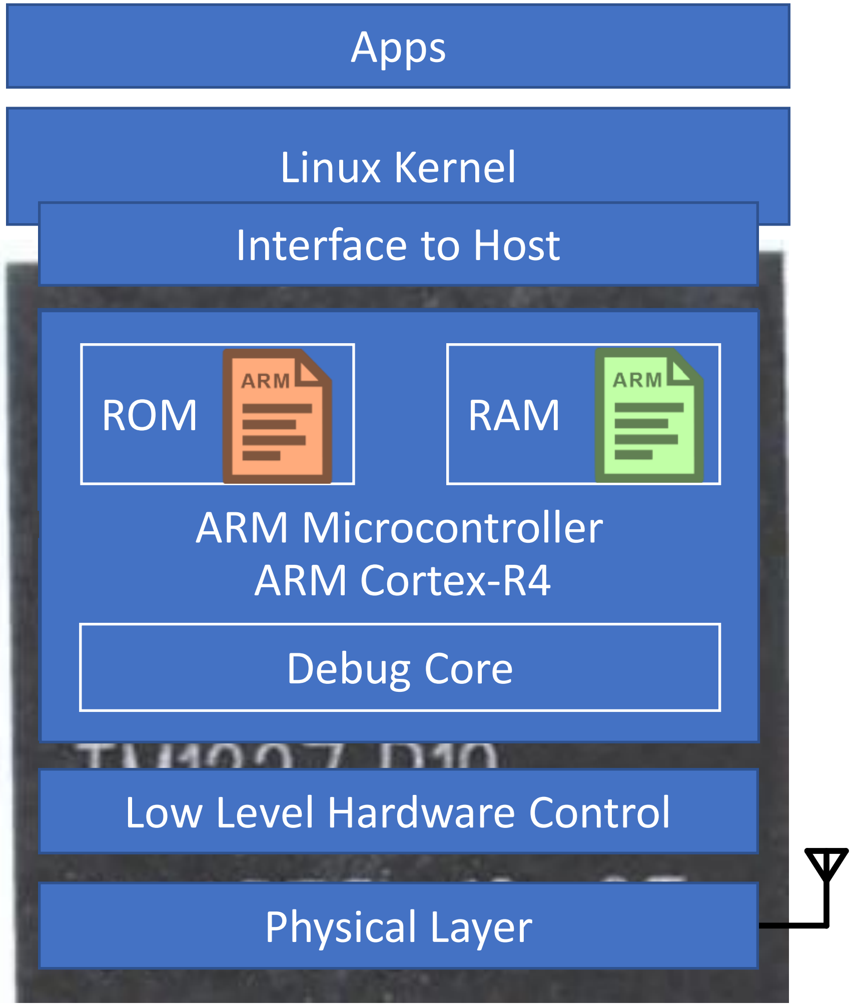
```
R0 = 13
R1 = 3
...
PC = 0x50
```


Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

Halting vs. Monitor Debug-Mode



Apps

Linux Kernel

Interface to Host

ROM

RAM

ARM Microcontroller
ARM Cortex-R4

Debug Core

Low Level Hardware Control

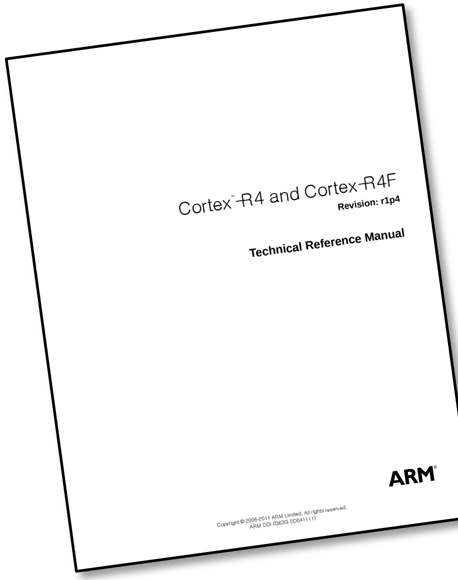
Physical Layer

Example Program

```
➡ 0x50: MOV R0, #10
● 0x52: MOV R1, #3
  0x54: ADD R0, R0, R1
  0x56: B 0x50
```

Registers

```
R0 = 13
R1 = 3
...
PC = 0x50
```

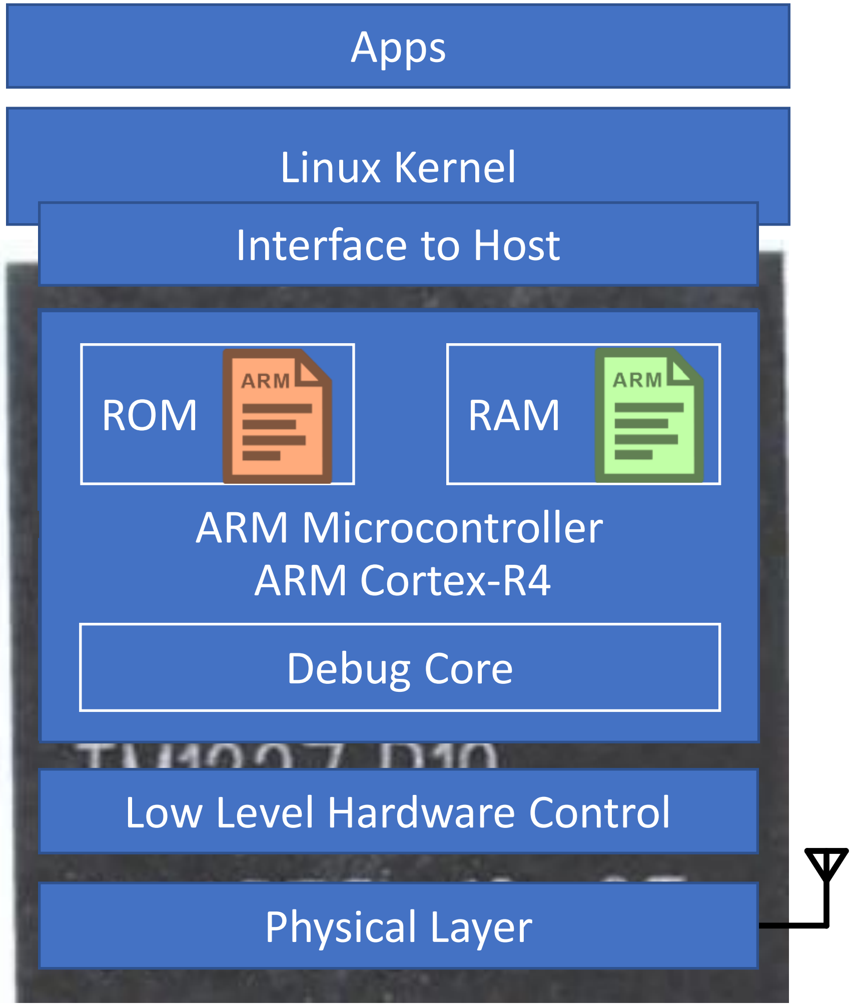


Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

Halting vs. Monitor Debug-Mode

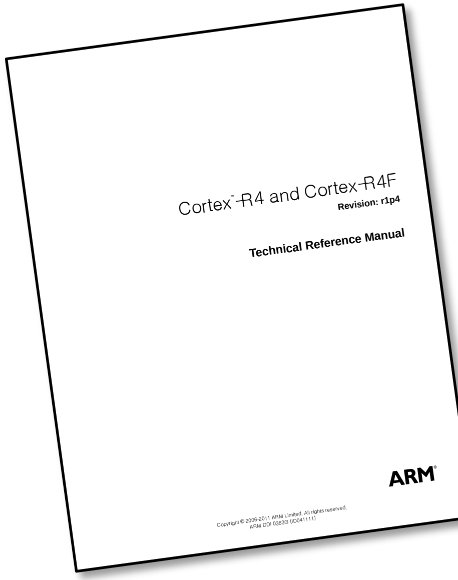


Example Program

```
0x50: MOV R0, #10
0x52: MOV R1, #3
0x54: ADD R0, R0, R1
0x56: B 0x50
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x52
```



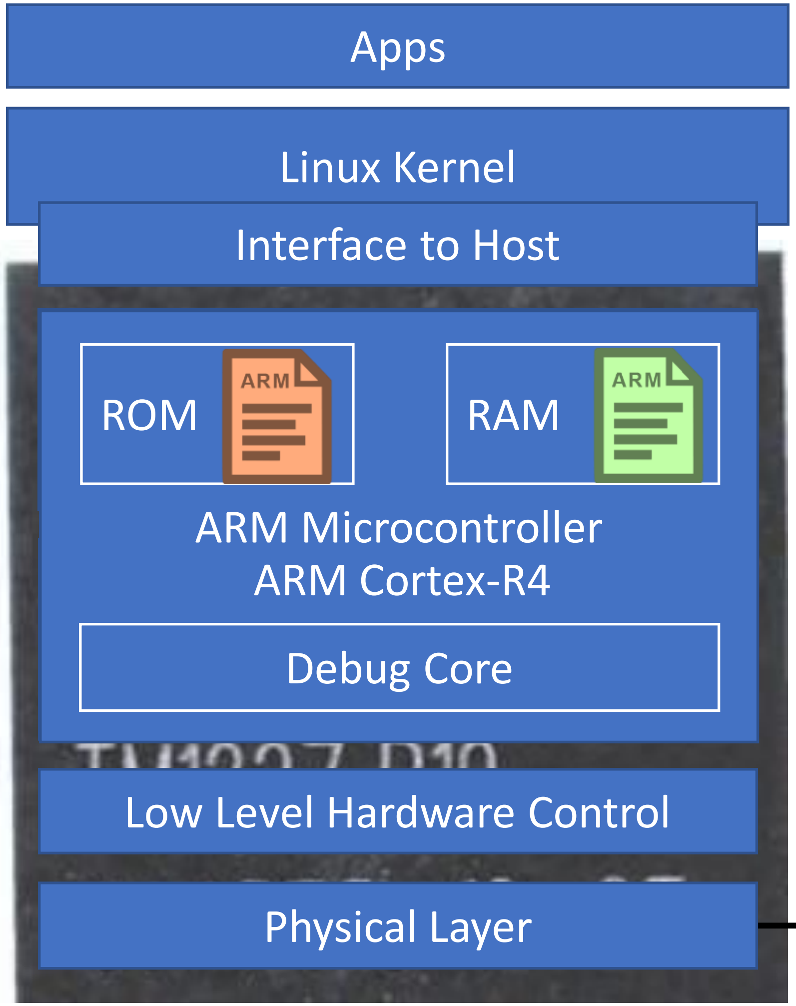
Cortex™-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM®

Copyright © 2006-2011 ARM Limited. All rights reserved.
ARM DDI 0363G (ID041111)

Halting vs. Monitor Debug-Mode



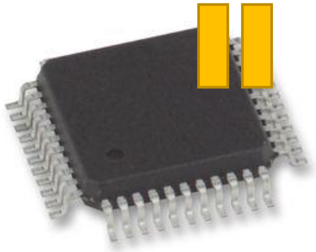
Example Program

```
  0x50: MOV R0, #10
● 0x52: MOV R1, #3
  0x54: ADD R0, R0, R1
  0x56: B 0x50
```

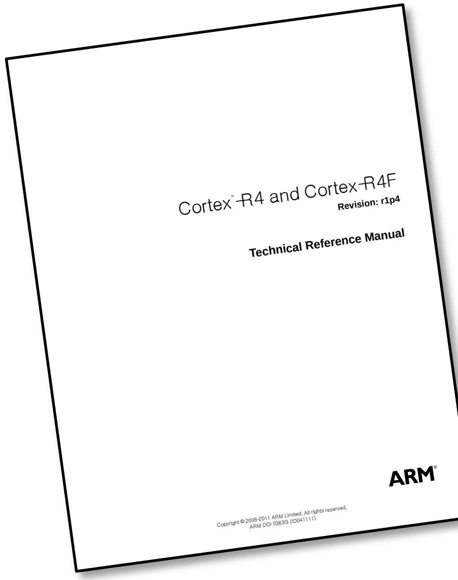
Registers

```
R0 = 10
R1 = 3
...
PC = 0x52
```

Halting Debug-Mode



Control handed
to external debugger

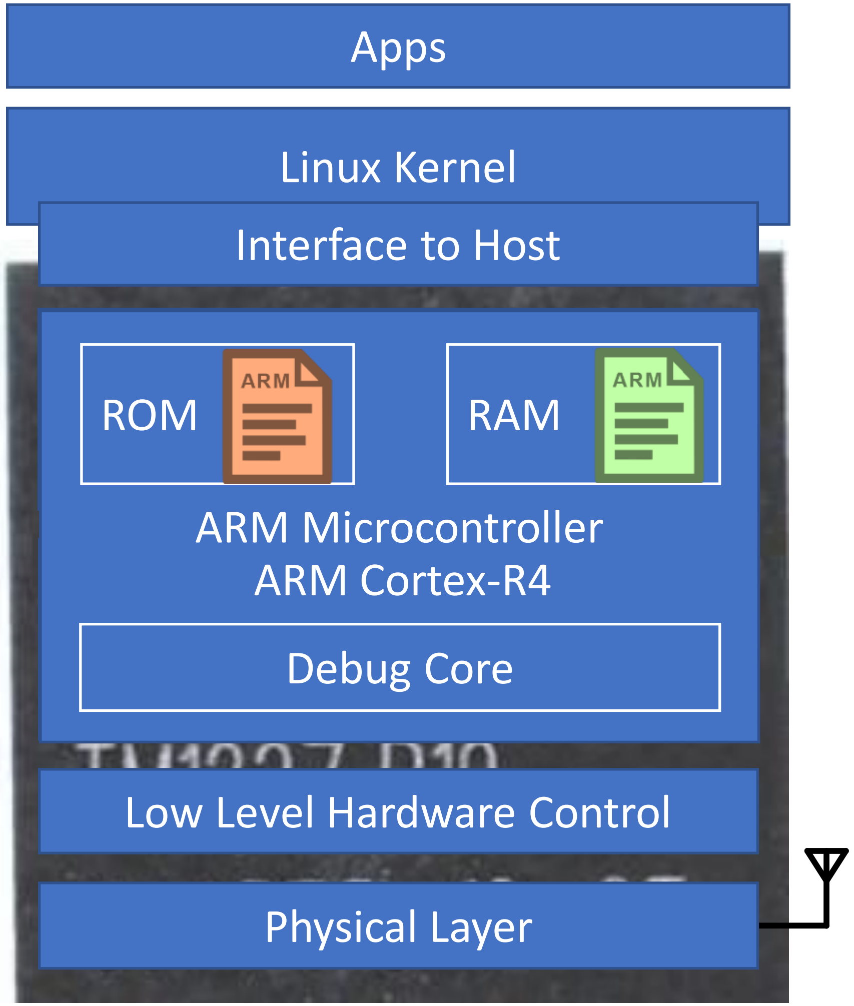


Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

Halting vs. Monitor Debug-Mode

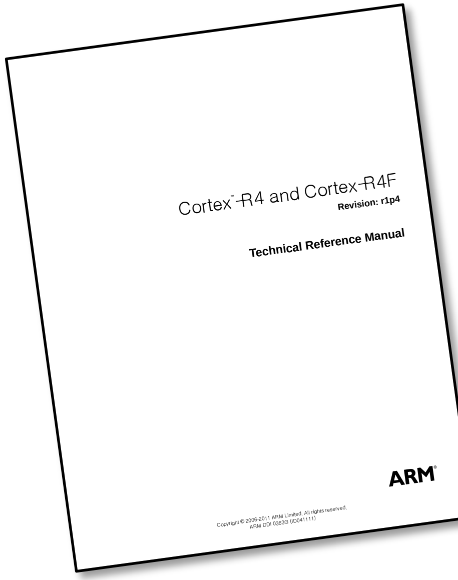


Example Program

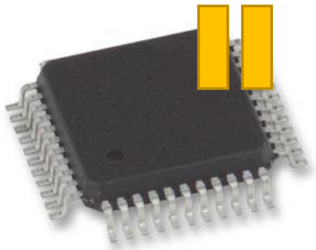
```
0x50: MOV R0, #10
0x52: MOV R1, #3
0x54: ADD R0, R0, R1
0x56: B 0x50
```

Registers

R0 = 10
R1 = 3
...
PC = 0x52

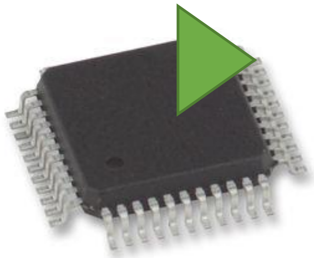

Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

Halting Debug-Mode



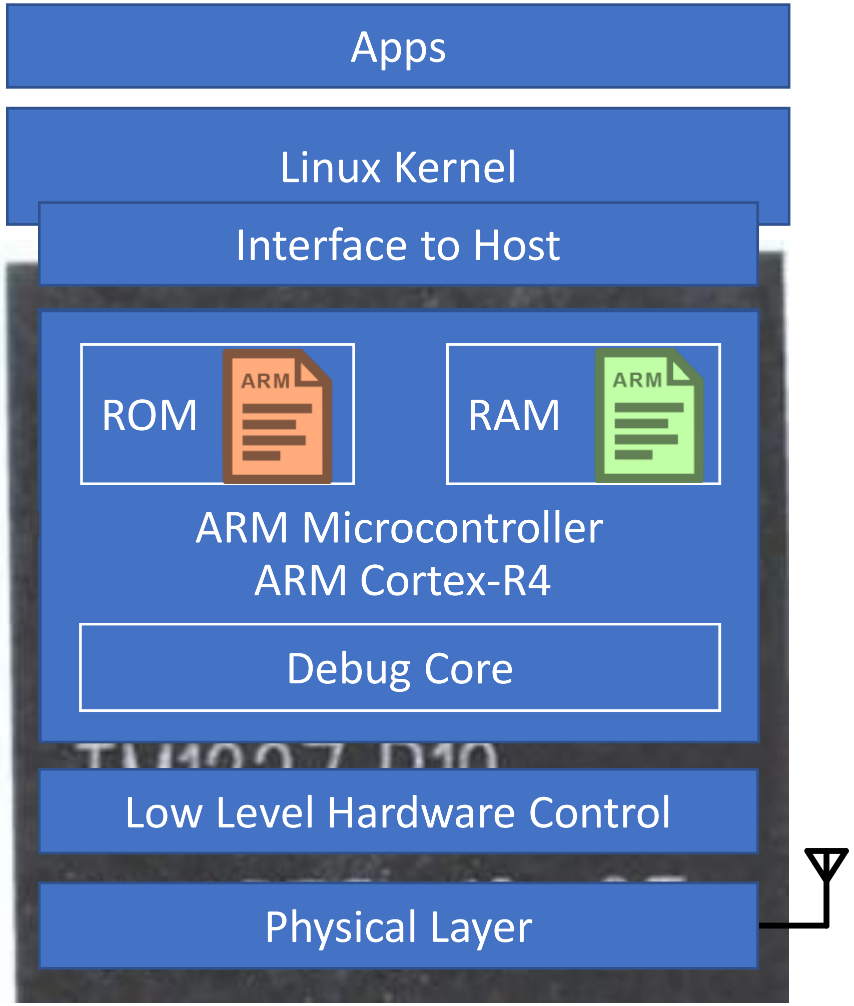
Control handed
to external debugger

Monitor Debug-Mode



Abort Exception
→ Execution continues
in exception handler

Halting vs. Monitor Debug-Mode



Apps

Linux Kernel

Interface to Host

ROM

RAM

ARM Microcontroller
ARM Cortex-R4

Debug Core

Low Level Hardware Control

Physical Layer

Example Program

```
0x50: MOV R0, #10
0x52: MOV R1, #3
0x54: ADD R0, R0, R1
0x56: B 0x50
```

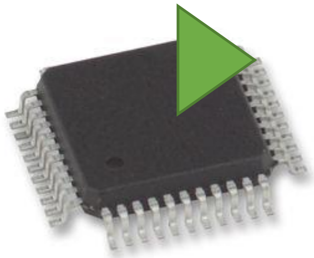
Registers

R0 = 10
R1 = 3
...
PC = 0x52

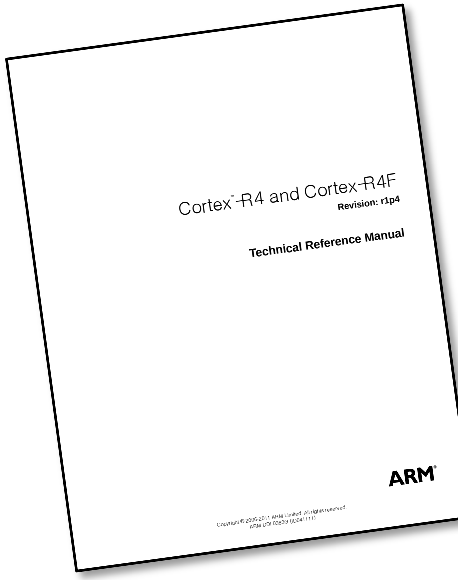
Halting Debug-Mode

Control handed
to external debugger

Monitor Debug-Mode



Abort Exception
→ Execution continues
in exception handler

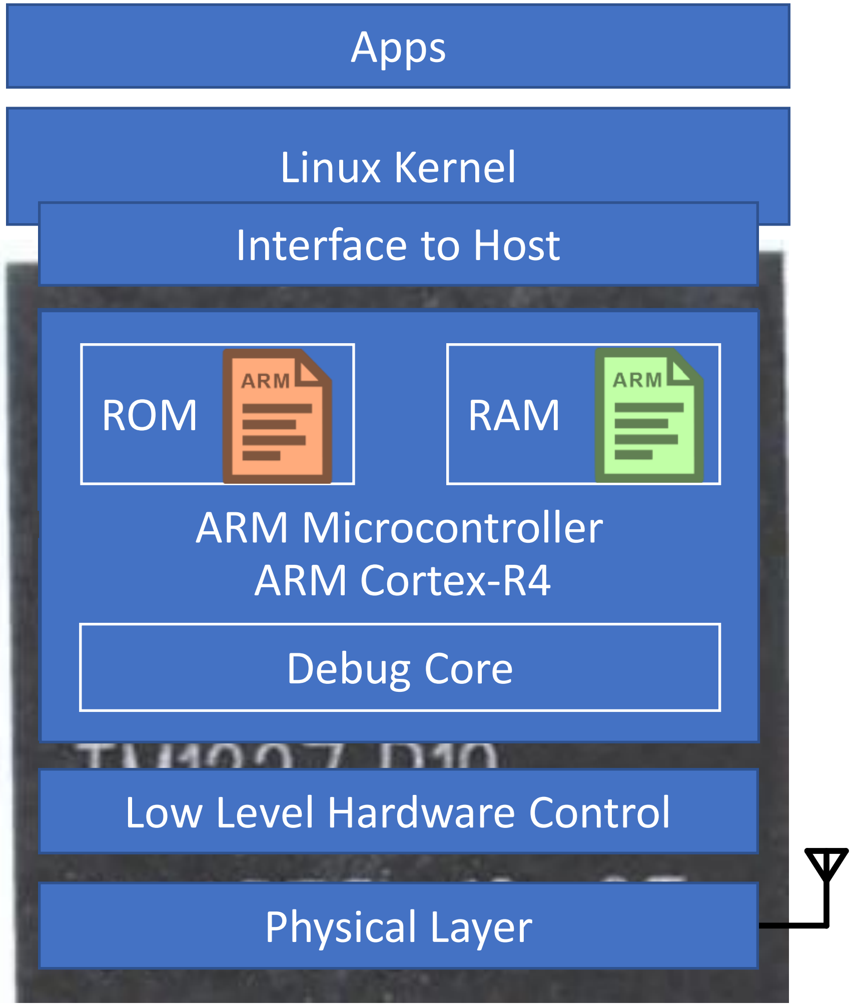


Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

Halting vs. Monitor Debug-Mode


Apps

Linux Kernel

Interface to Host

ARM Microcontroller
ARM Cortex-R4

Debug Core

Low Level Hardware Control

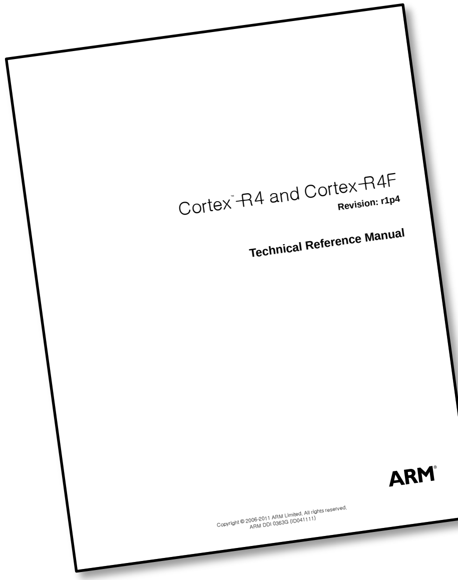
Physical Layer

Example Program

```
0x50: MOV R0, #10
0x52: MOV R1, #3
0x54: ADD R0, R0, R1
0x56: B 0x50
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x52
```


Cortex-R4 and Cortex-R4F
Revision: r1p4

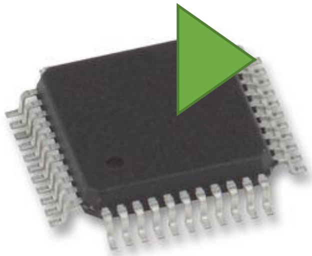
Technical Reference Manual

ARM

Halting Debug-Mode

Control handed
to external debugger

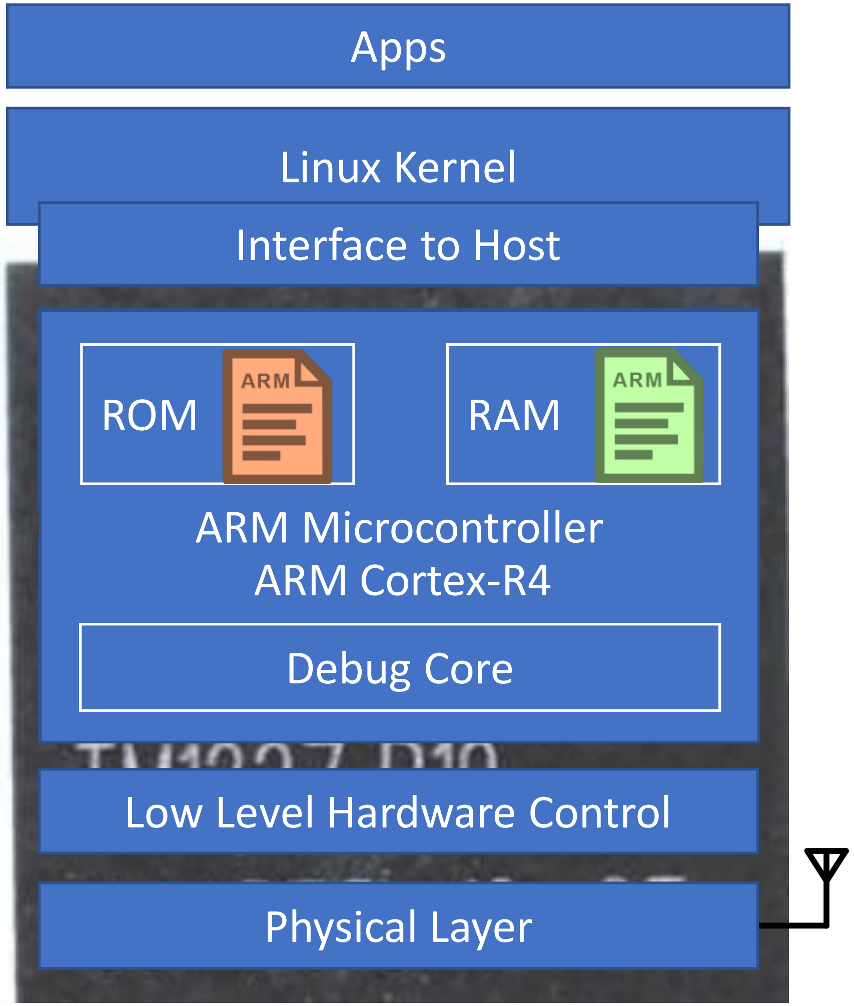
Monitor Debug-Mode



Abort Exception
→ Execution continues
in exception handler

What happens when an abort exception occurs?

Abort Exception Handling

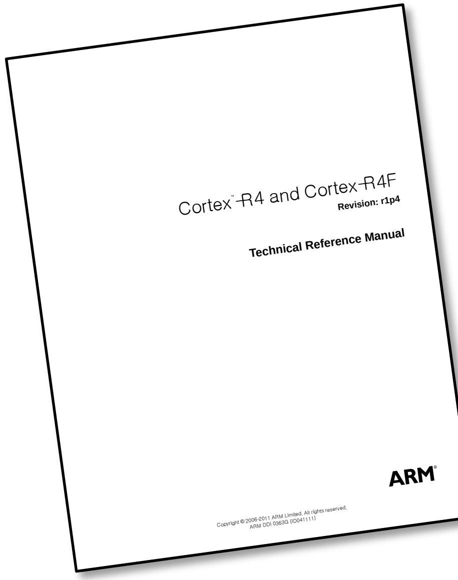


Example Program

```
0x50: MOV R0, #10
0x52: MOV R1, #3
0x54: ADD R0, R0, R1
0x56: B 0x50
```

Registers

R0 = 10
R1 = 3
...
PC = 0x52

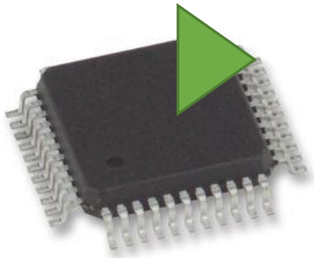


Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

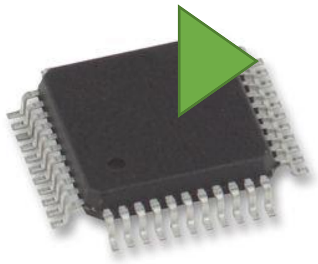
Monitor Debug-Mode



Abort Exception
→ Execution continues in exception handler

Abort Exception Handling

Monitor Debug-Mode



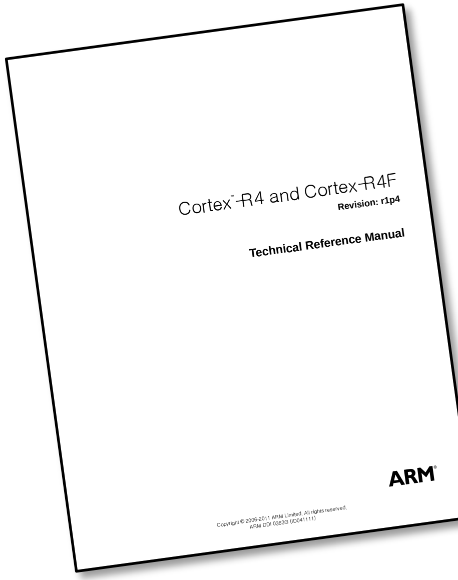
Abort Exception
→ Execution continues in exception handler

Example Program

```
Reset               0x00: B reset_hdl
Undef. Instr.       0x04: B undef_inst_hdl
Software Intr.      0x08: B sw_intr_hdl
Abort (prefetch)    0x0C: B pref_abt_hdl
Abort (data)        0x10: B data_abt_hdl
...                 ...
                    0x50: MOV R0, #10
                  ● 0x52: MOV R1, #3
                    0x54: ADD R0, R0, R1
                    0x56: B 0x50
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x52
```



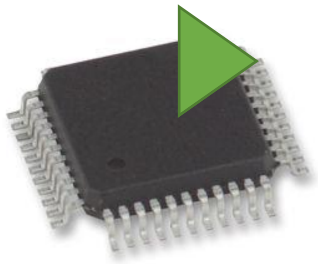
Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

Operating Modes

Monitor Debug-Mode



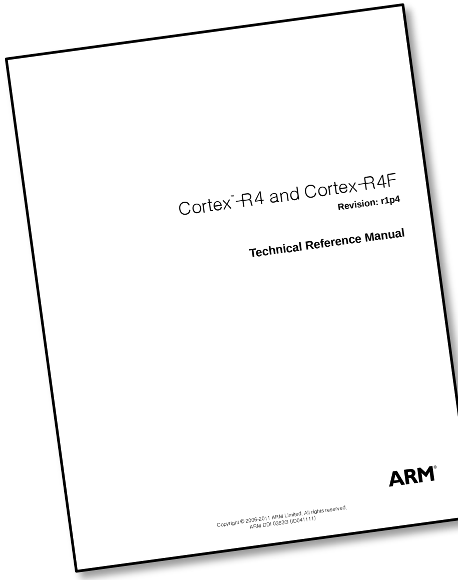
Abort Exception
→ Execution continues
in exception handler

Example Program

```
Reset                 0x00: B reset_hdl
Undef. Instr.         0x04: B undef_inst_hdl
Software Intr.        0x08: B sw_intr_hdl
Abort (prefetch) ➡    0x0C: B pref_abt_hdl
Abort (data)          0x10: B data_abt_hdl
...                   ...
                      0x50: MOV R0, #10
                 ●    0x52: MOV R1, #3
                      0x54: ADD R0, R0, R1
                      0x56: B 0x50
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x0C
```



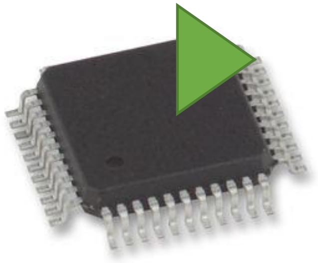
Cortex™-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM®

Operating Modes

Monitor Debug-Mode



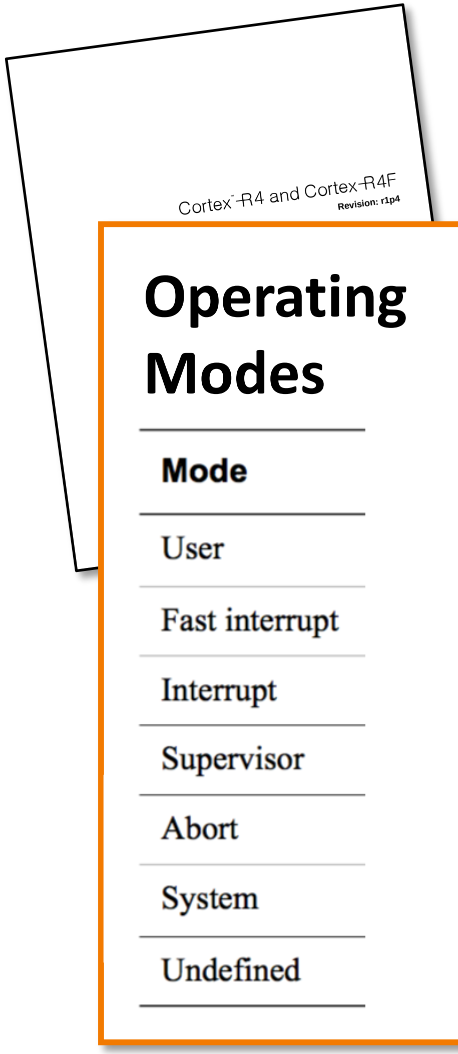
Abort Exception
→ Execution continues in exception handler

Example Program

```
Reset                 0x00: B reset_hdl
Undef. Instr.         0x04: B undef_inst_hdl
Software Intr.        0x08: B sw_intr_hdl
Abort (prefetch) ➡    0x0C: B pref_abt_hdl
Abort (data)          0x10: B data_abt_hdl
...                   ...
                      0x50: MOV R0, #10
                  ●   0x52: MOV R1, #3
                      0x54: ADD R0, R0, R1
                      0x56: B 0x50
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x0C
```



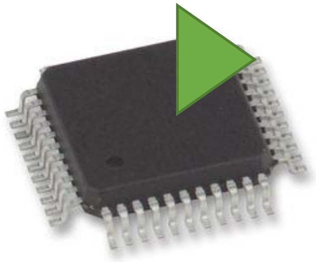
Cortex-R4 and Cortex-R4F
Revision: r1p4

Operating Modes

Mode
User
Fast interrupt
Interrupt
Supervisor
Abort
System
Undefined

Operating Modes

Monitor Debug-Mode



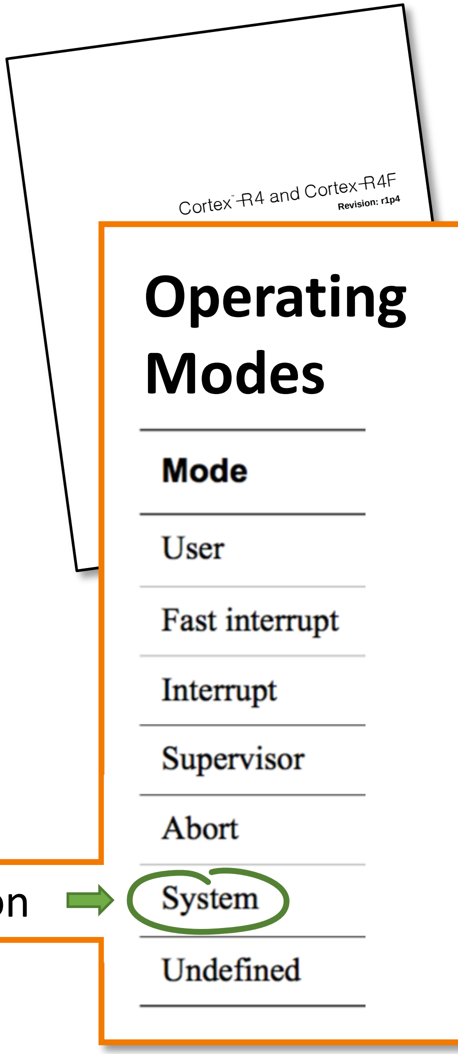
Abort Exception
→ Execution continues in exception handler

Example Program

```
Reset                0x00: B reset_hdl
Undef. Instr.        0x04: B undef_inst_hdl
Software Intr.       0x08: B sw_intr_hdl
Abort (prefetch) ➡   0x0C: B pref_abt_hdl
Abort (data)         0x10: B data_abt_hdl
...                  ...
                     0x50: MOV R0, #10
                 ●   0x52: MOV R1, #3
                     0x54: ADD R0, R0, R1
                     0x56: B 0x50
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x0C
```



Cortex-R4 and Cortex-R4F
Revision: r1p4

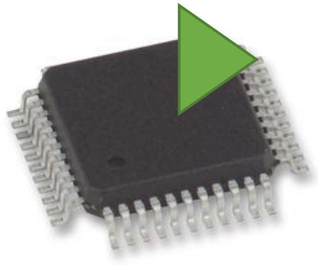
Operating Modes

Mode
User
Fast interrupt
Interrupt
Supervisor
Abort
System
Undefined

Regular Firmware Execution ➡

Operating Modes

Monitor Debug-Mode



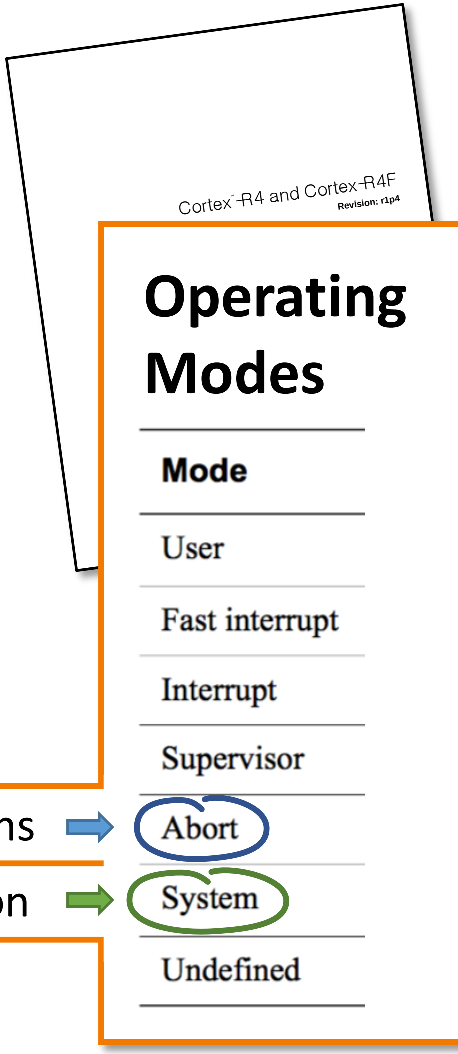
Abort Exception
→ Execution continues in exception handler

Example Program

```
Reset              0x00: B reset_hdl
Undef. Instr.      0x04: B undef_inst_hdl
Software Intr.     0x08: B sw_intr_hdl
Abort (prefetch) ➡ 0x0C: B pref_abt_hdl
Abort (data)       0x10: B data_abt_hdl
...                ...
                   0x50: MOV R0, #10
                 ● 0x52: MOV R1, #3
                   0x54: ADD R0, R0, R1
                   0x56: B 0x50
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x0C
```



Cortex-R4 and Cortex-R4F
Revision: r1p4

Operating Modes

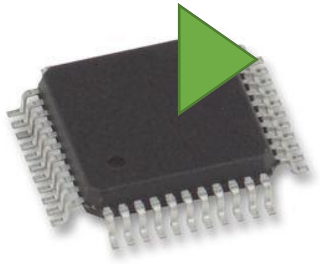
Mode
User
Fast interrupt
Interrupt
Supervisor
Abort
System
Undefined

For Handling Abort Exceptions ➡ Abort

Regular Firmware Execution ➡ System

Operating Modes

Monitor Debug-Mode



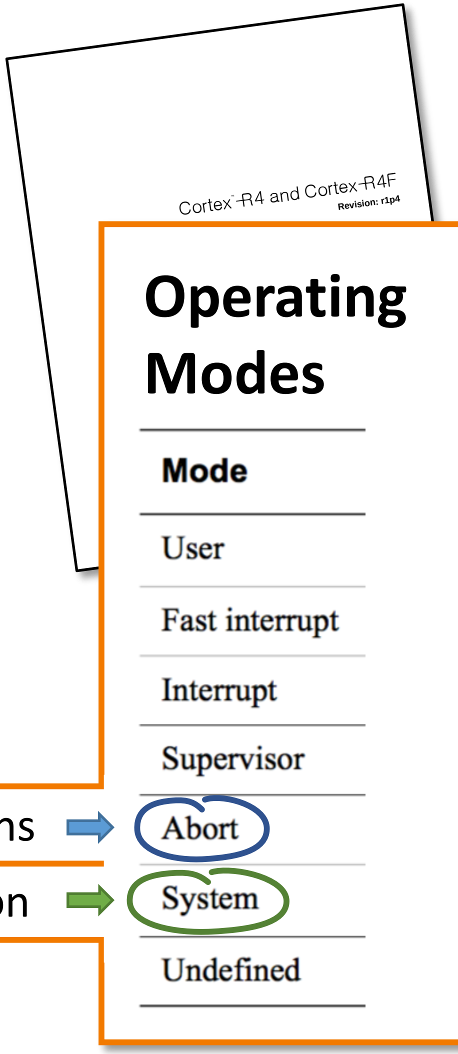
Abort Exception
→ Execution continues in exception handler

Example Program

```
Reset                 0x00: B reset_hdl
Undef. Instr.         0x04: B undef_inst_hdl
Software Intr.        0x08: B sw_intr_hdl
Abort (prefetch) ➡    0x0C: B pref_abt_hdl
Abort (data)          0x10: B data_abt_hdl
...                   ...
                      0x50: MOV R0, #10
                    ● 0x52: MOV R1, #3
                      0x54: ADD R0, R0, R1
                      0x56: B 0x50
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x0C
```



Cortex™-R4 and Cortex-R4F
Revision: r1p4

Operating Modes

Mode
User
Fast interrupt
Interrupt
Supervisor
Abort
System
Undefined

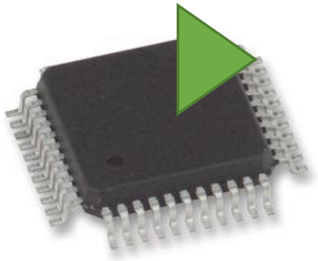
For Handling Abort Exceptions ➡ Abort

Regular Firmware Execution ➡ System

➡ Abort Mode
➡ System Mode

Operating Modes

Monitor Debug-Mode



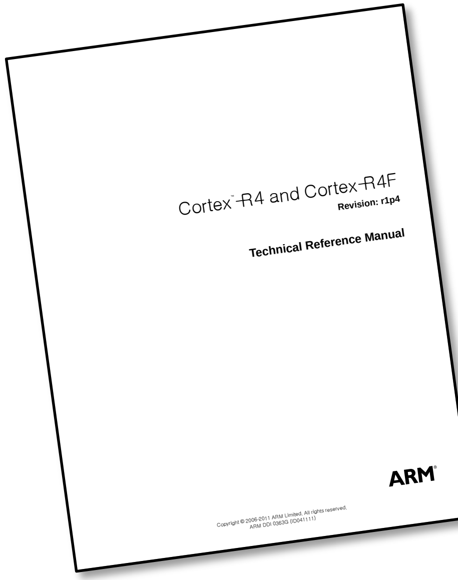
Abort Exception
→ Execution continues
in exception handler

Example Program

```
Reset                0x00: B reset_hdl
Undef. Instr.        0x04: B undef_inst_hdl
Software Intr.       0x08: B sw_intr_hdl
Abort (prefetch) ➡  0x0C: B pref_abt_hdl
Abort (data)         0x10: B data_abt_hdl
...                  ...
                     0x50: MOV R0, #10
                  ●  0x52: MOV R1, #3
                     0x54: ADD R0, R0, R1
                     0x56: B 0x50
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x0C
```

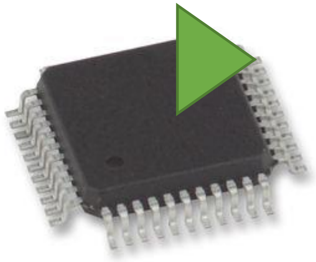


Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➡ Abort Mode
➡ System Mode

Banked Registers

Monitor Debug-Mode



Abort Exception

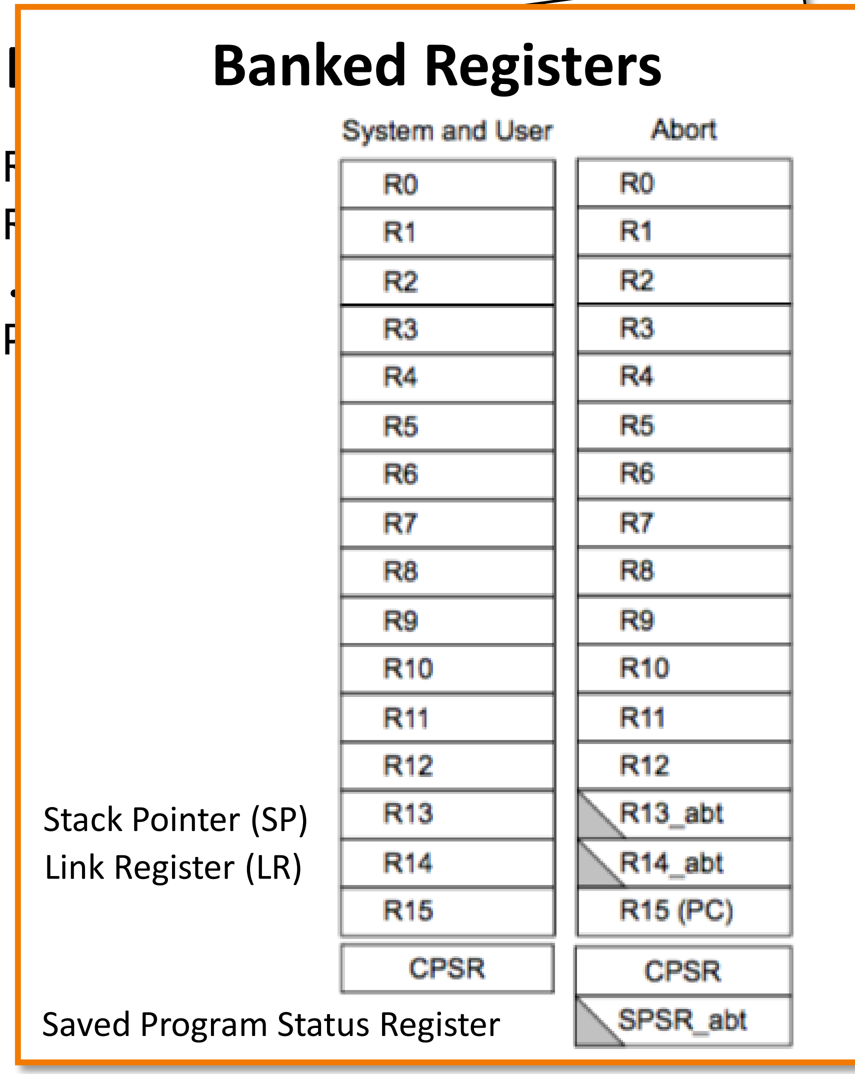
→ Execution continues in exception handler

Example Program

```
Reset              0x00: B reset_hdl
Undef. Instr.      0x04: B undef_inst_hdl
Software Intr.     0x08: B sw_intr_hdl
Abort (prefetch) ➡ 0x0C: B pref_abt_hdl
Abort (data)       0x10: B data_abt_hdl
...                ...
                   0x50: MOV R0, #10
                 ● 0x52: MOV R1, #3
                   0x54: ADD R0, R0, R1
                   0x56: B 0x50
```

➡ Abort Mode

➡ System Mode

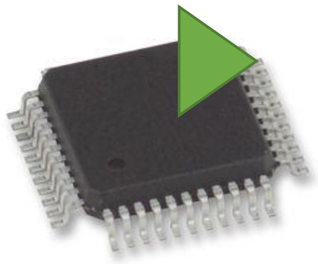


Banked Registers

	System and User	Abort
	R0	R0
	R1	R1
	R2	R2
	R3	R3
	R4	R4
	R5	R5
	R6	R6
	R7	R7
	R8	R8
	R9	R9
	R10	R10
	R11	R11
	R12	R12
Stack Pointer (SP)	R13	R13_abt
Link Register (LR)	R14	R14_abt
	R15	R15 (PC)
	CPSR	CPSR
Saved Program Status Register		SPSR_abt

Banked Registers

Monitor Debug-Mode



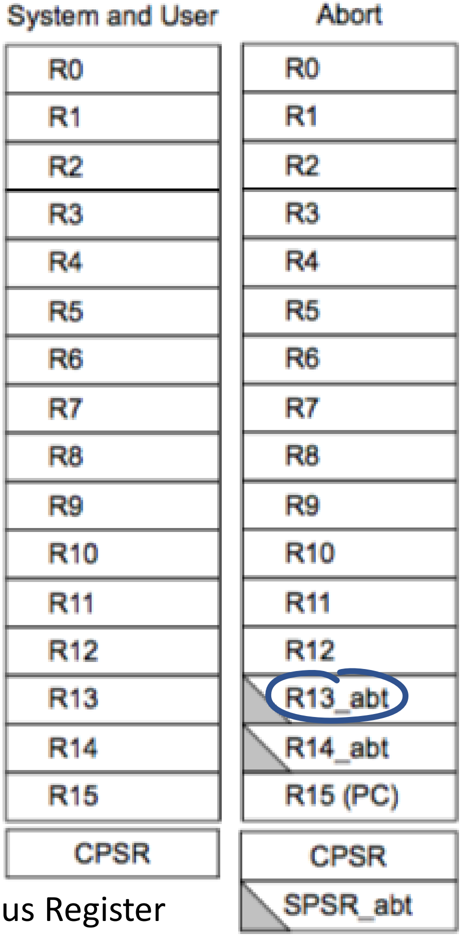
Abort Exception
→ Execution continues
in exception handler

Example Program

```
Reset                0x00: B reset_hdl
Undef. Instr.        0x04: B undef_inst_hdl
Software Intr.       0x08: B sw_intr_hdl
Abort (prefetch) ➡   0x0C: B pref_abt_hdl
Abort (data)         0x10: B data_abt_hdl
...                  ...
                     0x50: MOV R0, #10
                 ●   0x52: MOV R1, #3
                     0x54: ADD R0, R0, R1
                     0x56: B 0x50
```

➡ Abort Mode
➡ System Mode

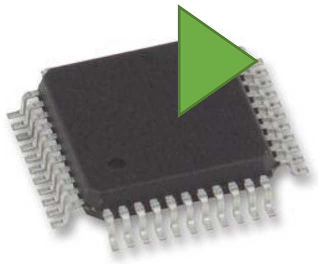
Banked Registers



	System and User	Abort
	R0	R0
	R1	R1
	R2	R2
	R3	R3
	R4	R4
	R5	R5
	R6	R6
	R7	R7
	R8	R8
	R9	R9
	R10	R10
	R11	R11
	R12	R12
Stack Pointer (SP)	R13	R13_abt
Link Register (LR)	R14	R14_abt
	R15	R15 (PC)
	CPSR	CPSR
Saved Program Status Register		SPSR_abt

Banked Registers

Monitor Debug-Mode

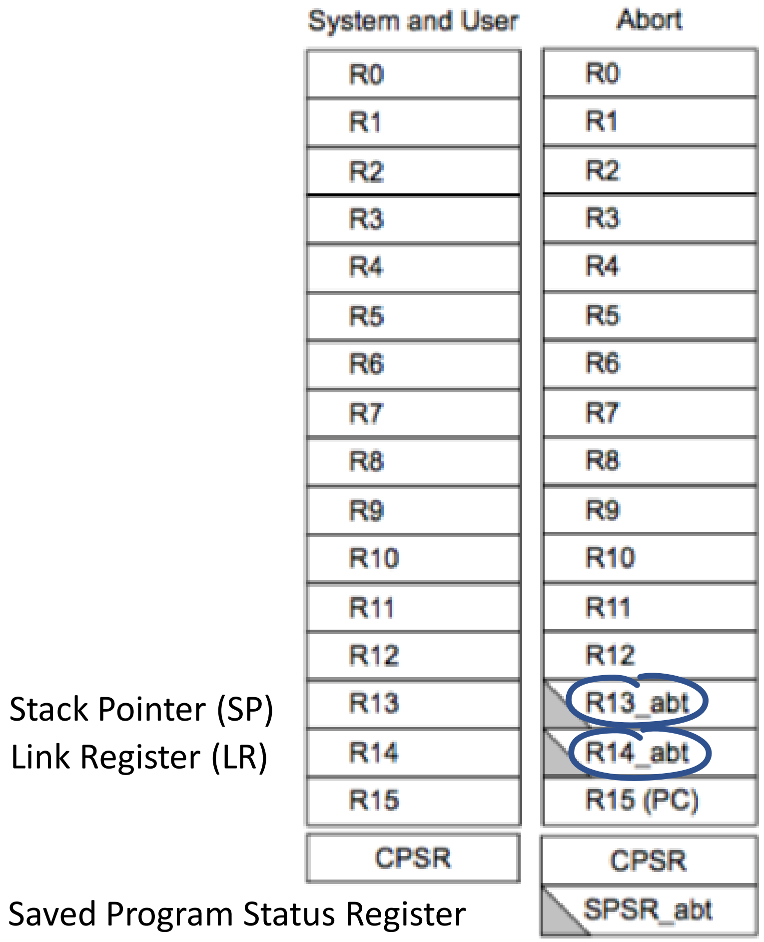


Abort Exception
→ Execution continues in exception handler

Example Program

```
Reset                 0x00: B reset_hdl
Undef. Instr.         0x04: B undef_inst_hdl
Software Intr.        0x08: B sw_intr_hdl
Abort (prefetch) ⇨    0x0C: B pref_abt_hdl
Abort (data)          0x10: B data_abt_hdl
...                   ...
                      0x50: MOV R0, #10
                 ●    0x52: MOV R1, #3
                      0x54: ADD R0, R0, R1
                      0x56: B 0x50
```

Banked Registers

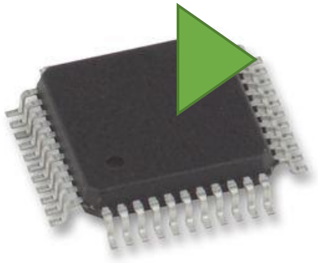


	System and User	Abort
	R0	R0
	R1	R1
	R2	R2
	R3	R3
	R4	R4
	R5	R5
	R6	R6
	R7	R7
	R8	R8
	R9	R9
	R10	R10
	R11	R11
	R12	R12
Stack Pointer (SP)	R13	R13_abt
Link Register (LR)	R14	R14_abt
	R15	R15 (PC)
	CPSR	CPSR
Saved Program Status Register		SPSR_abt

➡ Abort Mode
➡ System Mode

Banked Registers

Monitor Debug-Mode

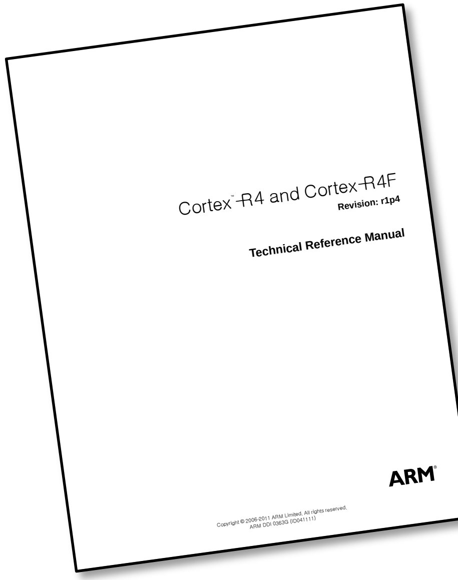


Abort Exception
→ Execution continues in exception handler

Example Program

Registers

```
Reset                 0x00: B reset_hdl        R0 = 10
Undef. Instr.         0x04: B undef_inst_hdl   R1 = 3
Software Intr.        0x08: B sw_intr_hdl      ...
Abort (prefetch)  ➡  0x0C: B pref_abt_hdl     PC = 0x0C
Abort (data)          0x10: B data_abt_hdl
...                   ...
                      0x50: MOV R0, #10
                  ●   0x52: MOV R1, #3
                      0x54: ADD R0, R0, R1
                      0x56: B 0x50
```

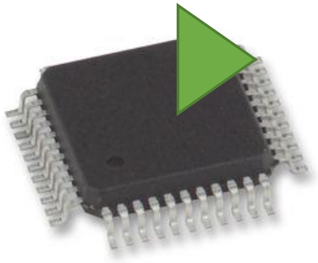


Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➡ Abort Mode
➡ System Mode

Banked Registers

Monitor Debug-Mode



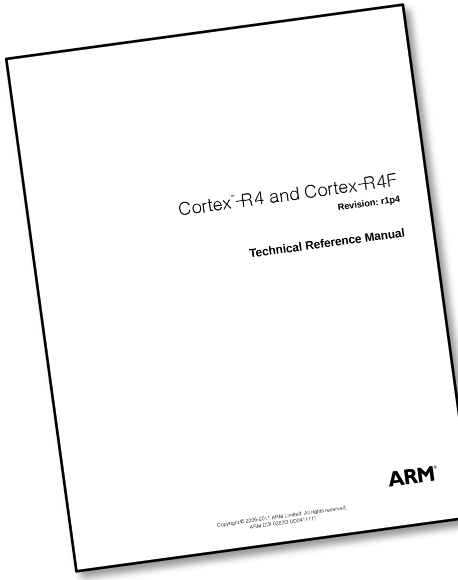
Abort Exception
→ Execution continues in exception handler

Example Program

```
Reset                0x00: B reset_hdl
Undef. Instr.        0x04: B undef_inst_hdl
Software Intr.       0x08: B sw_intr_hdl
Abort (prefetch)  ➡  0x0C: B pref_abt_hdl
Abort (data)         0x10: B data_abt_hdl
...                  ...
                     0x50: MOV R0, #10
                  ●  0x52: MOV R1, #3
                     0x54: ADD R0, R0, R1
                     0x56: B 0x50
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x0C
SP>R13_abt
LR>R14_abt
```



Cortex-R4 and Cortex-R4F
Revision: r1p4

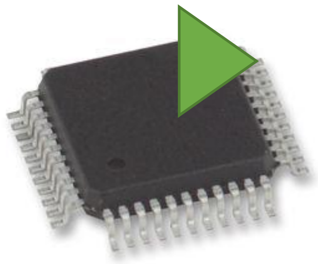
Technical Reference Manual

ARM

- ➡ Abort Mode
- ➡ System Mode

Banked Registers

Monitor Debug-Mode



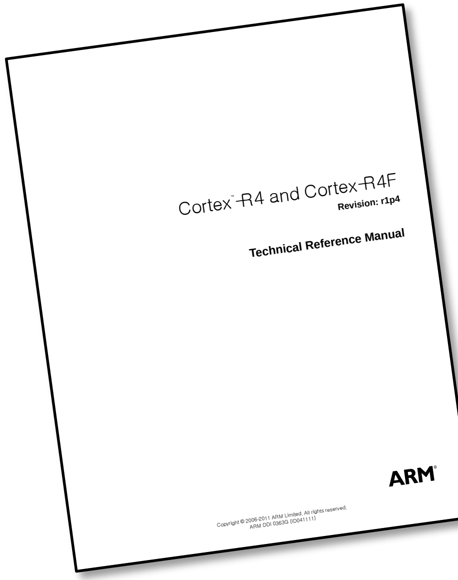
Abort Exception
→ Execution continues in exception handler

Example Program

```
Reset               0x00: B reset_hdl
Undef. Instr.       0x04: B undef_inst_hdl
Software Intr.      0x08: B sw_intr_hdl
Abort (prefetch) ➡ 0x0C: B pref_abt_hdl
Abort (data)        0x10: B data_abt_hdl
...                 ...
                    0x50: MOV R0, #10
                  ● 0x52: MOV R1, #3
                    0x54: ADD R0, R0, R1
                    0x56: B 0x50
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x0C
SP>R13_abt
LR>R14_abt
```



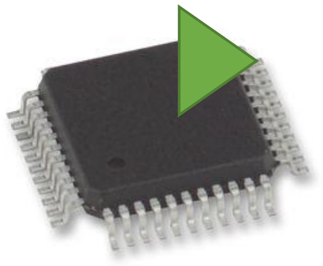
Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➔ **We can call functions within our Abort Handler without overwriting the regluar Stack Pointer and Link Register!**

➡ Abort Mode
➡ System Mode

Banked Registers

Monitor Debug-Mode



Abort Exception
→ Execution continues
in exception handler

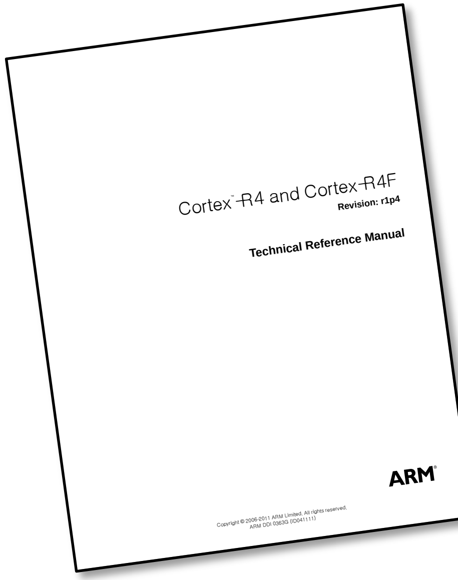
➡ Abort Mode
➡ System Mode

Example Program

```
Reset                 0x00: B reset_hdl
Undef. Instr.         0x04: B undef_inst_hdl
Software Intr.        0x08: B sw_intr_hdl
Abort (prefetch)  ➡  0x0C: B pref_abt_hdl
Abort (data)          0x10: B data_abt_hdl
...                   ...
                      0x50: MOV R0, #10
                  ●  0x52: MOV R1, #3
                      0x54: ADD R0, R0, R1
                      0x56: B 0x50
                      ...
    pref_abt_hdl      0x80: MOV SP, LR
                      ...
                      0x86: SRSDB SP!, #0x1F
                      0x8C: CPS #0x1F
                      ...
                      0x90: B handle_exceptions
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x0C
SP>R13_abt
LR>R14_abt
```

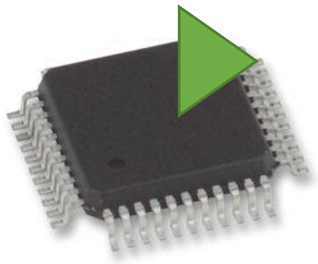


Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➔ We can call functions within our Abort Handler without overwriting the regluar Stack Pointer and Link Register!

Banked Registers

Monitor Debug-Mode



Abort Exception
→ Execution continues in exception handler

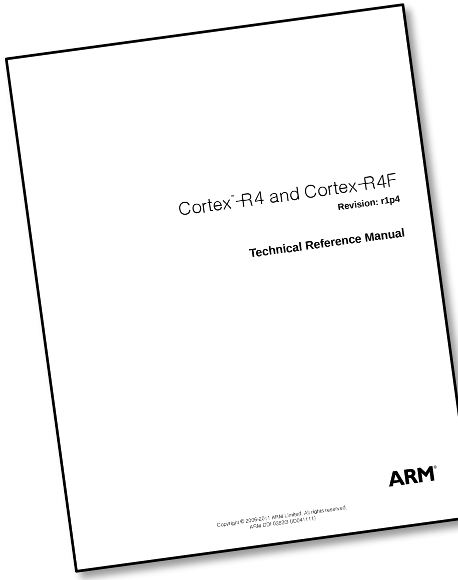
➡ Abort Mode
➡ System Mode

Example Program

```
Reset                0x00: B reset_hdl
Undef. Instr.        0x04: B undef_inst_hdl
Software Intr.       0x08: B sw_intr_hdl
Abort (prefetch)     0x0C: B pref_abt_hdl
Abort (data)         0x10: B data_abt_hdl
...                  ...
                     0x50: MOV R0, #10
                   ● 0x52: MOV R1, #3
                     0x54: ADD R0, R0, R1
                     0x56: B 0x50
                     ...
    pref_abt_hdl ➡ 0x80: MOV SP, LR
                     ...
                     0x86: SRSDB SP!, #0x1F
                     0x8C: CPS #0x1F
                     ...
                     0x90: B handle_exceptions
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x80
SP>R13_abt
LR>R14_abt
```

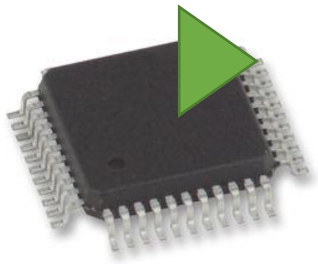


Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

→ We can call functions within our Abort Handler without overwriting the regluar Stack Pointer and Link Register!

Prefetch Abort Handler

Monitor Debug-Mode



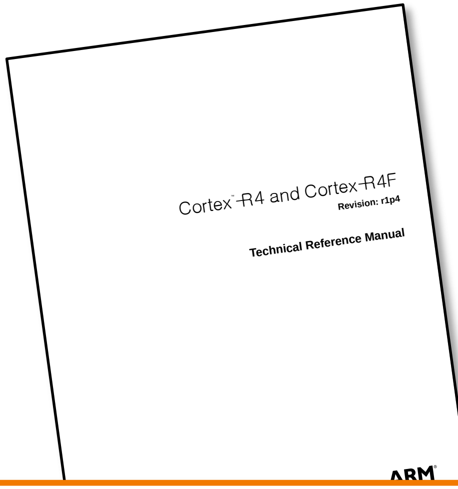
Abort Exception
→ Execution continues in exception handler

Example Program

```
Reset                0x00: B reset_hdl
Undef. Instr.        0x04: B undef_inst_hdl
Software Intr.       0x08: B sw_intr_hdl
Abort (prefetch)     0x0C: B pref_abt_hdl
Abort (data)         0x10: B data_abt_hdl
...                  ...
                     0x50: MOV R0, #10
                   ● 0x52: MOV R1, #3
                     0x54: ADD R0, R0, R1
                     0x56: B 0x50
                     ...
    pref_abt_hdl     0x80: MOV SP, LR
                     ...
                   ➡ 0x86: SRSDB SP!, #0x1F
                     0x8C: CPS #0x1F
                     ...
                     0x90: B handle_exceptions
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x86
SP>R13_abt
L
```



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

SRSDB Instruction

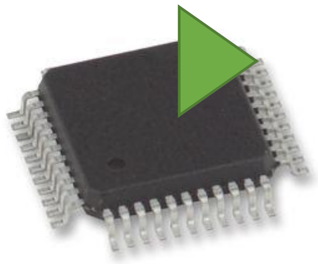
`SRSDB SP!, #0x1F`

SRS	Store Return State (LR, SPSR) onto a Stack
DB	Decrement address before each transfer
!	Write final address back to SP of mode 0x1F (system mode)

➡ Abort Mode
➡ System Mode

Prefetch Abort Handler

Monitor Debug-Mode



Abort Exception
→ Execution continues in exception handler

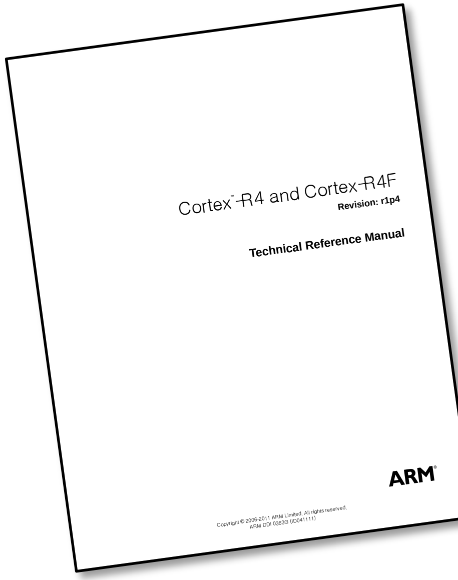
➡ Abort Mode
➡ System Mode

Example Program

```
Reset                    0x00: B reset_hdl
Undef. Instr.            0x04: B undef_inst_hdl
Software Intr.           0x08: B sw_intr_hdl
Abort (prefetch)         0x0C: B pref_abt_hdl
Abort (data)             0x10: B data_abt_hdl
...                      ...
                         0x50: MOV R0, #10
                       ● 0x52: MOV R1, #3
                         0x54: ADD R0, R0, R1
                         0x56: B 0x50
                         ...
    pref_abt_hdl         0x80: MOV SP, LR
                         ...
                         0x86: SRSDB SP!, #0x1F
                       ➡ 0x8C: CPS #0x1F
                         ...
                         0x90: B handle_exceptions
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x8C
SP>R13_abt
LR>R14_abt
```

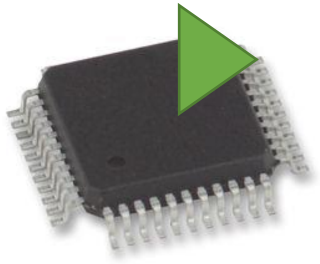


Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

→ We can call functions within our Abort Handler without overwriting the regluar Stack Pointer and Link Register!

Prefetch Abort Handler

Monitor Debug-Mode



Abort Exception
→ Execution continues in exception handler

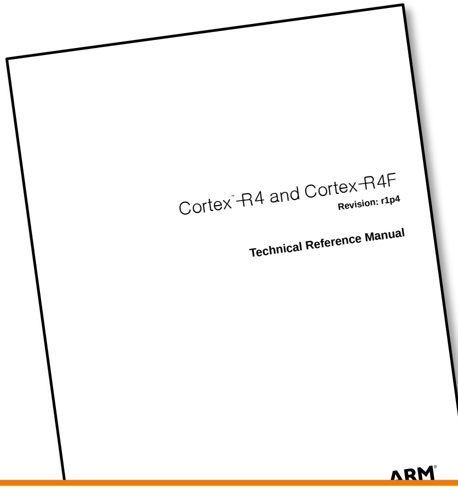
➡ Abort Mode
➡ System Mode

Example Program

```
Reset                  0x00: B reset_hdl
Undef. Instr.          0x04: B undef_inst_hdl
Software Intr.         0x08: B sw_intr_hdl
Abort (prefetch)       0x0C: B pref_abt_hdl
Abort (data)           0x10: B data_abt_hdl
...                    ...
                       0x50: MOV R0, #10
                     ● 0x52: MOV R1, #3
                       0x54: ADD R0, R0, R1
                       0x56: B 0x50
                       ...
      pref_abt_hdl     0x80: MOV SP, LR
                       ...
                       0x86: SRSDB SP!, #0x1F
                     ➡ 0x8C: CPS #0x1F
                       ...
                       0x90: B handle_exceptions
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x8C
SP>R13_abt
```



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

CPS Instruction

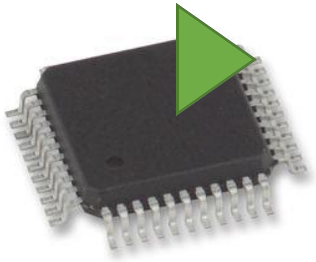
`CPS #0x1F`

CPS	Change Processor State
#0x1F	target mode = 0x1F (system mode)

regluar Stack Pointer and Link Register!

Prefetch Abort Handler

Monitor Debug-Mode



Abort Exception
→ Execution continues
in exception handler

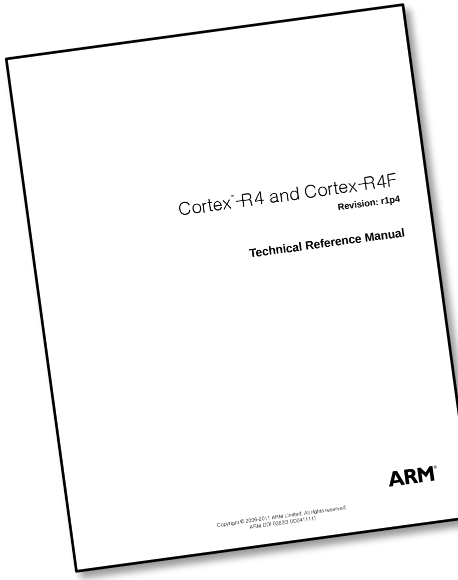
➡ Abort Mode
➡ System Mode

Example Program

```
Reset                 0x00: B reset_hdl
Undef. Instr.         0x04: B undef_inst_hdl
Software Intr.        0x08: B sw_intr_hdl
Abort (prefetch)      0x0C: B pref_abt_hdl
Abort (data)          0x10: B data_abt_hdl
...                   ...
                      0x50: MOV R0, #10
                    ● 0x52: MOV R1, #3
                      0x54: ADD R0, R0, R1
                      0x56: B 0x50
                      ...
    pref_abt_hdl      0x80: MOV SP, LR
                      ...
                      0x86: SRSDB SP!, #0x1F
                      0x8C: CPS #0x1F
                      ...
                    ➡ 0x90: B handle_exceptions
```

Registers

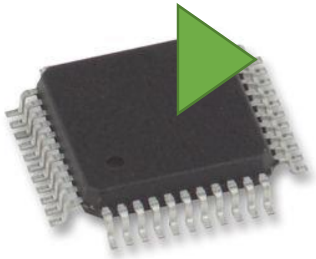
```
R0 = 10
R1 = 3
...
PC = 0x90
SP>R13
LR>R14
```



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

Prefetch Abort Handler

Monitor Debug-Mode



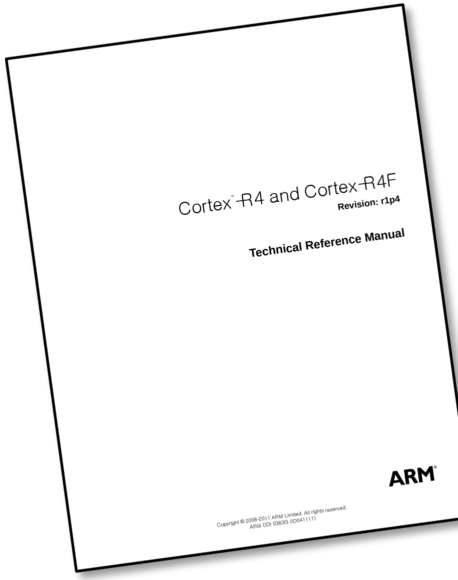
Abort Exception
→ Execution continues in exception handler

Example Program

```
Reset                 0x00: B reset_hdl
Undef. Instr.         0x04: B undef_inst_hdl
Software Intr.        0x08: B sw_intr_hdl
Abort (prefetch)      0x0C: B pref_abt_hdl
Abort (data)          0x10: B data_abt_hdl
...                   ...
                      0x50: MOV R0, #10
                    ● 0x52: MOV R1, #3
                      0x54: ADD R0, R0, R1
                      0x56: B 0x50
                      ...
   pref_abt_hdl       0x80: MOV SP, LR
                      ...
                      0x86: SRSDB SP!, #0x1F
                      0x8C: CPS #0x1F
                      ...
                    ➡ 0x90: B handle_exceptions
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x90
SP>R13
LR>R14
```



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➔ **Original Wi-Fi firmware always changes to System mode**

➔ `handle_exceptions` **handles all exceptions**

➡ Abort Mode
➡ System Mode

Prefetch Abort Handler

Monitor Debug-Mode

Reset

pref_abt_hdl

ToDos to Create DIY Debugger

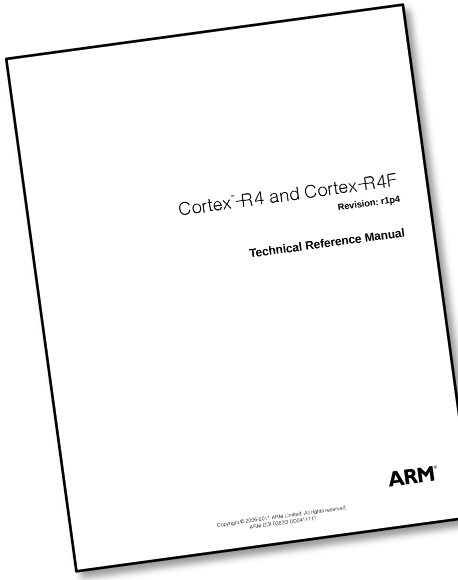
- Stay in Abort Mode
- Save Register State to Abort Mode Stack
    - Initialize ABT Stack Pointer
- Analyze `handle_exceptions` function
- Implement a breakpoint handler
- Activate breakpoints

Example Program

```
  0x00: B reset_hdl
  0x04: B undef_inst_hdl
  0x08: B sw_intr_hdl
  0x0C: B pref_abt_hdl
  0x10: B data_abt_hdl
  ...
  0x50: MOV R0, #10
  0x52: MOV R1, #3
  0x54: ADD R0, R0, R1
  0x56: B 0x50
  ...
  0x80: MOV SP, LR
  ...
  0x86: SRSDB SP!, #0x1F
  0x8C: CPS #0x1F
  ...
➡ 0x90: B handle_exceptions
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x90
SP>R13
LR>R14
```



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➔ Original Wi-Fi firmware always changes to System mode

➔ `handle_exceptions` handles all exceptions

➡ Abort Mode
➡ System Mode

Prefetch Abort Handler

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode
- Save Register State to Abort Mode Stack
  - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

pref_abt_hdl

➡ Abort Mode

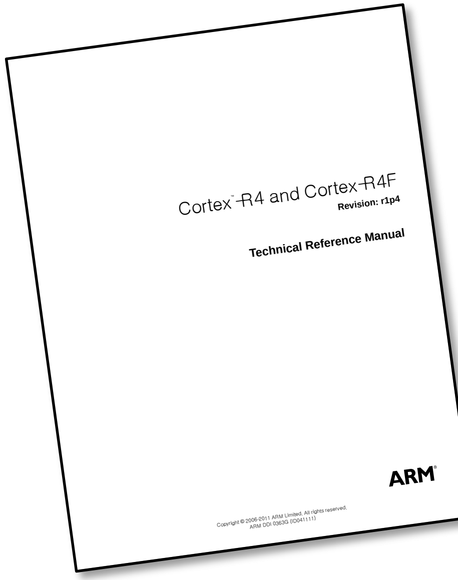
➡ System Mode

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x50: MOV R0, #10
0x52: MOV R1, #3
0x54: ADD R0, R0, R1
0x56: B 0x50
...
0x80: MOV SP, LR
...
0x86: SRSDB SP!, #0x1F
0x8C: ~~CPS #0x1F~~
...
➡ 0x90: B handle_exceptions
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x90
SP>R13
LR>R14
```



Cortex™-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM®

➔ **Original Wi-Fi firmware always changes to System mode**

➔ `handle_exceptions` **handles all exceptions**

Prefetch Abort Handler

Monitor Debug-Mode

Reset

pref_abt_hdl

ToDos to Create DIY Debugger

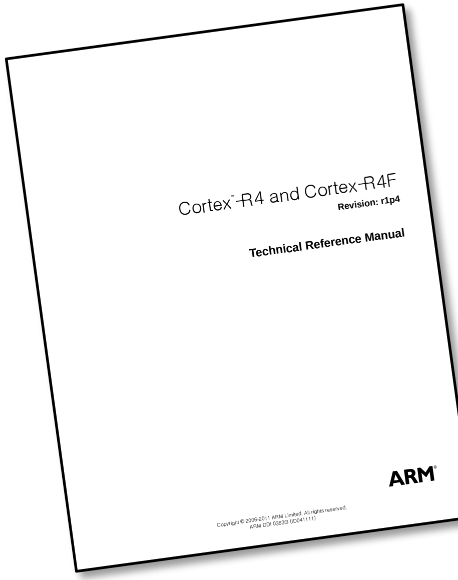
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack
    - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x50: MOV R0, #10
0x52: MOV R1, #3
0x54: ADD R0, R0, R1
0x56: B 0x50
...
0x80: MOV SP, LR
...
0x86: SRSDB SP!, #0x1F
0x8C: ~~CPS #0x1F~~
...
➡ 0x90: B handle_exceptions
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x90
SP>R13_abt
LR>R14_abt
```



Cortex-R4 and Cortex-R4F

Revision: r1p4

Technical Reference Manual

ARM

➔ **Original Wi-Fi firmware always changes to System mode**

➔ `handle_exceptions` **handles all exceptions**

➡ Abort Mode

➡ System Mode

Prefetch Abort Handler

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

pref_abt_hdl

➡ Abort Mode

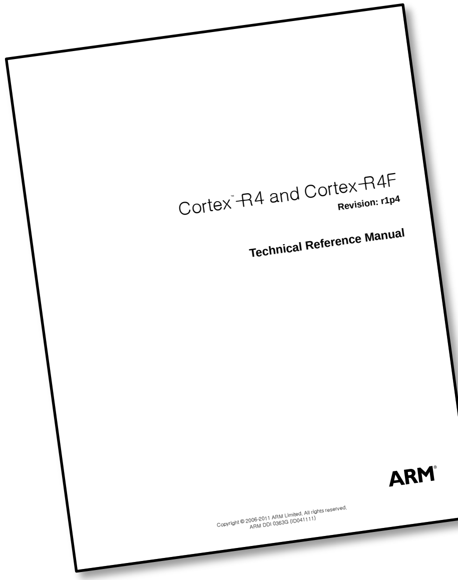
➡ System Mode

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x50: MOV R0, #10
0x52: MOV R1, #3
0x54: ADD R0, R0, R1
0x56: B 0x50
...
0x80: MOV SP, LR
...                  #0x17
0x86: SRSDB SP!, ~~#0x1F~~
0x8C: ~~CPS #0x1F~~
...
➡ 0x90: B handle_exceptions
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x90
SP>R13_abt
LR>R14_abt
```



Cortex-R4 and Cortex-R4F

Revision: r1p4

Technical Reference Manual

ARM

➔ **Original Wi-Fi firmware always changes to System mode**

➔ `handle_exceptions` **handles all exceptions**

Initialize Abort Stack Pointer

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

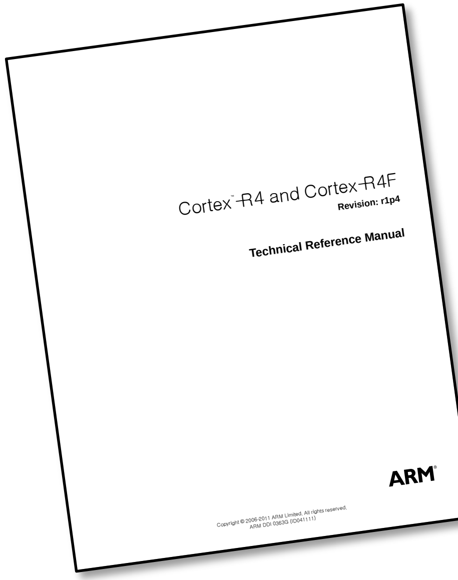
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer
- Analyze `handle_exceptions` function
- Implement a breakpoint handler
- Activate breakpoints

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x90
SP>R13_abt
LR>R14_abt
```



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➡ Abort Mode
➡ System Mode

Initialize Abort Stack Pointer

Monitor Debug-Mode

`Reset`

ToDos to Create DIY Debugger

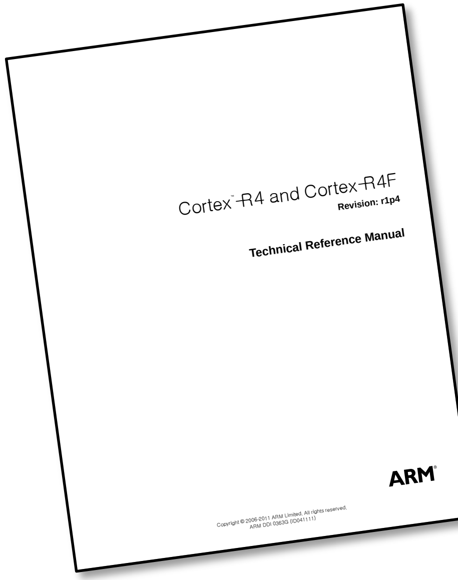
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer
- Analyze `handle_exceptions` function
- Implement a breakpoint handler
- Activate breakpoints

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x90
SP>R13_abt
LR>R14_abt
```



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➡ Abort Mode
➡ System Mode

Initialize Abort Stack Pointer

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

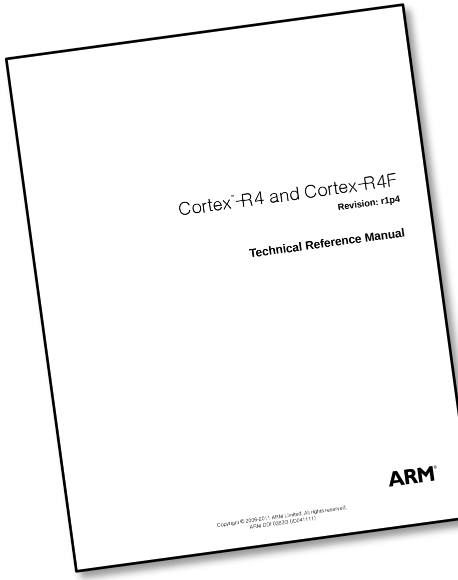
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer
- Analyze `handle_exceptions` function
- Implement a breakpoint handler
- Activate breakpoints

Example Program

```
➡ 0x00: B reset_hdl
  0x04: B undef_inst_hdl
  0x08: B sw_intr_hdl
  0x0C: B pref_abt_hdl
  0x10: B data_abt_hdl
  ...
```

Registers

```
R0 = UNDEF
R1 = UNDEF
...
PC = 0x00
SP>R13_svc
LR>R14_svc
```



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Initialize Abort Stack Pointer

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

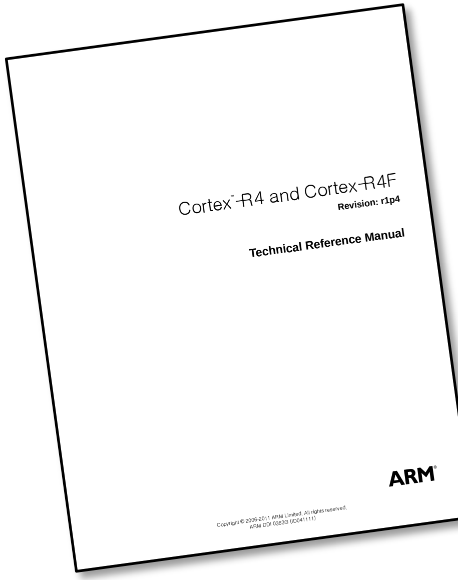
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

Example Program

```
➡ 0x00: B reset_hdl
  0x04: B undef_inst_hdl
  0x08: B sw_intr_hdl
  0x0C: B pref_abt_hdl
  0x10: B data_abt_hdl
  ...
  0x30: Check exception vectors
  0x32: Change to System Mode
  0x34: B setup
```

Registers

```
R0 = UNDEF
R1 = UNDEF
...
PC = 0x00
SP>R13_svc
LR>R14_svc
```



Cortex™-R4 and Cortex-R4F

Revision: r1p4

Technical Reference Manual

ARM®

➡ Supervisor Mode

➡ Abort Mode

➡ System Mode

Initialize Abort Stack Pointer

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

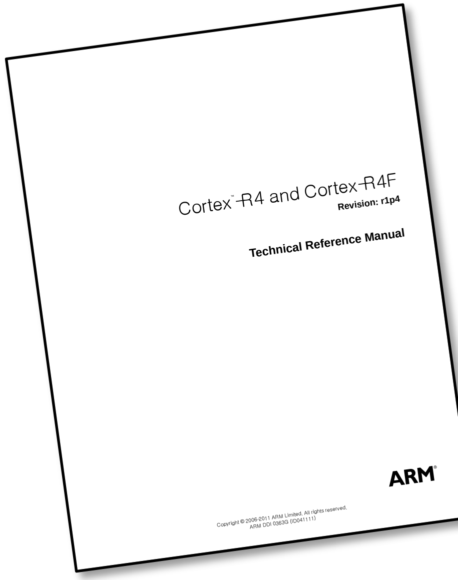
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

Example Program

```
  0x00: B reset_hdl
  0x04: B undef_inst_hdl
  0x08: B sw_intr_hdl
  0x0C: B pref_abt_hdl
  0x10: B data_abt_hdl
  ...
➡ 0x30: Check exception vectors
  0x32: Change to System Mode
  0x34: B setup
```

Registers

```
R0 = UNDEF
R1 = UNDEF
...
PC = 0x30
SP>R13_svc
LR>R14_svc
```



Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Initialize Abort Stack Pointer

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

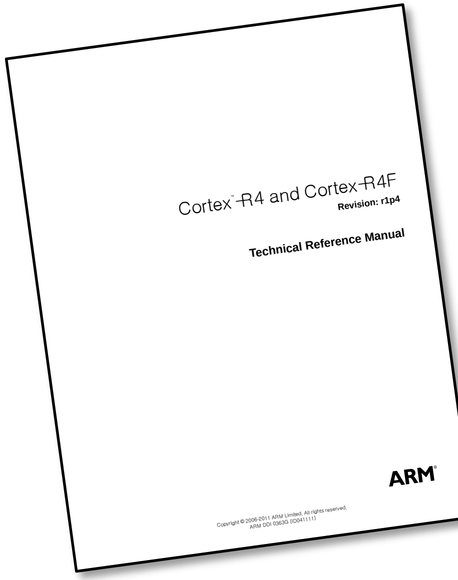
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x30: Check exception vectors
0x32: Change to System Mode
0x34: B setup
```

Registers

```
R0 = ...
R1 = ...
...
PC = 0x32
SP>R13_svc
LR>R14_svc
```



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Initialize Abort Stack Pointer

Monitor Debug-Mode

`Reset`

ToDos to Create DIY Debugger

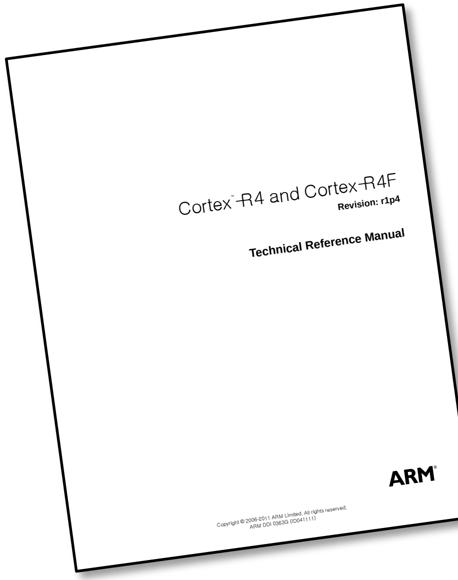
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

Example Program

```
  0x00: B reset_hdl
  0x04: B undef_inst_hdl
  0x08: B sw_intr_hdl
  0x0C: B pref_abt_hdl
  0x10: B data_abt_hdl
  ...
  0x30: Check exception vectors
  0x32: Change to System Mode
➡ 0x34: B setup
```

Registers

```
R0 = ...
R1 = ...
...
PC = 0x32
SP>R13
LR>R14
```



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Initialize Abort Stack Pointer

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

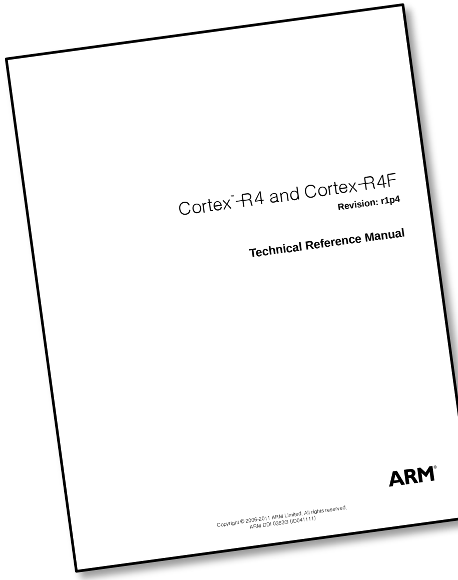
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

Example Program

```
  0x00: B reset_hdl
  0x04: B undef_inst_hdl
  0x08: B sw_intr_hdl
  0x0C: B pref_abt_hdl
  0x10: B data_abt_hdl
  ...
  0x30: Check exception vectors
  0x32: Change to System Mode
➡ 0x34: B setup
  ...
  0x40: Initialize processor
  0x42: B c_main
```

Registers

```
R0 = ...
R1 = ...
...
PC = 0x32
SP>R13
LR>R14
```



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Initialize Abort Stack Pointer

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

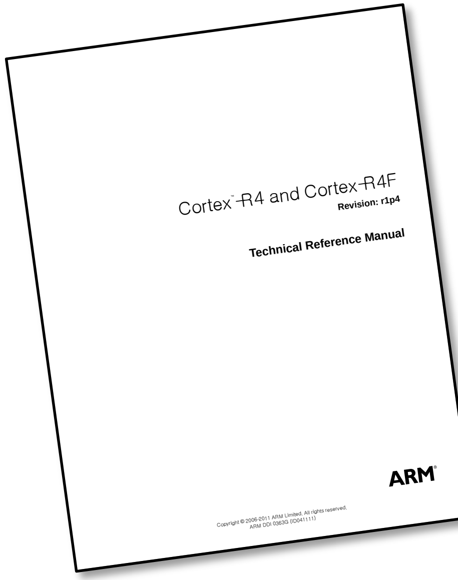
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x30: Check exception vectors
0x32: Change to System Mode
0x34: B setup
...
0x40: Initialize processor
0x42: B c_main
```

Registers

```
R0 = ...
R1 = ...
...
PC = 0x40
SP>R13
LR>R14
```



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

⇨ Supervisor Mode
⇨ Abort Mode
⇨ System Mode

Initialize Abort Stack Pointer

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

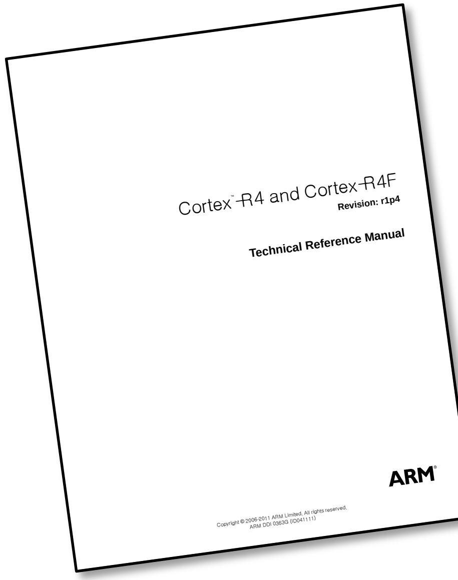
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

Example Program

```
  0x00: B reset_hdl
  0x04: B undef_inst_hdl
  0x08: B sw_intr_hdl
  0x0C: B pref_abt_hdl
  0x10: B data_abt_hdl
  ...
  0x30: Check exception vectors
  0x32: Change to System Mode
  0x34: B setup
  ...
  0x40: Initialize processor
➡ 0x42: B c_main
```

Registers

```
R0 = ...
R1 = ...
...
PC = 0x42
SP>R13
LR>R14
```


Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Initialize Abort Stack Pointer

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

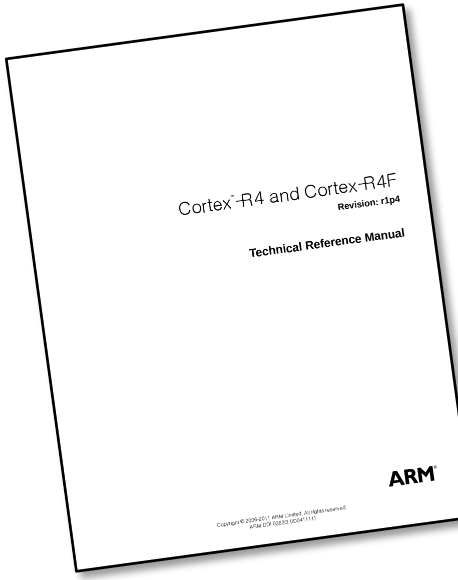
Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x30: Check exception vectors
0x32: Change to System Mode
0x34: B setup
...
0x40: Initialize processor
➡ 0x42: B c_main
...
0x60: Initialize the „driver“
0x62: Activate interrupts
0x64: Wait for interrupts
```

c_main

Registers

```
R0 = ...
R1 = ...
...
PC = 0x42
SP>R13
LR>R14
```



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Initialize Abort Stack Pointer

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

c_main

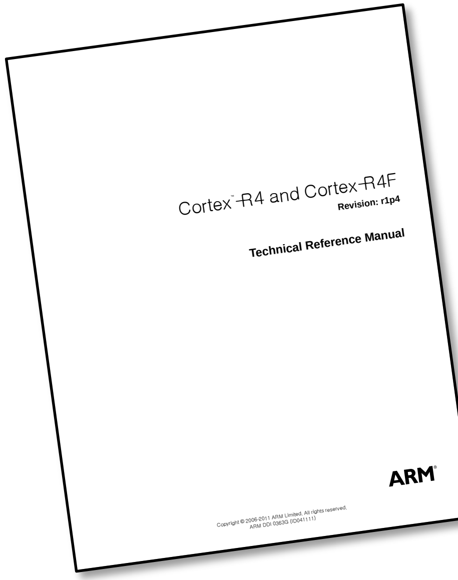
➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x30: Check exception vectors
0x32: Change to System Mode
0x34: B setup
...
0x40: Initialize processor
➡ 0x42: B ~~c_main~~ c_main_hook
...
0x60: Initialize the „driver“
0x62: Activate interrupts
0x64: Wait for interrupts
```

Registers

```
R0 = ...
R1 = ...
...
PC = 0x42
SP>R13
LR>R14
```



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

Initialize Abort Stack Pointer

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

c_main

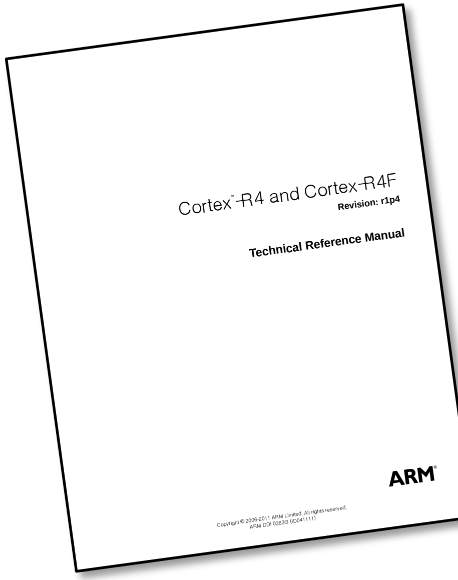
Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x30: Check exception vectors
0x32: Change to System Mode
0x34: B setup
...
0x40: Initialize processor
0x42: B ~~c_main~~ c_main_hook
...
0x60: Initialize the „driver“
0x62: Activate interrupts
0x64: Wait for interrupts
```

```
c_main_hook:
0x60: PUSH {R0-R3,LR}
0x62: BL set_abort_SP
0x64: POP {R0-R3,LR}
0x66: B c_main
```

Registers

R0 = ...
R1 = ...
...
PC = 0x42
SP>R13
LR>R14



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

Initialize Abort Stack Pointer

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode

➡ Abort Mode

➡ System Mode

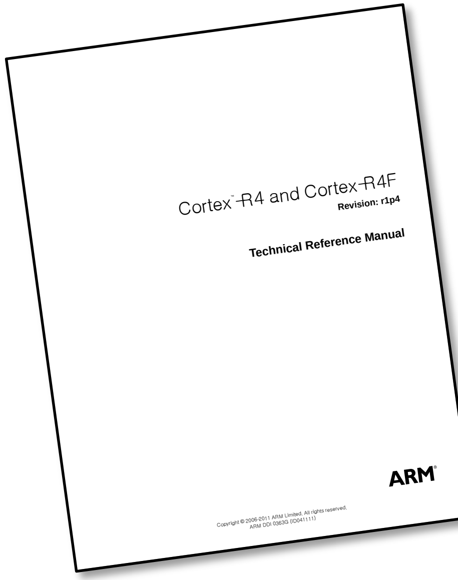
c_main

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x30: Check exception vectors
0x32: Change to System Mode
0x34: B setup
...
0x40: Initialize processor
➡ 0x42: B ~~c_main~~ c_main_hook
...
0x60: Initialize the „driver“
0x62: Activate interrupts
0x64: Wait for interrupts
```

Registers

```
R0 = ...
R1 = ...
...
PC = 0x42
SP>R13
LR>R14
```



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

```
c_main_hook:
0x60: PUSH {R0-R3,LR}
0x62: BL set_abort_SP
0x64: POP {R0-R3,LR}
0x66: B c_main
```

Initialize Abort Stack Pointer

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

c_main

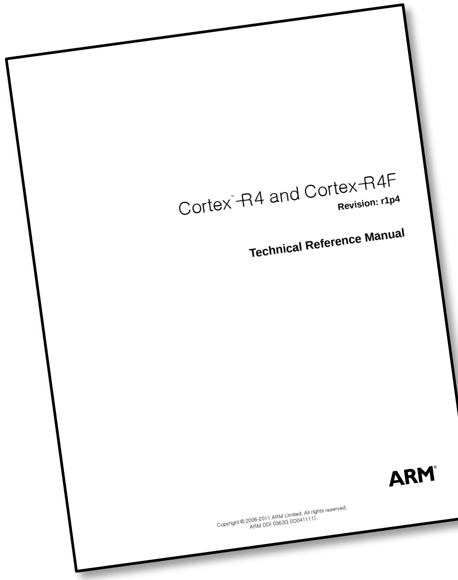
Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x30: Check exception vectors
0x32: Change to System Mode
0x34: B setup
...
0x40: Initialize processor
0x42: B ~~c_main~~ c_main_hook
...
0x60: Initialize the „driver“
0x62: Activate interrupts
0x64: Wait for interrupts
```

```
c_main_hook:
0x60: PUSH {R0-R3,LR}
0x62: BL set_abort_SP
0x64: POP {R0-R3,LR}
0x66: B c_main
```

Registers

```
R0 = ...
R1 = ...
...
PC = 0x42
SP>R13
LR>R14
```



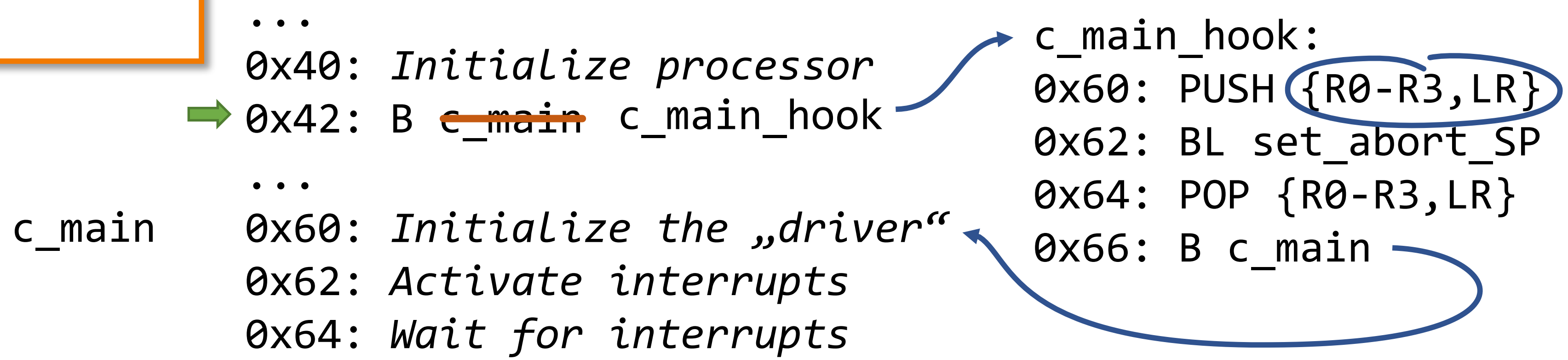
Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

Initialize Abort Stack Pointer

Monitor
Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints



...
0x40: *Initialize processor*
➡ 0x42: B ~~c_main~~ c_main_hook
...
c_main
0x60: *Initialize the „driver“*
0x62: *Activate interrupts*
0x64: *Wait for interrupts*

```
c_main_hook:
0x60: PUSH {R0-R3,LR}
0x62: BL set_abort_SP
0x64: POP {R0-R3,LR}
0x66: B c_main
```

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

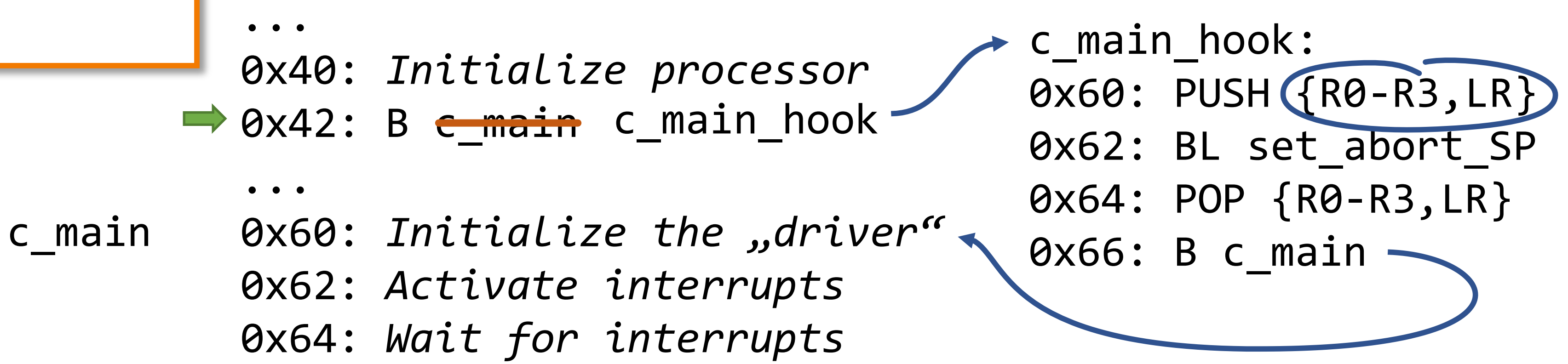
Initialize Abort Stack Pointer

Monitor
Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

```
uint32 stack_abt[256] = { 0x54424153 };

void __attribute__((optimize("O0")))
set_abort_SP(void) {
    register uint32 sp_abt asm("r0") = (uint32) &stack_abt[255];

    dbg_change_processor_mode(DBG_PROCESSOR_MODE_ABT);
    asm("mov sp, %[value]" : : [value] "r" (sp_abt));
    dbg_change_processor_mode(DBG_PROCESSOR_MODE_SYS);
}
```



...
0x40: Initialize processor
0x42: B ~~c_main~~ c_main_hook
...
c_main
0x60: Initialize the „driver“
0x62: Activate interrupts
0x64: Wait for interrupts

c_main_hook:
0x60: PUSH {R0-R3,LR}
0x62: BL set_abort_SP
0x64: POP {R0-R3,LR}
0x66: B c_main

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Initialize Abort Stack Pointer

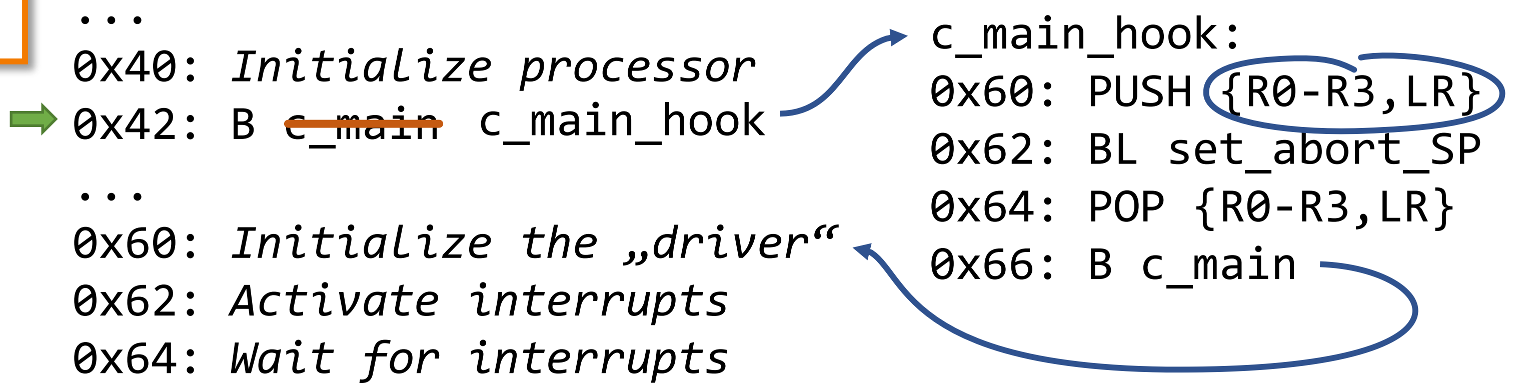
Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

```
uint32 stack_abt[256] = { 0x54424153 };

void __attribute__((optimize("O0")))
set_abort_SP(void) {
    register uint32 sp_abt asm("r0") = (uint32) &stack_abt[255];

    dbg_change_processor_mode(DBG_PROCESSOR_MODE_ABT);
    asm("mov sp, %[value]" : : [value] "r" (sp_abt));
    dbg_change_processor_mode(DBG_PROCESSOR_MODE_SYS);
}
```

Change to Abort Mode



...

0x40: *Initialize processor*

➡ 0x42: B ~~c_main~~ c_main_hook

...

c_main

0x60: *Initialize the „driver“*

0x62: *Activate interrupts*

0x64: *Wait for interrupts*

c_main_hook:

0x60: PUSH {R0-R3,LR}

0x62: BL set_abort_SP

0x64: POP {R0-R3,LR}

0x66: B c_main

➡ Supervisor Mode

➡ Abort Mode

➡ System Mode

Initialize Abort Stack Pointer

Monitor Debug-Mode

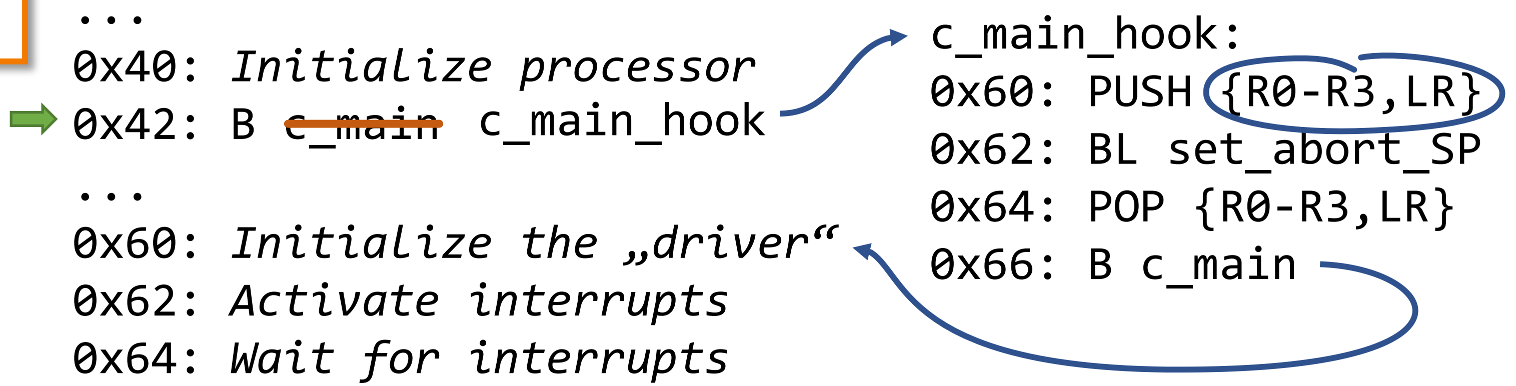
ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

```
uint32 stack_abt[256] = { 0x54424153 };

void __attribute__((optimize("O0")))
set_abort_SP(void) {
    register uint32 sp_abt asm("r0") = (uint32) &stack_abt[255];

    dbg_change_processor_mode(DBG_PROCESSOR_MODE_ABT);
    asm("mov sp, %[value]" : : [value] "r" (sp_abt));
    dbg_change_processor_mode(DBG_PROCESSOR_MODE_SYS);
}
```

Change to Abort Mode

Set Stack Pointer to end of stack_abt array



```
...
0x40: Initialize processor
0x42: B ~~c_main~~ c_main_hook
...
0x60: Initialize the „driver“
0x62: Activate interrupts
0x64: Wait for interrupts
```

c_main

```
c_main_hook:
0x60: PUSH {R0-R3,LR}
0x62: BL set_abort_SP
0x64: POP {R0-R3,LR}
0x66: B c_main
```

🟧 Supervisor Mode
🟦 Abort Mode
🟩 System Mode

Initialize Abort Stack Pointer

Monitor Debug-Mode

ToDos to Create DIY Debugger

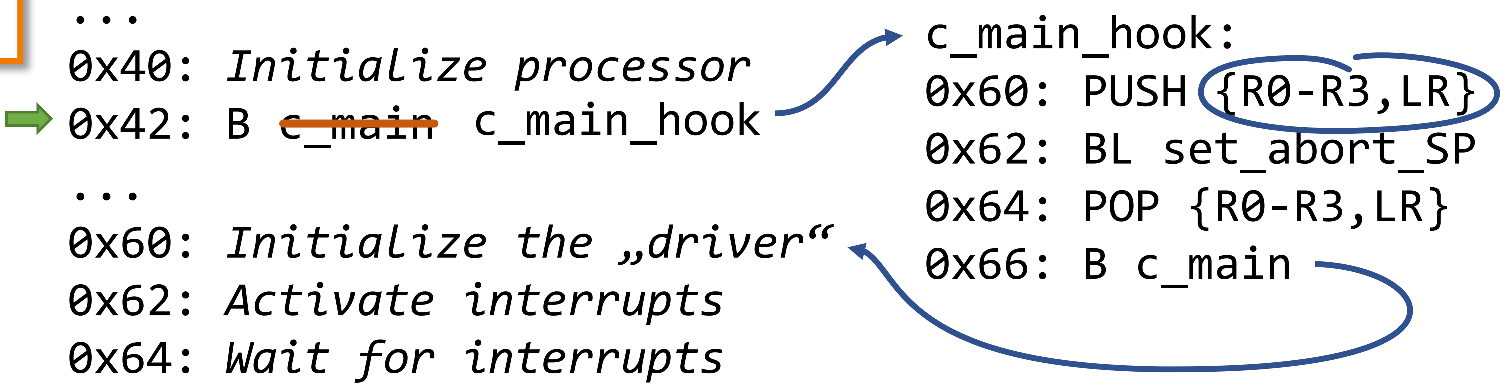
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

```
uint32 stack_abt[256] = { 0x54424153 };

void __attribute__((optimize("O0")))
set_abort_SP(void) {
    register uint32 sp_abt asm("r0") = (uint32) &stack_abt[255];

    dbg_change_processor_mode(DBG_PROCESSOR_MODE_ABT);
    asm("mov sp, %[value]" : : [value] "r" (sp_abt));
    dbg_change_processor_mode(DBG_PROCESSOR_MODE_SYS);
}
```

Change to Abort Mode

Set Stack Pointer to end of stack_abt array

Change to System Mode



```
...
0x40: Initialize processor
0x42: B ~~c_main~~ c_main_hook
...
```

c_main

```
0x60: Initialize the „driver"
0x62: Activate interrupts
0x64: Wait for interrupts
```

```
c_main_hook:
0x60: PUSH {R0-R3,LR}
0x62: BL set_abort_SP
0x64: POP {R0-R3,LR}
0x66: B c_main
```

- 🟧 ➡ Supervisor Mode
- 🟦 ➡ Abort Mode
- 🟩 ➡ System Mode

Initialize Abort Stack Pointer

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

```
uint32 stack_abt[256] = { 0x54424153 };

void __attribute__((optimize("O0")))
set_abort_SP(void) {
    register uint32 sp_abt asm("r0") = (uint32) &stack_abt[255];

    dbg_change_processor_mode(DBG_PROCESSOR_MODE_ABT);
    asm("mov sp, %[value]" : : [value] "r" (sp_abt));
    dbg_change_processor_mode(DBG_PROCESSOR_MODE_SYS);
}
```

Change to Abort Mode

Set Stack Pointer to end of stack_abt array

Change to System Mode

```
...
0x40: Initialize processor
0x42: B ~~c_main~~ c_main_hook
...
0x60: Initialize the „driver“
0x62: Activate interrupts
0x64: Wait for interrupts
```

c_main

```
c_main_hook:
0x60: PUSH {R0-R3,LR}
0x62: BL set_abort_SP
0x64: POP {R0-R3,LR}
0x66: B c_main
```

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Initialize Abort Stack Pointer

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

```
uint32 stack_abt[256] = { 0x54424153 };

void __attribute__((optimize("O0")))
set_abort_SP(void) {
    register uint32 sp_abt asm("r0") = (uint32) &stack_abt[255];

    dbg_change_processor_mode(DBG_PROCESSOR_MODE_ABT);
    asm("mov sp, %[value]" : : [value] "r" (sp_abt));
    dbg_change_processor_mode(DBG_PROCESSOR_MODE_SYS);
}
```

Change to Abort Mode

Set Stack Pointer to end of stack_abt array

Change to System Mode

...
0x40: *Initialize processor*
➡ 0x42: B ~~c_main~~ c_main_hook
...

c_main
0x60: *Initialize the „driver“*
0x62: *Activate interrupts*
0x64: *Wait for interrupts*

c_main_hook:
0x60: PUSH {R0-R3,LR}
0x62: BL set_abort_SP
0x64: POP {R0-R3,LR}
0x66: B c_main

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Initialize Abort Stack Pointer

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Analyzing Handle Exceptions Function

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

pref_abt_hdl

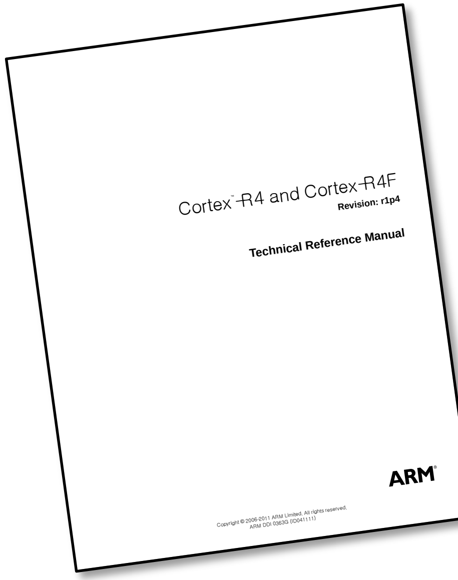
➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x50: MOV R0, #10
0x52: MOV R1, #3
0x54: ADD R0, R0, R1
0x56: B 0x50
...
0x80: MOV SP, LR
...                   #0x17
0x86: SRSDB SP!, ~~#0x1F~~
0x8C: ~~CPS #0x1F~~
...
➡ 0x90: B handle_exceptions
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x90
SP>R13_abt
LR>R14_abt
```



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➔ **Original Wi-Fi firmware always changes to System mode**

➔ `handle_exceptions` **handles all exceptions**

Analyzing Handle Exceptions Function

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

pref_abt_hdl

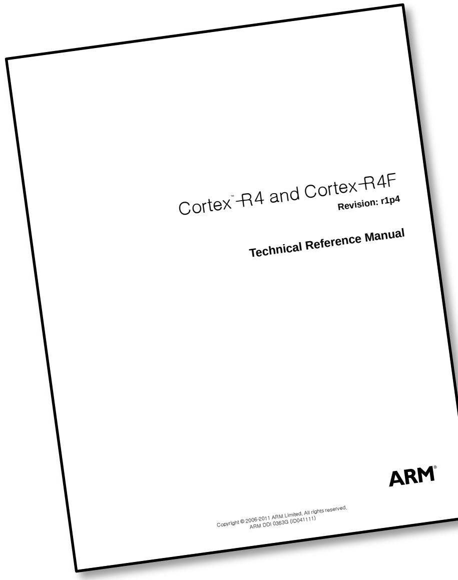
➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x50: MOV R0, #10
0x52: MOV R1, #3
0x54: ADD R0, R0, R1
0x56: B 0x50
...
0x80: MOV SP, LR
...
                           #0x17
0x86: SRSDB SP!, ~~#0x1F~~
0x8C: ~~CPS #0x1F~~
...
0x90: B handle_exceptions
```

Registers

```
R0 = 10
R1 = 3
...
PC = 0x90
SP>R13_abt
LR>R14_abt
```



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➔ Original Wi-Fi firmware always changes to System mode

➔ `handle_exceptions` handles all exceptions

Our Prefetch Abort Handler

Monitor Debug-Mode

Reset

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
```

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function
- Implement a breakpoint handler
- Activate breakpoints

⇨ Supervisor Mode
⇨ Abort Mode
⇨ System Mode

Our Prefetch Abort Handler

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function
- Implement a breakpoint handler
- Activate breakpoints

pref_abt_hdl

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...

...
0x80: MOV SP, LR
...
```

#0x17

0x86: SRSDB SP!, ~~#0x1F~~

0x8C: ~~CPS #0x1F~~

...

0x90: B handle_exceptions

Our Prefetch Abort Handler

Monitor Debug-Mode

Reset

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x80: SUB LR, LR, #4
0x82: SRSDB SP!, #0x17
0x84: PUSH {R0}
0x86: PUSH {LR}
0x88: SUB SP, SP, #24
0x8A: PUSH {R0-R7}
0x8C: MOV R0, #3
0x8E: B handle_exceptions
```

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Our Prefetch Abort Handler

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x80: SUB LR, LR, #4
0x82: SRSDB SP!, #0x17
0x84: PUSH {R0}
0x86: PUSH {LR}
0x88: SUB SP, SP, #24
0x8A: PUSH {R0-R7}
0x8C: MOV R0, #3
0x8E: B handle_exceptions
```

Our Prefetch Abort Handler

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x80: SUB LR, LR, #4
0x82: SRSDB SP!, #0x17
0x84: PUSH {R0}
0x86: PUSH {LR}
0x88: SUB SP, SP, #24
0x8A: PUSH {R0-R7}
0x8C: MOV R0, #3
0x8E: B handle_exceptions
```

LR = Breakpoint‘s PC Address + 4
LR (new) := Breakpoint‘s PC Address

Our Prefetch Abort Handler

Monitor Debug-Mode

Reset

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x80: SUB LR, LR, #4
0x82: SRSDB SP!, #0x17
0x84: PUSH {R0}
0x86: PUSH {LR}
0x88: SUB SP, SP, #24
0x8A: PUSH {R0-R7}
0x8C: MOV R0, #3
0x8E: B handle_exceptions
```

LR = Breakpoint‘s PC Address + 4

LR (new) := Breakpoint‘s PC Address

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode

➡ Abort Mode

➡ System Mode

Our Prefetch Abort Handler

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x80: SUB LR, LR, #4
0x82: SRSDB SP!, #0x17
0x84: PUSH {R0}
0x86: PUSH {LR}
0x88: SUB SP, SP, #24
0x8A: PUSH {R0-R7}
0x8C: MOV R0, #3
0x8E: B handle_exceptions
```

SRSDB Instruction

`SRSDB SP!, #0x17`

SRS	Store Return State (LR, SPSR) onto a Stack
DB	Decrement address before each transfer
!	Write final address back to SP of mode 0x17 (abort mode)

Our Prefetch Abort Handler

Monitor Debug-Mode

Reset

Example Program

Abort Stack

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x80: SUB LR, LR, #4
0x82: SRSDB SP!, #0x17
0x84: PUSH {R0}
0x86: PUSH {LR}
0x88: SUB SP, SP, #24
0x8A: PUSH {R0-R7}
0x8C: MOV R0, #3
0x8E: B handle_exceptions
```

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

SRSDB Instruction

`SRSDB SP!, #0x17`

SRS	Store Return State (LR, SPSR) onto a Stack
DB	Decrement address before each transfer
!	Write final address back to SP of mode 0x17 (abort mode)

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Our Prefetch Abort Handler

Monitor Debug-Mode

Reset

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x80: SUB LR, LR, #4
0x82: SRSDB SP!, #0x17
0x84: PUSH {R0}
0x86: PUSH {LR}
0x88: SUB SP, SP, #24
0x8A: PUSH {R0-R7}
0x8C: MOV R0, #3
0x8E: B handle_exceptions
```

Abort Stack

SPSR = CPSR SYS
LR = PC SYS

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

SRSDB Instruction

`SRSDB SP!, #0x17`

SRS	Store Return State (LR, SPSR) onto a Stack
DB	Decrement address before each transfer
!	Write final address back to SP of mode 0x17 (abort mode)

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Our Prefetch Abort Handler

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x80: SUB LR, LR, #4
0x82: SRSDB SP!, #0x17
0x84: PUSH {R0}
0x86: PUSH {LR}
0x88: SUB SP, SP, #24
0x8A: PUSH {R0-R7}
0x8C: MOV R0, #3
0x8E: B handle_exceptions
```

LR = Breakpoint‘s PC Address + 4

LR (new) := Breakpoint‘s PC Address

Abort Stack

SPSR = CPSR SYS

LR = PC SYS

➡ Supervisor Mode

➡ Abort Mode

➡ System Mode

Our Prefetch Abort Handler

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x80: SUB LR, LR, #4
0x82: SRSDB SP!, #0x17
0x84: PUSH {R0}
0x86: PUSH {LR}
0x88: SUB SP, SP, #24
0x8A: PUSH {R0-R7}
0x8C: MOV R0, #3
0x8E: B handle_exceptions
```

LR = Breakpoint‘s PC Address + 4
LR (new) := Breakpoint‘s PC Address

Abort Stack

SPSR = CPSR SYS
LR = PC SYS
R0

Our Prefetch Abort Handler

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x80: SUB LR, LR, #4
0x82: SRSDB SP!, #0x17
0x84: PUSH {R0}
0x86: PUSH {LR}
0x88: SUB SP, SP, #24
0x8A: PUSH {R0-R7}
0x8C: MOV R0, #3
0x8E: B handle_exceptions
```

LR = Breakpoint‘s PC Address + 4
LR (new) := Breakpoint‘s PC Address

Abort Stack

SPSR = CPSR SYS
LR = PC SYS
R0
LR ABT

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Our Prefetch Abort Handler

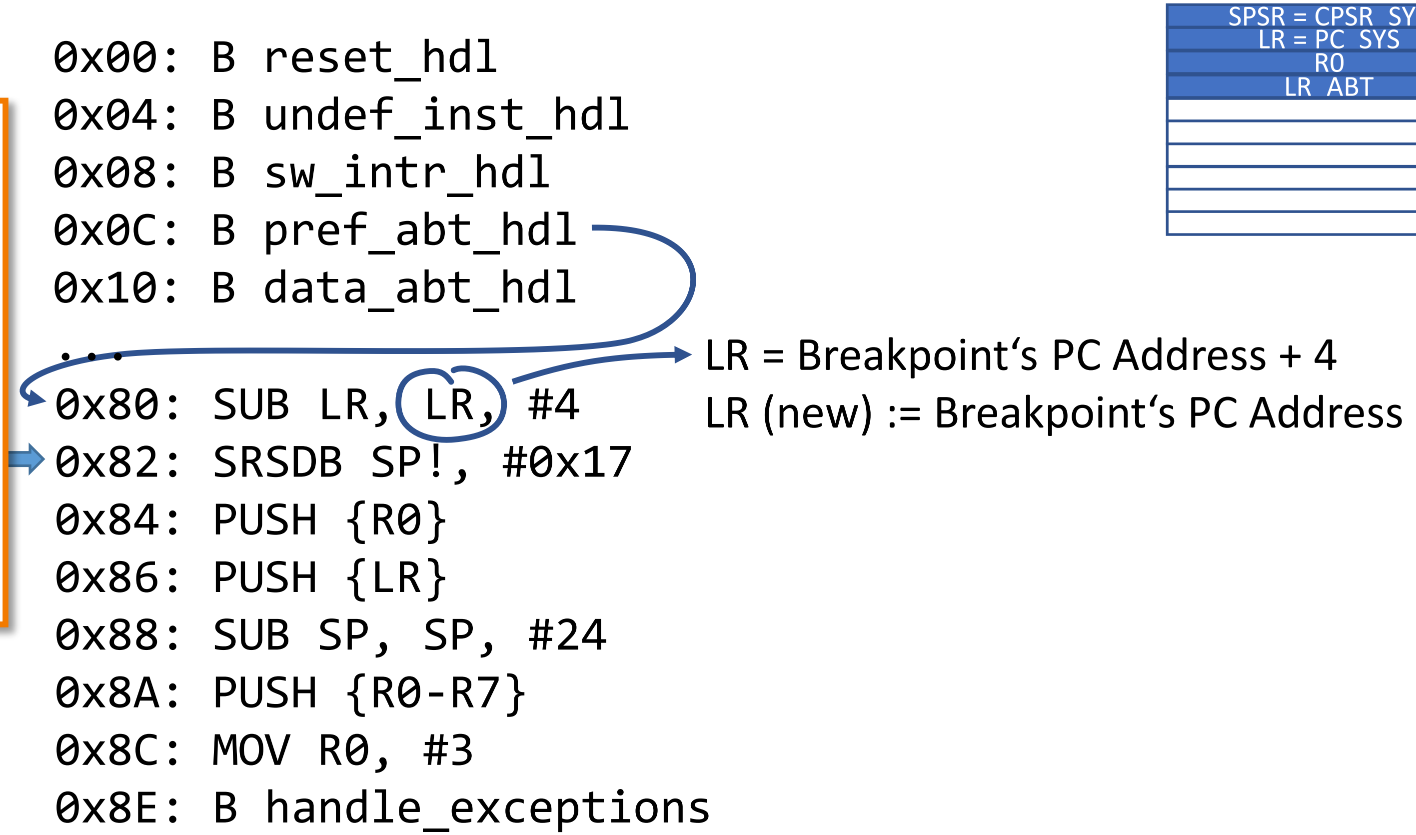
Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

Example Program



```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x80: SUB LR, LR, #4
0x82: SRSDB SP!, #0x17
0x84: PUSH {R0}
0x86: PUSH {LR}
0x88: SUB SP, SP, #24
0x8A: PUSH {R0-R7}
0x8C: MOV R0, #3
0x8E: B handle_exceptions
```

LR = Breakpoint‘s PC Address + 4

LR (new) := Breakpoint‘s PC Address

Abort Stack

SPSR = CPSR SYS
LR = PC SYS
R0
LR ABT

➡ Supervisor Mode

➡ Abort Mode

➡ System Mode

Our Prefetch Abort Handler

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

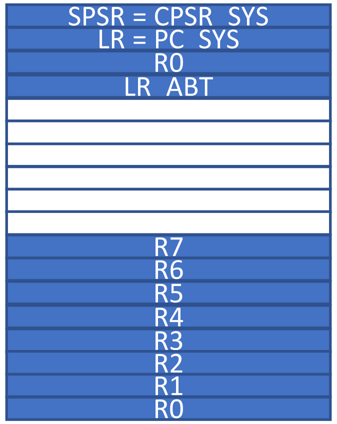
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x80: SUB LR, LR, #4
0x82: SRSDB SP!, #0x17
0x84: PUSH {R0}
0x86: PUSH {LR}
0x88: SUB SP, SP, #24
0x8A: PUSH {R0-R7}
0x8C: MOV R0, #3
0x8E: B handle_exceptions
```

Abort Stack



Our Prefetch Abort Handler

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

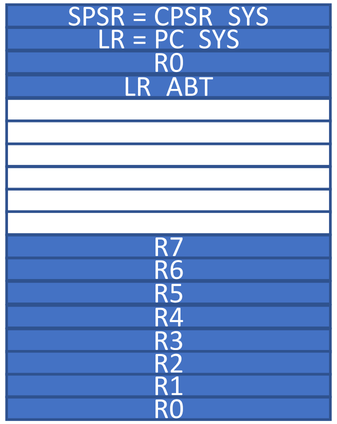
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

Example Program

```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x80: SUB LR, LR, #4
0x82: SRSDB SP!, #0x17
0x84: PUSH {R0}
0x86: PUSH {LR}
0x88: SUB SP, SP, #24
0x8A: PUSH {R0-R7}
0x8C: MOV R0, #3
0x8E: B handle_exceptions
```

R0 := Exception ID (3 → Prefetch Abort)

Abort Stack



SPSR = CPSR_SYS
LR = PC_SYS
R0
LR_ABT

R7
R6
R5
R4
R3
R2
R1
R0

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Our Prefetch Abort Handler

Monitor Debug-Mode

Reset

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

⇨ Supervisor Mode
⇨ Abort Mode
⇨ System Mode

Example Program

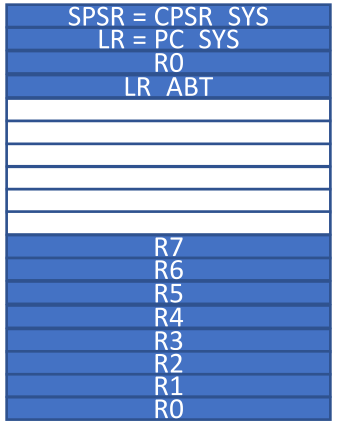
```
0x00: B reset_hdl
0x04: B undef_inst_hdl
0x08: B sw_intr_hdl
0x0C: B pref_abt_hdl
0x10: B data_abt_hdl
...
0x80: SUB LR, LR, #4
0x82: SRSDB SP!, #0x17
0x84: PUSH {R0}
0x86: PUSH {LR}
0x88: SUB SP, SP, #24
0x8A: PUSH {R0-R7}
0x8C: MOV R0, #3
0x8E: B handle_exceptions
```

R0 := Exception ID (3 → Prefetch Abort)

Registers

R0 = 3

Abort Stack



Analyzing handle_exceptions

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

⇨ Supervisor Mode
⇨ Abort Mode
⇨ System Mode

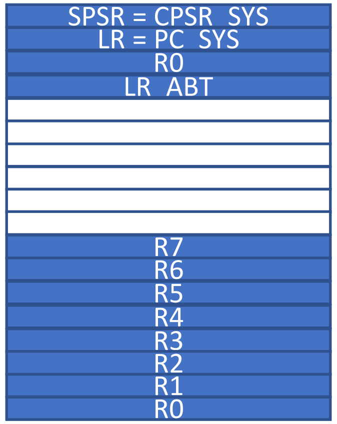
handle_exceptions

```
⇨ 0xA0: MOV R4, SP
  0xA2: ADD R4, R4, #64
  0xA4: LDMIA R4!, {R1,R3}
  0xA6: MRS R2, CPSR
  0xA8: PUSH {R0-R3}
  0xAA: SUB R4, R4, #12
  0xAC: STR R1, [R4]
  0xAE: AND R1, R3, #64
  0xB0: MOV R7, SP
  0xB2: ADD R7, R7, #88
  0xB4: MOV R6, R12
  0xB6: MOV R5, R11
  0xB8: MOV R4, R10
  0xBA: MOV R3, R9
  0xBC: MOV R2, R8
  0xBE: ADD SP, SP, #72
  0xC0: PUSH {R2-R7}
  0xC2: SUB SP, SP, #48
  0xC4: BL choose_exception_handler
```

Registers

R0 = 3

Abort Stack



SPSR = CPSR_SYS
LR = PC_SYS
R0
LR_ABT
R7
R6
R5
R4
R3
R2
R1
R0

Analyzing handle_exceptions

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

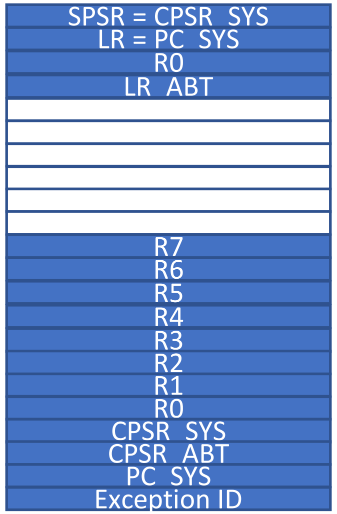
handle_exceptions

```
➡ 0xA0: MOV R4, SP
  0xA2: ADD R4, R4, #64
  0xA4: LDMIA R4!, {R1,R3}
  0xA6: MRS R2, CPSR
  0xA8: PUSH {R0-R3}
  0xAA: SUB R4, R4, #12
  0xAC: STR R1, [R4]
  0xAE: AND R1, R3, #64
  0xB0: MOV R7, SP
  0xB2: ADD R7, R7, #88
  0xB4: MOV R6, R12
  0xB6: MOV R5, R11
  0xB8: MOV R4, R10
  0xBA: MOV R3, R9
  0xBC: MOV R2, R8
  0xBE: ADD SP, SP, #72
  0xC0: PUSH {R2-R7}
  0xC2: SUB SP, SP, #48
  0xC4: BL choose_exception_handler
```

Registers

```
R0 = 3
```

Abort Stack



Analyzing handle_exceptions

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

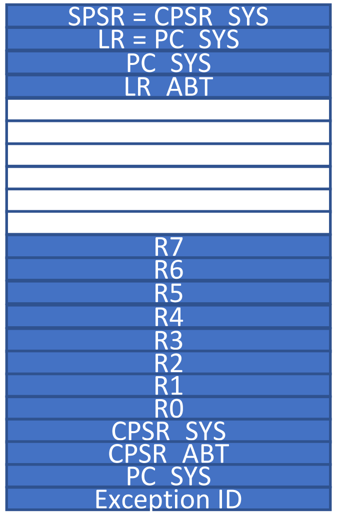
handle_exceptions

```
  0xA0: MOV R4, SP
  0xA2: ADD R4, R4, #64
  0xA4: LDMIA R4!, {R1,R3}
  0xA6: MRS R2, CPSR
  0xA8: PUSH {R0-R3}
➡ 0xAA: SUB R4, R4, #12
  0xAC: STR R1, [R4]
  0xAE: AND R1, R3, #64
  0xB0: MOV R7, SP
  0xB2: ADD R7, R7, #88
  0xB4: MOV R6, R12
  0xB6: MOV R5, R11
  0xB8: MOV R4, R10
  0xBA: MOV R3, R9
  0xBC: MOV R2, R8
  0xBE: ADD SP, SP, #72
  0xC0: PUSH {R2-R7}
  0xC2: SUB SP, SP, #48
  0xC4: BL choose_exception_handler
```

Registers

R0 = 3

Abort Stack



SPSR = CPSR_SYS
LR = PC_SYS
PC_SYS
LR_ABT
R7
R6
R5
R4
R3
R2
R1
R0
CPSR_SYS
CPSR_ABT
PC_SYS
Exception ID

Analyzing handle_exceptions

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

handle_exceptions

```
  0xA0: MOV R4, SP
  0xA2: ADD R4, R4, #64
  0xA4: LDMIA R4!, {R1,R3}
  0xA6: MRS R2, CPSR
  0xA8: PUSH {R0-R3}
➡ 0xAA: SUB R4, R4, #12
  0xAC: STR R1, [R4]
  0xAE: AND R1, R3, #64
  0xB0: MOV R7, SP
  0xB2: ADD R7, R7, #88
  0xB4: MOV R6, R12
  0xB6: MOV R5, R11
  0xB8: MOV R4, R10
  0xBA: MOV R3, R9
  0xBC: MOV R2, R8
  0xBE: ADD SP, SP, #72
  0xC0: PUSH {R2-R7}
  0xC2: SUB SP, SP, #48
  0xC4: BL choose_exception_handler
```

Registers

`R0 = 3`

Abort Stack

SPSR = CPSR_SYS
LR = PC_SYS
PC_SYS
LR_ABT
SP_ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR_SYS
CPSR_ABT
PC_SYS
Exception ID

Analyzing handle_exceptions

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

handle_exceptions

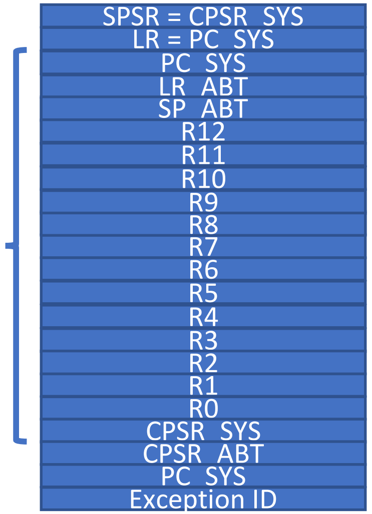
```
0xA0: MOV R4, SP
0xA2: ADD R4, R4, #64
0xA4: LDMIA R4!, {R1,R3}
0xA6: MRS R2, CPSR
0xA8: PUSH {R0-R3}
0xAA: SUB R4, R4, #12
0xAC: STR R1, [R4]
0xAE: AND R1, R3, #64
0xB0: MOV R7, SP
0xB2: ADD R7, R7, #88
0xB4: MOV R6, R12
0xB6: MOV R5, R11
0xB8: MOV R4, R10
0xBA: MOV R3, R9
0xBC: MOV R2, R8
0xBE: ADD SP, SP, #72
0xC0: PUSH {R2-R7}
0xC2: SUB SP, SP, #48
0xC4: BL choose_exception_handler
```

Registers

`R0 = 3`

Processor state when breakpoint was triggered

Abort Stack


SPSR = CPSR SYS
LR = PC SYS
PC SYS
LR ABT
SP ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR SYS
CPSR ABT
PC SYS
Exception ID

Analyzing handle_exceptions

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

handle_exceptions

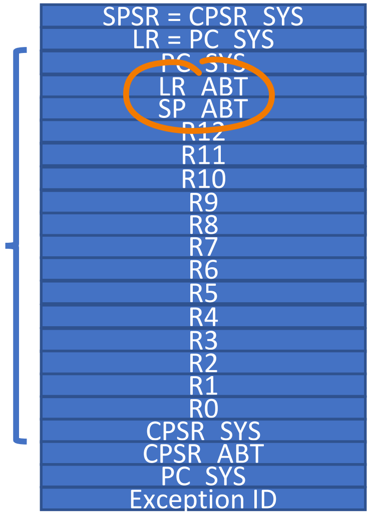
```
0xA0: MOV R4, SP
0xA2: ADD R4, R4, #64
0xA4: LDMIA R4!, {R1,R3}
0xA6: MRS R2, CPSR
0xA8: PUSH {R0-R3}
0xAA: SUB R4, R4, #12
0xAC: STR R1, [R4]
0xAE: AND R1, R3, #64
0xB0: MOV R7, SP
0xB2: ADD R7, R7, #88
0xB4: MOV R6, R12
0xB6: MOV R5, R11
0xB8: MOV R4, R10
0xBA: MOV R3, R9
0xBC: MOV R2, R8
0xBE: ADD SP, SP, #72
0xC0: PUSH {R2-R7}
0xC2: SUB SP, SP, #48
0xC4: BL choose_exception_handler
```

Registers

```
R0 = 3
```

Processor state when breakpoint was triggered

Abort Stack



SPSR = CPSR SYS
LR = PC SYS
PC SYS
LR ABT
SP ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR SYS
CPSR ABT
PC SYS
Exception ID

Analyzing handle_exceptions

Monitor
Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

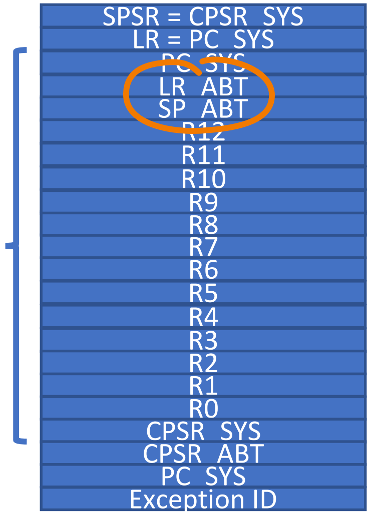
handle_exceptions

```
  0xA0: MOV R4, SP
  0xA2: ADD R4, R4, #64
  0xA4: LDMIA R4!, {R1,R3}
  0xA6: MRS R2, CPSR
  0xA8: PUSH {R0-R3}
➡ 0xAA: SUB R4, R4, #12
  0xAC: STR R1, [R4]
  0xAE: AND R1, R3, #64
  0xB0: MOV R7, SP
  0xB2: ADD R7, R7, #88
  0xB4: MOV R6, R12
  0xB6: MOV R5, R11
  0xB8: MOV R4, R10
  0xBA: MOV R3, R9
  0xBC: MOV R2, R8
  0xBE: ADD SP, SP, #72
  0xC0: PUSH {R2-R7}
  0xC2: SUB SP, SP, #48
  0xC4: BL choose_exception_handler
```

Registers

```
R0 = 3
```

Abort Stack



SPSR = CPSR_SYS
LR = PC_SYS
PC_SYS
LR_ABT
SP_ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR_SYS
CPSR_ABT
PC_SYS
Exception ID

Processor state when breakpoint was triggered

Analyzing handle_exceptions

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
    - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

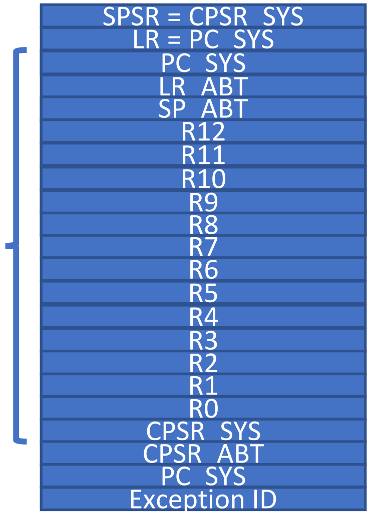
handle_exceptions

```
0xA0: MOV R4, SP
0xA2: ADD R4, R4, #64
0xA4: LDMIA R4!, {R1,R3}
0xA6: MRS R2, CPSR
0xA8: PUSH {R0-R3}
0xAA: SUB R4, R4, #12
0xAC: STR R1, [R4]
0xAE: AND R1, R3, #64
0xB0: MOV R7, SP
0xB2: ADD R7, R7, #88
0xB4: MOV R6, R12
0xB6: MOV R5, R11
0xB8: MOV R4, R10
0xBA: MOV R3, R9
0xBC: MOV R2, R8
0xBE: ADD SP, SP, #72
0xC0: PUSH {R2-R7}
0xC2: SUB SP, SP, #48
➡ 0xC4: BL choose_exception_handler
0xC6: CPSID IF
0xC8: ADD SP, SP, #48
0xCA: POP {R0-R6}
```

Registers

```
R0 = 3
```

Abort Stack



SPSR = CPSR_SYS
LR = PC_SYS
PC_SYS
LR_ABT
SP_ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR_SYS
CPSR_ABT
PC_SYS
Exception ID

Processor state when breakpoint was triggered

Analyzing handle_exceptions

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

⇨ Supervisor Mode
⇨ Abort Mode
⇨ System Mode

handle_exceptions

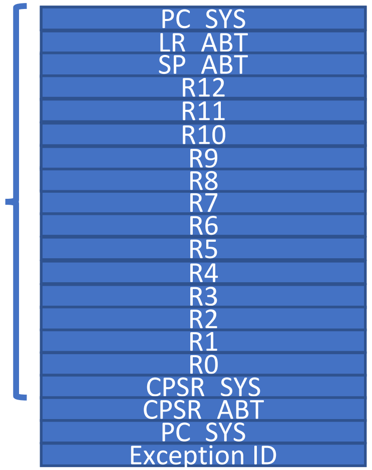
```
  0xB8: MOV R4, R10
  0xBA: MOV R3, R9
  0xBC: MOV R2, R8
  0xBE: ADD SP, SP, #72
  0xC0: PUSH {R2-R7}
  0xC2: SUB SP, SP, #48
⇨ 0xC4: BL choose_exception_handler
  0xC6: CPSID IF
  0xC8: ADD SP, SP, #48
  0xCA: POP {R0-R6}
  0xCC: MOV R8, R0
  0xCE: MOV R9, R1
  0xD0: MOV R10, R2
  0xD2: MOV R11, R3
  0xD4: MOV R12, R4
  0xD6: MOV LR, R6
  0xD8: SUB SP, SP, #60
  0xDA: POP {R0-R7}
  0xDC: ADD SP, SP, #32
  0xDE: RFEFD SP!
```

Restores saved registers

Registers

```
R0 = 3
```

Abort Stack



PC_SYS
LR_ABT
SP_ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR_SYS
CPSR_ABT
PC_SYS
Exception ID

Processor state when breakpoint was triggered

Analyzing handle_exceptions

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

handle_exceptions

```
0xB8: MOV R4, R10
0xBA: MOV R3, R9
0xBC: MOV R2, R8
0xBE: ADD SP, SP, #72
0xC0: PUSH {R2-R7}
0xC2: SUB SP, SP, #48
0xC4: BL choose_exception_handl
0xC6: CPSID IF
0xC8: ADD SP, SP, #48
0xCA: POP {R0-R6}
0xCC: MOV R8, R0
0xCE: MOV R9, R1
0xD0: MOV R10, R2
0xD2: MOV R11, R3
0xD4: MOV R12, R4
0xD6: MOV LR, R6
0xD8: SUB SP, SP, #60
0xDA: POP {R0-R7}
0xDC: ADD SP, SP, #32
0xDE: RFEFD SP!
```

Restores saved registers

Registers

`R0 = 3`

Pr

break

Abort Stack

RFEFD Instruction

`RFEFD SP!`

RFE	Return From Exception: writes PC and CPSR back
FD	Fulldescending stack
SP	Points at PC and CPSR
SP!	Write final address back to SP

Analyzing handle_exceptions

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode

➡ Abort Mode

➡ System Mode

handle_exceptions

```
0xB8: MOV R4, R10
0xBA: MOV R3, R9
0xBC: MOV R2, R8
0xBE: ADD SP, SP, #72
0xC0: PUSH {R2-R7}
0xC2: SUB SP, SP, #48
0xC4: BL choose_exception_hand
0xC6: CPSID IF
0xC8: ADD SP, SP, #48
0xCA: POP {R0-R6}
0xCC: MOV R8, R0
0xCE: MOV R9, R1
0xD0: MOV R10, R2
0xD2: MOV R11, R3
0xD4: MOV R12, R4
0xD6: MOV LR, R6
0xD8: SUB SP, SP, #60
0xDA: POP {R0-R7}
0xDC: ADD SP, SP, #32
0xDE: RFEFD SP!
```

Restores saved registers

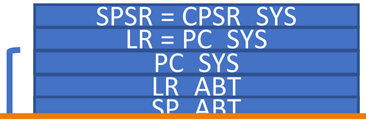
Registers

`R0 = 3`

Pr

break

Abort Stack



RFEFD Instruction

`RFEFD SP!`

RFE	Return From Exception: writes PC and CPSR back
FD	Fulldescending stack
SP	Points at PC and CPSR
SP!	Write final address back to SP

Analyzing handle_exceptions

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

🟧⇨ Supervisor Mode
🟦⇨ Abort Mode
🟩⇨ System Mode

handle_exceptions

```
0xB8: MOV R4, R10
0xBA: MOV R3, R9
0xBC: MOV R2, R8
0xBE: ADD SP, SP, #72
0xC0: PUSH {R2-R7}
0xC2: SUB SP, SP, #48
0xC4: BL choose_exception_handl
0xC6: CPSID IF
0xC8: ADD SP, SP, #48
0xCA: POP {R0-R6}
0xCC: MOV R8, R0
0xCE: MOV R9, R1
0xD0: MOV R10, R2
0xD2: MOV R11, R3
0xD4: MOV R12, R4
0xD6: MOV LR, R6
0xD8: SUB SP, SP, #60
0xDA: POP {R0-R7}
0xDC: ADD SP, SP, #32
0xDE: RFEFD SP!
```

Restores saved registers

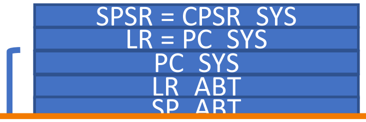
Registers

R0 = 3

Pr

break

Abort Stack



RFEFD Instruction

`RFEFD SP!`

RFE	Return From Exception: writes PC and CPSR back
FD	Fulldescending stack
SP	Points at PC and CPSR
SP!	Write final address back to SP

→ After handling an exception we can return to regular program execution

Analyzing handle_exceptions

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

handle_exceptions

```
 0xB8: MOV R4, R10
 0xBA: MOV R3, R9
 0xBC: MOV R2, R8
 0xBE: ADD SP, SP, #72
 0xC0: PUSH {R2-R7}
 0xC2: SUB SP, SP, #48
➡0xC4: BL choose_exception_handl
 0xC6: CPSID IF
 0xC8: ADD SP, SP, #48
 0xCA: POP {R0-R6}
 0xCC: MOV R8, R0
 0xCE: MOV R9, R1
 0xD0: MOV R10, R2
 0xD2: MOV R11, R3
 0xD4: MOV R12, R4
 0xD6: MOV LR, R6
 0xD8: SUB SP, SP, #60
 0xDA: POP {R0-R7}
 0xDC: ADD SP, SP, #32
 0xDE: RFEFD SP!
```

Restores saved registers

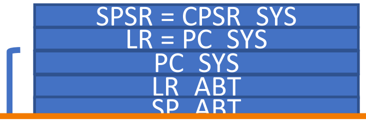
Registers

`R0 = 3`

Pr

break

Abort Stack



RFEFD Instruction

`RFEFD SP!`

RFE	Return From Exception: writes PC and CPSR back
FD	Fulldescending stack
SP	Points at PC and CPSR
SP!	Write final address back to SP

→ After handling an exception we can return to regular program execution

Analyzing handle_exceptions

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

🟧 ⇨ Supervisor Mode
🟦 ⇨ Abort Mode
🟩 ⇨ System Mode

handle_exceptions

```
0xB8: MOV R4, R10
0xBA: MOV R3, R9
0xBC: MOV R2, R8
0xBE: ADD SP, SP, #72
0xC0: PUSH {R2-R7}
0xC2: SUB SP, SP, #48
0xC4: BL choose_exception_handler
0xC6: CPSID IF
0xC8: ADD SP, SP, #48
0xCA: POP {R0-R6}
0xCC: MOV R8, R0
0xCE: MOV R9, R1
0xD0: MOV R10, R2
0xD2: MOV R11, R3
0xD4: MOV R12, R4
0xD6: MOV LR, R6
0xD8: SUB SP, SP, #60
0xDA: POP {R0-R7}
0xDC: ADD SP, SP, #32
0xDE: RFEFD SP!
```

Restores saved registers

Registers

R0 = 3

Processor state when breakpoint was triggered

Abort Stack

SPSR = CPSR_SYS
LR = PC_SYS
PC_SYS
LR_ABT
SP_ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR_SYS
CPSR_ABT
PC_SYS
Exception ID

➔ After handling an exception we can return to regular program execution

Analyzing handle_exceptions

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

choose_exception _handler

Registers

```
R0 = 3
```

```
0x0F4: CMP R0, #6
0x0F6: BNE 0xFE
0x0F8: CMP R1, #64
0x0FA: BEQ 0xFE
0x0FC: CPSIE F
0x0FE: MOV R0, SP
0x100: PUSH {LR}
0x102: POP {LR}
0x104: B handle_FIQ_or_trigger_trap
```

Abort Stack

SPSR = CPSR_SYS
LR = PC_SYS
PC_SYS
LR_ABT
SP_ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR_SYS
CPSR_ABT
PC_SYS
Exception ID

⇨ Supervisor Mode
⇨ Abort Mode
⇨ System Mode

Calling Prefetch Abort Handler

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

choose_exception _handler

```
0x0F4: CMP R0, #6
0x0F6: BNE 0xFE
0x0F8: CMP R1, #64
0x0FA: BEQ 0xFE
0x0FC: CPSIE F
0x0FE: MOV R0, SP
0x100: PUSH {LR}
0x102: POP {LR}
0x104: B handle_FIQ_or_trigger_trap
```

Registers

```
R0 = 3
```

Abort Stack

SPSR = CPSR_SYS
LR = PC_SYS
PC_SYS
LR_ABT
SP_ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR_SYS
CPSR_ABT
PC_SYS
Exception ID

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Calling Prefetch Abort Handler

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

choose_exception _handler

```
0x0F4: CMP R0, #6
0x0F6: BNE 0xFE
0x0F8: CMP R1, #64
0x0FA: BEQ 0xFE
0x0FC: CPSIE F
0x0FE: MOV R0, SP
0x100: PUSH {LR}
0x102: POP {LR}
0x104: B handle_FIQ_or_trigger_trap
```

Check whether Exception ID (R0) equals 6 (fast interrupt/FIQ)

Registers

`R0 = 3`

Abort Stack

SPSR = CPSR SYS
LR = PC SYS
PC SYS
LR ABT
SP ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR SYS
CPSR ABT
PC SYS
Exception ID

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Calling Prefetch Abort Handler

Monitor Debug-Mode

choose_exception _handler

Registers

Abort Stack

R0 = 3

SPSR = CPSR_SYS
LR = PC_SYS
PC_SYS
LR_ABT
SP_ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR_SYS
CPSR_ABT
PC_SYS
Exception ID

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
    - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

```
0x0F4: CMP R0, #6
0x0F6: BNE 0xFE
0x0F8: CMP R1, #64
0x0FA: BEQ 0xFE
0x0FC: CPSIE F
0x0FE: MOV R0, SP
0x100: PUSH {LR}
0x102: POP {LR}
0x104: B handle_FIQ_or_trigger_trap
```

Check whether Exception ID (R0) equals 6 (fast interrupt/FIQ)

Handles FIQ or prints debug information and stops execution

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Calling Prefetch Abort Handler

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

choose_exception _handler

```
0x0F4: CMP R0, #6
0x0F6: BNE 0xFE
0x0F8: CMP R1, #64
0x0FA: BEQ 0xFE
0x0FC: CPSIE F
0x0FE: MOV R0, SP
0x100: PUSH {LR}
0x102: POP {LR}
0x104: B handle_FIQ_or_trigger_trap
```

Check whether Exception ID (R0) equals 6 (fast interrupt/FIQ)

Handles FIQ or prints debug information and stops execution

Registers

`R0 = 3`

Abort Stack

SPSR = CPSR_SYS
LR = PC_SYS
PC_SYS
LR_ABT
SP_ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR_SYS
CPSR_ABT
PC_SYS
Exception ID

→ We want to handle our breakpoint instead of triggering the trap

Calling Prefetch Abort Handler

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

choose_exception _handler

```
0x0F4: CMP R0, #6
0x0F6: BNE 0xFE
0x0F8: CMP R1, #64
0x0FA: BEQ 0xFE
0x0FC: CPSIE F
0x0FE: MOV R0, SP
0x100: PUSH {LR}
0x102: POP {LR}
0x104: B handle_FIQ_or_trigger_trap
```

Registers

`R0 = 3`

Abort Stack

SPSR = CPSR_SYS
LR = PC_SYS
PC_SYS
LR_ABT
SP_ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR_SYS
CPSR_ABT
PC_SYS
Exception ID

→ We want to handle our breakpoint instead of triggering the trap

Calling Prefetch Abort Handler

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

choose_exception _handler

```
0x0F0: CMP R0, #3
0x0F2: BEQ pref_abort
0x0F4: CMP R0, #6
0x0F6: BNE 0xFE
0x0F8: CMP R1, #64
0x0FA: BEQ 0xFE
0x0FC: CPSIE F
0x0FE: MOV R0, SP
0x100: PUSH {LR}
0x102: POP {LR}
0x104: B handle_FIQ_or_trigger_trap
```

Registers

```
R0 = 3
```

Abort Stack

SPSR = CPSR SYS
LR = PC SYS
PC SYS
LR ABT
SP ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR SYS
CPSR ABT
PC SYS
Exception ID

→ We want to handle our breakpoint instead of triggering the trap

Calling Prefetch Abort Handler

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

choose_exception _handler

```
0x0F0: CMP R0, #3
0x0F2: BEQ pref_abort
0x0F4: CMP R0, #6
0x0F6: BNE 0xFE
0x0F8: CMP R1, #64
0x0FA: BEQ 0xFE
0x0FC: CPSIE F
0x0FE: MOV R0, SP
0x100: PUSH {LR}
0x102: POP {LR}
0x104: B handle_FIQ_or_trigger_trap
pref_abort:
0x106: MOV R0, SP
0x108: PUSH {LR}
0x10A: POP {LR}
0x10C: B handle_pref_abort_exception
```

Registers

```
R0 = 3
```

Abort Stack

SPSR = CPSR SYS
LR = PC SYS
PC SYS
LR ABT
SP ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR SYS
CPSR ABT
PC SYS
Exception ID

→ **We want to handle our breakpoint instead of triggering the trap**

Calling Prefetch Abort Handler

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

choose_exception _handler

```
0x0F0: CMP R0, #3
0x0F2: BEQ pref_abort
0x0F4: CMP R0, #6
0x0F6: BNE 0xFE
0x0F8: CMP R1, #64
0x0FA: BEQ 0xFE
0x0FC: CPSIE F
0x0FE: MOV R0, SP
0x100: PUSH {LR}
0x102: POP {LR}
0x104: B handle_FIQ_or_trigger_trap
pref_abort:
0x106: MOV R0, SP
0x108: PUSH {LR}
0x10A: POP {LR}
0x10C: B handle_pref_abort_exception
```

Registers

R0 = 3

Abort Stack

SPSR = CPSR_SYS
LR = PC_SYS
PC_SYS
LR_ABT
SP_ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR_SYS
CPSR_ABT
PC_SYS
Exception ID

→ We want to handle our breakpoint instead of triggering the trap

Calling Prefetch Abort Handler

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

choose_exception _handler

```
0x0F0: CMP R0, #3
0x0F2: BEQ pref_abort
0x0F4: CMP R0, #6
0x0F6: BNE 0xFE
0x0F8: CMP R1, #64
0x0FA: BEQ 0xFE
0x0FC: CPSIE F
0x0FE: MOV R0, SP
0x100: PUSH {LR}
0x102: POP {LR}
0x104: B handle_FIQ_or_trigger_trap
pref_abort:
0x106: MOV R0, SP
0x108: PUSH {LR}
0x10A: POP {LR}
0x10C: B handle_pref_abort_exception
```

Registers

`R0 = 3`

Abort Stack

SPSR = CPSR SYS
LR = PC SYS
PC SYS
LR ABT
SP ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR SYS
CPSR ABT
PC SYS
Exception ID

→ We want to handle our breakpoint instead of triggering the trap

Calling Prefetch Abort Handler

Monitor
Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

choose_exception _handler

```
0x0F0: CMP R0, #3
0x0F2: BEQ pref_abort
0x0F4: CMP R0, #6
0x0F6: BNE 0xFE
0x0F8: CMP R1, #64
0x0FA: BEQ 0xFE
0x0FC: CPSIE F
0x0FE: MOV R0, SP
0x100: PUSH {LR}
0x102: POP {LR}
0x104: B handle_FIQ_or_trigger_trap
pref_abort:
0x106: MOV R0, SP
0x108: PUSH {LR}
0x10A: POP {LR}
0x10C: B handle_pref_abort_exception
```

Registers

`R0 = 3`

Abort Stack

SPSR = CPSR SYS
LR = PC SYS
PC SYS
LR ABT
SP ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR SYS
CPSR ABT
PC SYS
Exception ID

→ We want to handle our breakpoint instead of triggering the trap

Calling Prefetch Abort Handler

Monitor
Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

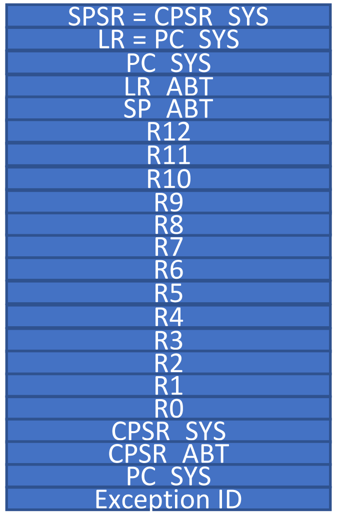
choose_exception _handler

```
0x0F0: CMP R0, #3
0x0F2: BEQ pref_abort
0x0F4: CMP R0, #6
0x0F6: BNE 0xFE
0x0F8: CMP R1, #64
0x0FA: BEQ 0xFE
0x0FC: CPSIE F
0x0FE: MOV R0, SP
0x100: PUSH {LR}
0x102: POP {LR}
0x104: B handle_FIQ_or_trigger_trap
pref_abort:
0x106: MOV R0, SP
0x108: PUSH {LR}
0x10A: POP {LR}
0x10C: B handle_pref_abort_exception
```

Registers

`R0 = 3`

Abort Stack



SPSR = CPSR_SYS
LR = PC_SYS
PC_SYS
LR_ABT
SP_ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR_SYS
CPSR_ABT
PC_SYS
Exception ID

→ We want to handle our breakpoint instead of triggering the trap

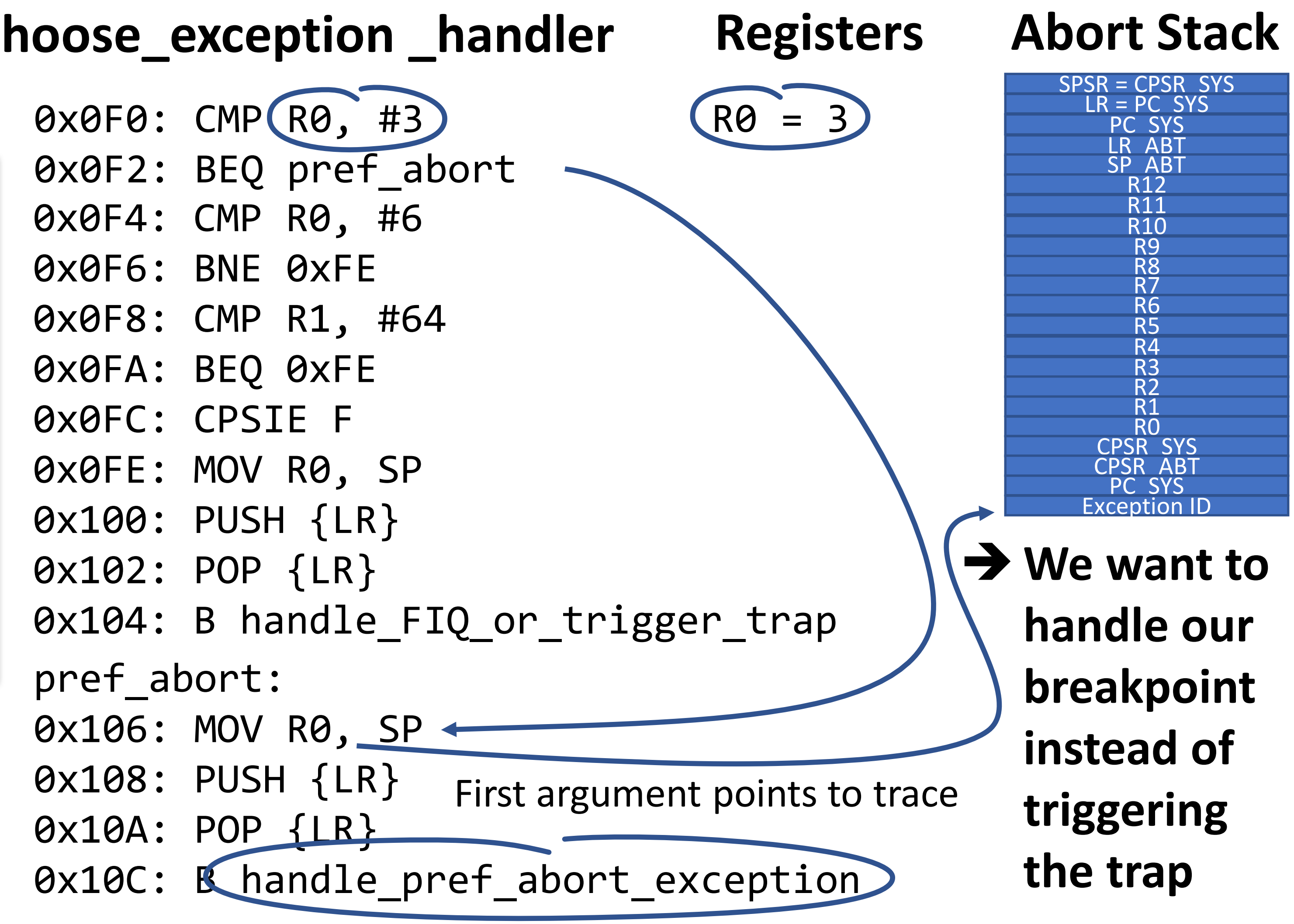
Calling Prefetch Abort Handler

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode



choose_exception _handler

```
0x0F0: CMP R0, #3
0x0F2: BEQ pref_abort
0x0F4: CMP R0, #6
0x0F6: BNE 0xFE
0x0F8: CMP R1, #64
0x0FA: BEQ 0xFE
0x0FC: CPSIE F
0x0FE: MOV R0, SP
0x100: PUSH {LR}
0x102: POP {LR}
0x104: B handle_FIQ_or_trigger_trap
pref_abort:
0x106: MOV R0, SP
0x108: PUSH {LR}
0x10A: POP {LR}
0x10C: B handle_pref_abort_exception
```

First argument points to trace

Registers

R0 = 3

Abort Stack

➔ We want to handle our breakpoint instead of triggering the trap

Fixing LR and SP in Trace

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);
    ...
}

void
fix_sp_lr(struct trace *trace)
{
    register unsigned int sp_sys asm("r1");
    register unsigned int lr_sys asm("r2");

    dbg_disable_monitor_mode_debugging();
    dbg_change_processor_mode(DBG_PROCESSOR_MODE_SYS);
    asm("mov %[result], sp" : [result] "=r" (sp_sys));
    asm("mov %[result], lr" : [result] "=r" (lr_sys));
    dbg_change_processor_mode(DBG_PROCESSOR_MODE_ABT);
    dbg_enable_monitor_mode_debugging();

    trace->lr = lr_sys;
    trace->sp = sp_sys;
}
```

Trace

SPSR = CPSR SYS
LR = PC SYS
PC SYS
LR ABT
SP ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR SYS
CPSR ABT
PC SYS
Exception ID

Fixing LR and SP in Trace

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);
    ...
}

void
fix_sp_lr(struct trace *trace)
{
    register unsigned int sp_sys asm("r1");
    register unsigned int lr_sys asm("r2");

    dbg_disable_monitor_mode_debugging();
    dbg_change_processor_mode(DBG_PROCESSOR_MODE_SYS);
    asm("mov %[result], sp" : [result] "=r" (sp_sys));
    asm("mov %[result], lr" : [result] "=r" (lr_sys));
    dbg_change_processor_mode(DBG_PROCESSOR_MODE_ABT);
    dbg_enable_monitor_mode_debugging();

    trace->lr = lr_sys;
    trace->sp = sp_sys;
}
```

Disable debugger and enter System Mode

Trace

SPSR = CPSR_SYS
LR = PC_SYS
PC_SYS
LR_ABT
SP_ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR_SYS
CPSR_ABT
PC_SYS
Exception ID

Fixing LR and SP in Trace

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

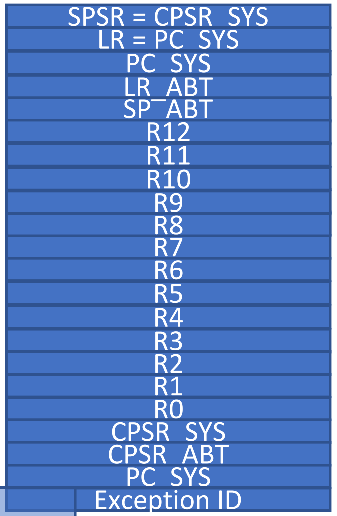
➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);
    ...
}

void
fix_sp_lr(struct trace *trace)
{
    register unsigned int sp_sys asm("r1");
    register unsigned int lr_sys asm("r2");

    dbg_disable_monitor_mode_debugging();
    dbg_change_processor_mode(DBG_PROCESSOR_MODE_SYS);
    asm("mov %[result], sp" : [result] "=r" (sp_sys));
    asm("mov %[result], lr" : [result] "=r" (lr_sys));
    dbg_change_processor_mode(DBG_PROCESSOR_MODE_ABT);
    dbg_enable_monitor_mode_debugging();

    trace->lr = lr_sys;
    trace->sp = sp_sys;
}
```

Disable debugger and enter System Mode

Store SP_SYS and LR_SYS in R1 and R2

Trace



Fixing LR and SP in Trace

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);
    ...
}

void
fix_sp_lr(struct trace *trace)
{
    register unsigned int sp_sys asm("r1");
    register unsigned int lr_sys asm("r2");

    dbg_disable_monitor_mode_debugging();
    dbg_change_processor_mode(DBG_PROCESSOR_MODE_SYS);
    asm("mov %[result], sp" : [result] "=r" (sp_sys));
    asm("mov %[result], lr" : [result] "=r" (lr_sys));
    dbg_change_processor_mode(DBG_PROCESSOR_MODE_ABT);
    dbg_enable_monitor_mode_debugging();

    trace->lr = lr_sys;
    trace->sp = sp_sys;
}
```

Disable debugger and enter System Mode

Store SP_SYS and LR_SYS in R1 and R2

Return to Abort Mode and re-enable debugger

Trace

SPSR = CPSR_SYS
LR = PC_SYS
PC_SYS
LR_ABT
SP_ABT
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR_SYS
CPSR_ABT
PC_SYS
Exception ID

Fixing LR and SP in Trace

Monitor Debug-Mode

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);
    ...
}

void
fix_sp_lr(struct trace *trace)
{
    register unsigned int sp_sys asm("r1");
    register unsigned int lr_sys asm("r2");

    dbg_disable_monitor_mode_debugging();
    dbg_change_processor_mode(DBG_PROCESSOR_MODE_SYS);
    asm("mov %[result], sp" : [result] "=r" (sp_sys));
    asm("mov %[result], lr" : [result] "=r" (lr_sys));
    dbg_change_processor_mode(DBG_PROCESSOR_MODE_ABT);
    dbg_enable_monitor_mode_debugging();

    trace->lr = lr_sys;
    trace->sp = sp_sys;
}
```

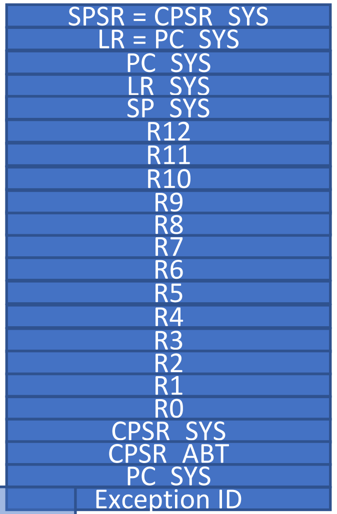
Disable debugger and enter System Mode

Store SP_SYS and LR_SYS in R1 and R2

Return to Abort Mode and re-enable debugger

Update Trace

Trace


SPSR = CPSR SYS
LR = PC SYS
PC SYS
LR SYS
SP SYS
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
CPSR SYS
CPSR ABT
PC SYS
Exception ID

Handling Breakpoints

Monitor
Debug-Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);
    ...
}
```

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Handling Breakpoints

Simplest Implementation
Breakpoint triggers only once

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address
               (0, trace->pc)) {

            // Handle Breakpoint:
            //  - Print information
            //  - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }



    ...
}
```

```
  start:
➡ MOV R0, #1
● MOV R0, #3
  B start
```

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Handling Breakpoints

Simplest Implementation
Breakpoint triggers only once

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address
                (0, trace->pc)) {

            // Handle Breakpoint:
            //   - Print information
            //   - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }



    ...
}
```

```
  start:
  MOV R0, #1
● MOV R0, #3
  B start
```

→ Supervisor Mode
→ Abort Mode
→ System Mode

Handling Breakpoints

Simplest Implementation
Breakpoint triggers only once

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address
                (0, trace->pc)) {

            // Handle Breakpoint:
            //   - Print information
            //   - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

```
start:
MOV R0, #1
● MOV R0, #3
B start
```

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Handling Breakpoints

Simplest Implementation
Breakpoint triggers only once

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
⇨     if(dbg_triggers_on_breakpoint_address
                (0, trace->pc)) {

            // Handle Breakpoint:
            //   - Print information
            //   - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

```
 start:
 MOV R0, #1
●MOV R0, #3
 B start
```

⇨ Supervisor Mode
⇨ Abort Mode
⇨ System Mode

Handling Breakpoints

Simplest Implementation
Breakpoint triggers only once

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_triggers_on_breakpoint_address
                (0, trace->pc)) {
            // Handle Breakpoint:
            //  - Print information
            //  - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

```
start:
MOV R0, #1
MOV R0, #3
B start
```

Handling Breakpoints

Simplest Implementation
Breakpoint triggers only once

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_triggers_on_breakpoint_address
                (0, trace->pc)) {
            // Handle Breakpoint:
            //  - Print information
            //  - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

Disable the breakpoint

```
start:
MOV R0, #1
MOV R0, #3
B start
```

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Handling Breakpoints

Simplest Implementation
Breakpoint triggers only once

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_triggers_on_breakpoint_address
               (0, trace->pc)) {

            // Handle Breakpoint:
            //  - Print information
            //  - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }

...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

Disable the breakpoint

Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

- Supervisor Mode
- Abort Mode
- System Mode

Handling Breakpoints

Simplest Implementation
Breakpoint triggers only once

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler
- Activate breakpoints

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_[illegible]address
            (0, trace->pc)) {

            // Handle Breakpoint:
            // - [illegible]
            // - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

Disable the breakpoint

Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

➡ (at MOV R0, #3)

Handling Breakpoints

Simplest Implementation
Breakpoint triggers only once

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_triggers_on_breakpoint_address
            (0, trace->pc)) {
            // Handle Breakpoint:
            // - Print information
            // - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

Disable the breakpoint

Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

➡ Supervisor Mode
➡ Abort Mode
➡ System Mode

Handling Breakpoints

Simplest Implementation
Breakpoint triggers only once

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
    - Handle and reset breakpoints
    - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_triggers_on_breakpoint_address
                (0, trace->pc)) {

            // Handle Breakpoint:
            //  - Print information
            //  - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

Disable the breakpoint

Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

➡ B start

Handling Breakpoints

Simplest Implementation
Breakpoint triggers only once

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_triggers_on_breakpoint_address
                (0, trace->pc)) {
            // Handle Breakpoint:
            //  - Print information
            //  - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }



    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

Disable the breakpoint

Continue firmware execution

```
  start:
➡ MOV R0, #1
● MOV R0, #3
  B start
```

➡ Abort Mode
➡ System Mode

Handling Breakpoints

Simplest Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_triggers_on_breakpoint_address
                (0, trace->pc)) {

            // Handle Breakpoint:
            //  - Print information
            //  - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

~~Disable the breakpoint~~

Continue firmware execution

```
  start:
➡ MOV R0, #1
● MOV R0, #3
  B start
```

Handling Breakpoints

Simplest Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_triggers_on_breakpoint_address
            (0, trace->pc)) {

            // Handle Breakpoint:
            //  - Print information
            //  - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

~~Disable the breakpoint~~

Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

➡ Abort Mode
➡ System Mode

Handling Breakpoints

Simplest Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_[illegible]
            (0, trace->pc)) {

            // Handle Breakpoint:
            //  - [illegible]
            //  - Change register values in trace
```

~~dbg_disable_breakpoint(0);~~

```
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

~~Disable the breakpoint~~

Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

➡ Abort Mode
➡ System Mode

Handling Breakpoints

Simplest Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
    - Handle and reset breakpoints
    - Perform Single-Stepping
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_triggers_on_breakpoint_address
                (0, trace->pc)) {
            // Handle Breakpoint:
            //  - Print information
            //  - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

~~Disable the breakpoint~~

Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

➡ Abort Mode
➡ System Mode

Handling Breakpoints

Simplest Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
    - Handle and reset breakpoints
    - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_triggers_on_breakpoint_address
           (0, trace->pc)) {

            // Handle Breakpoint:
            //   - Print information
            //   - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

~~**Disable the breakpoint**~~

Continue firmware execution

```
start:
MOV R0, #1
● MOV R0, #3
B start
```

Handling Breakpoints

Simplest Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_triggers_on_breakpoint_address
            (0, trace->pc)) {

            // Handle Breakpoint:
            //   - Print information
            //   - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }
➡   ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

~~Disable the breakpoint~~

Continue firmware execution

```
start:
MOV R0, #1
● MOV R0, #3
B start
```

Handling Breakpoints

Simplest Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
    - Handle and reset breakpoints
    - Perform Single-Stepping
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_triggers_on_breakpoint_address
                (0, trace->pc)) {

            // Handle Breakpoint:
            //   - Print information
            //   - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }



    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

~~Disable the breakpoint~~

Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

➡ Abort Mode
➡ System Mode

Handling Breakpoints

Simplest Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

```
            // Handle Breakpoint:
            //   - Change register values in trace
```

Write code to handle the breakpoint

~~**Disable the breakpoint**~~

```
        }
    }
```

...	**Continue firmware execution**

```
}
```

```
start:
MOV R0, #1
MOV R0, #3
B start
```

➡ Abort Mode
➡ System Mode

Handling Breakpoints

Simplest Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
    - Handle and reset breakpoints
    - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_is_breakpoint_address_equals
            (0, trace->pc)) {

            // Handle Breakpoint:
            //   - Print information
            //   - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

~~Disable the breakpoint~~

Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

Handling Breakpoints

Simplest Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_triggers_on_breakpoint_address
                (0, trace->pc)) {

            // Handle Breakpoint:
            //   - Print information
            //   - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

~~Disable the breakpoint~~

Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

Handling Breakpoints

Simplest Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_triggers_on_breakpoint_address
            (0, trace->pc)) {

            // Handle Breakpoint:
            //  - Print information
            //  - Change register values in trace

            dbg_disable_breakpoint(0);
        }
    }
➡   ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

~~Disable the breakpoint~~

Continue firmware execution

```
start:
MOV R0, #1
● MOV R0, #3
B start
```

Handling Breakpoints

Simplest Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_
            (0, trace->pc)) {
            // Handle Breakpoint:
            //
            //   - Change register values in trace

            dbg_
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

~~Disable the breakpoint~~

Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

Handling Breakpoints

Simplest Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if(dbg_is_breakpoint_enabled(0)) {
        if(dbg_triggers_on_breakpoint_address
                (0, trace->pc)) {
            // Handle Breakpoint:
            //   - Print information
            //   - Change register values in trace

            ~~dbg_disable_breakpoint(0);~~
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals the program counter of the system mode

Write code to handle the breakpoint

~~Disable the breakpoint~~

Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

→ **We are in an endless debugging loop**

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
    - Handle and reset breakpoints
    - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address (0, trace->pc)) {

            // Handle Breakpoint:

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_type_to_instr_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {
            dbg_set_breakpoint_type_to_instr_addr_match(0);
            breakpoint_hit &= ~DBGBP0;
        }
    }


    ...
}
```

```
  start:
  MOV R0, #1
● MOV R0, #3
  B start
```

→ We are in an endless debugging loop

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
➡   if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address (0, trace->pc)) {

            // Handle Breakpoint:

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_type_to_instr_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {
            dbg_set_breakpoint_type_to_instr_addr_match(0);
            breakpoint_hit &= ~DBGBP0;
        }
    }

    ...
}
```

Check whether breakpoint is enabled

```
  start:
  MOV R0, #1
● MOV R0, #3
  B start
```

→ We are in an endless debugging loop

➡ Abort Mode
➡ System Mode

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
    - Handle and reset breakpoints
    - Perform Single-Stepping
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint:

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_type_to_instr_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {
            dbg_set_breakpoint_type_to_instr_addr_match(0);
            breakpoint_hit &= ~DBGBP0;
        }
    }


    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

```
start:
MOV R0, #1
MOV R0, #3
B start
```

→ We are in an endless debugging loop

➡ Abort Mode
➡ System Mode

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_type_to_instr_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {
            dbg_set_breakpoint_type_to_instr_addr_match(0);
            breakpoint_hit &= ~DBGBP0;
        }
    }


    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the breakpoint

```
start:
MOV R0, #1
● MOV R0, #3
B start
```

→ We are in an endless debugging loop

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_type_to_instr_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {
            dbg_set_breakpoint_type_to_instr_addr_match(0);
            breakpoint_hit &= ~DBGBP0;
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the breakpoint

Remember which breakpoint was hit

```
start:
MOV R0, #1
MOV R0, #3
B start
```

→ We are in an endless debugging loop

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
    - Handle and reset breakpoints
    - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle breakpoint

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_type_to_instr_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {
            dbg_set_breakpoint_type_to_instr_addr_match(0);
            breakpoint_hit &= ~DBGBP0;
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

```
  start:
  MOV R0, #1
● MOV R0, #3
  B start
```

→ We are in an endless debugging loop

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_type_to_instr_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {
            dbg_set_breakpoint_type_to_instr_addr_match(0);
            breakpoint_hit &= ~DBGBP0;
        }
    }

➡   ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Continue firmware execution

```
  start:
  MOV R0, #1
● MOV R0, #3
  B start
```

→ We are in an endless debugging loop

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

```
        } else if (breakpoint_hit & DBGBP0) {
            dbg_set_breakpoint_type_to_instr_addr_match(0);
            breakpoint_hit &= ~DBGBP0;
        }
    }
```

Continue firmware execution

```
    ...
}
```

```
start:
MOV R0, #1
MOV R0, #3
B start
```

→ We are in an endless debugging loop

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

```
        } else if (breakpoint_hit & DBGBP0) {
            dbg_set_breakpoint_type_to_instr_addr_match(0);
            breakpoint_hit &= ~DBGBP0;
        }
    }
```

`...` **Continue firmware execution**

```
}
```

```
  start:
  MOV R0, #1
● MOV R0, #3
➡ B start
```

→ **We are in an endless debugging loop**

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle breakpoint

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_type_to_instr_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {
            dbg_set_breakpoint_type_to_instr_addr_match(0);
            breakpoint_hit &= ~DBGBP0;
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Continue firmware execution

```
start:
MOV R0, #1
● MOV R0, #3
B start
```

→ We are in an endless debugging loop

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_type_to_instr_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {
            dbg_set_breakpoint_type_to_instr_addr_match(0);
            breakpoint_hit &= ~DBGBP0;
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Continue firmware execution

```
  start:
  MOV R0, #1
● MOV R0, #3
  B start
```

→ We are in an endless debugging loop

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_type_to_instr_addr_mismatch(0);
➡       } else if (breakpoint_hit & DBGBP0) {
            dbg_set_breakpoint_type_to_instr_addr_match(0);
            breakpoint_hit &= ~DBGBP0;
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

Continue firmware execution

```
  start:
  MOV R0, #1
● MOV R0, #3
  B start
```

→ We are in an endless debugging loop

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_to_trigger_on_address_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {
            dbg_set_breakpoint_to_trigger_on_address_match(0);
            breakpoint_hit &= ~DBGBP0;
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

Set breakpoint to trigger on address match

Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

→ We are in an endless debugging loop

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

Set breakpoint to trigger on address match

➡ Forget that breakpoint was set

```
        }
    }
```

`...` Continue firmware execution

```
}
```

```
start:
MOV R0, #1
MOV R0, #3
B start
```

→ We are in an endless debugging loop

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

Set breakpoint to trigger on address match

Forget that breakpoint was set

```
        }
    }
```

➡ ... Continue firmware execution

```
}
```

```
start:
MOV R0, #1
● MOV R0, #3
B start
```

→ **We are in an endless debugging loop**

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
    - Handle and reset breakpoints
    - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

Check whether breakpoint is enabled
Check whether breakpoint address equals PC_SYS

Write code to handle the breakpoint

Remember which breakpoint was hit
Set breakpoint to trigger on address mismatch
Check whether we remember that breakpoint was set
Set breakpoint to trigger on address match
Forget that breakpoint was set

```
        }
    }

    ...
}
```

Continue firmware execution

```
  start:
  MOV R0, #1
● MOV R0, #3
➡ B start
```

→ **We are in an endless debugging loop**

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

Set breakpoint to trigger on address match

Forget that breakpoint was set

```
        }
    }

    ...
```

Continue firmware execution

```
}
```

```
  start:
➡ MOV R0, #1
● MOV R0, #3
  B start
```

→ **We are in an endless debugging loop**

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

Set breakpoint to trigger on address match

Forget that breakpoint was set

```
        }
    }

    ...
}
```

Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

→ **We are in an endless debugging loop**

Handling Breakpoints

Address Mismatch Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

Set breakpoint to trigger on address match

Forget that breakpoint was set

```
        }
    }
```

`...` **Continue firmware execution**

```
}
```

```
start:
MOV R0, #1
MOV R0, #3
B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address (0, trace->pc)) {

            // Handle Breakpoint:

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_type_to_instr_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {

            // Handle Breakpoint

            dbg_set_breakpoint_for_addr_mismatch(0, trace->pc);
        }
    }

    ...
}
```

```
start:
MOV R0, #1
MOV R0, #3
B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address (0, trace->pc)) {

            // Handle Breakpoint:

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_type_to_instr_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {

            // Handle Breakpoint

            dbg_set_breakpoint_for_addr_mismatch(0, trace->pc);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

```
  start:
  MOV R0, #1
● MOV R0, #3
  B start
```

➡ Abort Mode
➡ System Mode

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint:

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_type_to_instr_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {

            // Handle Breakpoint

            dbg_set_breakpoint_for_addr_mismatch(0, trace->pc);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

```
start:
MOV R0, #1
MOV R0, #3
B start
```

➡ Abort Mode
➡ System Mode

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_type_to_instr_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {

            // Handle Breakpoint

            dbg_set_breakpoint_for_addr_mismatch(0, trace->pc);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

```
start:
MOV R0, #1
● MOV R0, #3
B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit = DBGBP0;
            dbg_set_breakpoint_type_to_instr_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {

            // Handle Breakpoint

            dbg_set_breakpoint_for_addr_mismatch(0, trace->pc);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

```
start:
MOV R0, #1
● MOV R0, #3
B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit |= DBGBP0;
➡           dbg_set_breakpoint_for_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {

            // Handle Breakpoint

            dbg_set_breakpoint_for_addr_mismatch(0, trace->pc);
        }
    }

    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

```
  start:
  MOV R0, #1
● MOV R0, #3
  B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_type_to_instr_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {

            // Handle Breakpoint

            dbg_set_breakpoint_for_addr_mismatch(0, trace->pc);
        }
    }
➡   ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Continue firmware execution

```
  start:
  MOV R0, #1
● MOV R0, #3
  B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_for_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {

            // Handle Breakpoint

            dbg_set_breakpoint_for_addr_mismatch(0, trace->pc);
        }
    }
    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if ([illegible] trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit |= DBGBP0;
            db[illegible]tch(0);
        } else if (breakpoint_hit & DBGBP0) {

            // Handle Breakpoint

            dbg_set_breakpoint_for_addr_mismatch(0, trace->pc);
        }
    }
    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Continue firmware execution

```
  start:
  MOV R0, #1
● MOV R0, #3
➡ B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
    - Handle and reset breakpoints ✓
    - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_for_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {

            // Handle Breakpoint

            dbg_set_breakpoint_for_addr_mismatch(0, trace->pc);
        }
    }
    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Continue firmware execution

```
start:
MOV R0, #1
● MOV R0, #3
B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit |= DBGBP0;
            dbg_set_breakpoint_for_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBP0) {

            // Handle Breakpoint

            dbg_set_breakpoint_for_addr_mismatch(0, trace->pc);
        }
    }
    ...
}
```

Check whether breakpoint is enabled

➡ Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Continue firmware execution

```
  start:
  MOV R0, #1
● MOV R0, #3
  B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

➡ Check whether we remember that breakpoint was set

```
            // Handle Breakpoint

            dbg_set_breakpoint_for_addr_mismatch(0, trace->pc);
        }
    }
```

... Continue firmware execution

```
}
```

```
start:
MOV R0, #1
● MOV R0, #3
B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

➡ Write code to handle the single-stepping breakpoint

```
            dbg_set_breakpoint_for_addr_mismatch(0, trace->pc);
        }
    }
```

Continue firmware execution

```
    ...
}
```

```
start:
MOV R0, #1
MOV R0, #3
B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

Write code to handle the single-stepping breakpoint

➡ Set breakpoint to trigger on address match

```
        }
    }
```

Continue firmware execution

```
}
```

```
start:
MOV R0, #1
● MOV R0, #3
B start
```

➡ Abort Mode
➡ System Mode

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

Write code to handle the single-stepping breakpoint

Set breakpoint to trigger on address match

```
        }
    }
```

➡ `...` Continue firmware execution

```
}
```

```
start:
MOV R0, #1
MOV R0, #3
B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit |= DBGBB0;
            dbg_set_breakpoint_to_trigger_on_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBB0) {

            // Handle Breakpoint

            dbg_set_breakpoint_for_addr_mismatch(0, trace->pc);
        }
    }
    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

Write code to handle the single-stepping breakpoint

Set breakpoint to trigger on address match

Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit |= DBGBB0;
            dbg_set_breakpoint_for_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBB0) {

            // Handle Breakpoint

            dbg_set_breakpoint_for_addr_match(0, trace->pc);
        }
    }
    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

Write code to handle the single-stepping breakpoint

Set breakpoint to trigger on address match

Continue firmware execution

```
  start:
➡ MOV R0, #1
  MOV R0, #3
● B start
```

➡ Abort Mode
➡ System Mode

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

➡ Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

Write code to handle the single-stepping breakpoint

Set breakpoint to trigger on address match

```
        }
    }
```

Continue firmware execution

```
}
```

```
start:
MOV R0, #1
MOV R0, #3
● B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit |= DBGB0;
            dbg_set_breakpoint_to_trigger_on_address_mismatch(0);
        } else if (breakpoint_hit & DBGB0) {

            // Handle Breakpoint

            dbg_set_breakpoint_to_trigger_on_address_match(0, trace->pc);
        }
    }
    ...
}
```

Check whether breakpoint is enabled
Check whether breakpoint address equals PC_SYS
Write code to handle the first breakpoint
Remember which breakpoint was hit
Set breakpoint to trigger on address mismatch
Check whether we remember that breakpoint was set
Write code to handle the single-stepping breakpoint
Set breakpoint to trigger on address match
Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
    if (dbg_is_breakpoint_enabled(0)) {
        if (dbg_triggers_on_breakpoint_address(0, trace->pc)) {

            // Handle Breakpoint

            breakpoint_hit |= DBGBB0;
            dbg_set_breakpoint_for_addr_mismatch(0);
        } else if (breakpoint_hit & DBGBB0) {

            // Handle Breakpoint

            dbg_set_breakpoint_for_addr_match(0, trace->pc);
        }
    }
    ...
}
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

Write code to handle the single-stepping breakpoint

Set breakpoint to trigger on address match

Continue firmware execution

```
start:
MOV R0, #1
MOV R0, #3
B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

➡ Write code to handle the single-stepping breakpoint

Set breakpoint to trigger on address match

```
        }
    }
```

... Continue firmware execution

```
}
```

```
start:
MOV R0, #1
MOV R0, #3
● B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode

➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

Write code to handle the single-stepping breakpoint

➡ Set breakpoint to trigger on address match

```
        }
    }
```

Continue firmware execution

```
    ...
}
```

```
start:
MOV R0, #1
MOV R0, #3
B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

Write code to handle the single-stepping breakpoint

Set breakpoint to trigger on address match

```
        }
    }
```

➡ `...` Continue firmware execution

```
}
```

```
start:
● MOV R0, #1
  MOV R0, #3
  B start
```

Handling Breakpoints

Single-Stepping Implementation
Breakpoint triggers multiple times

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping ✓
- Activate breakpoints

➡ Abort Mode
➡ System Mode

```
void
handle_pref_abort_exception(struct trace *trace) {
    fix_sp_lr(trace);

    // Do for any of the four hardware breakpoints
```

Check whether breakpoint is enabled

Check whether breakpoint address equals PC_SYS

Write code to handle the first breakpoint

Remember which breakpoint was hit

Set breakpoint to trigger on address mismatch

Check whether we remember that breakpoint was set

Write code to handle the single-stepping breakpoint

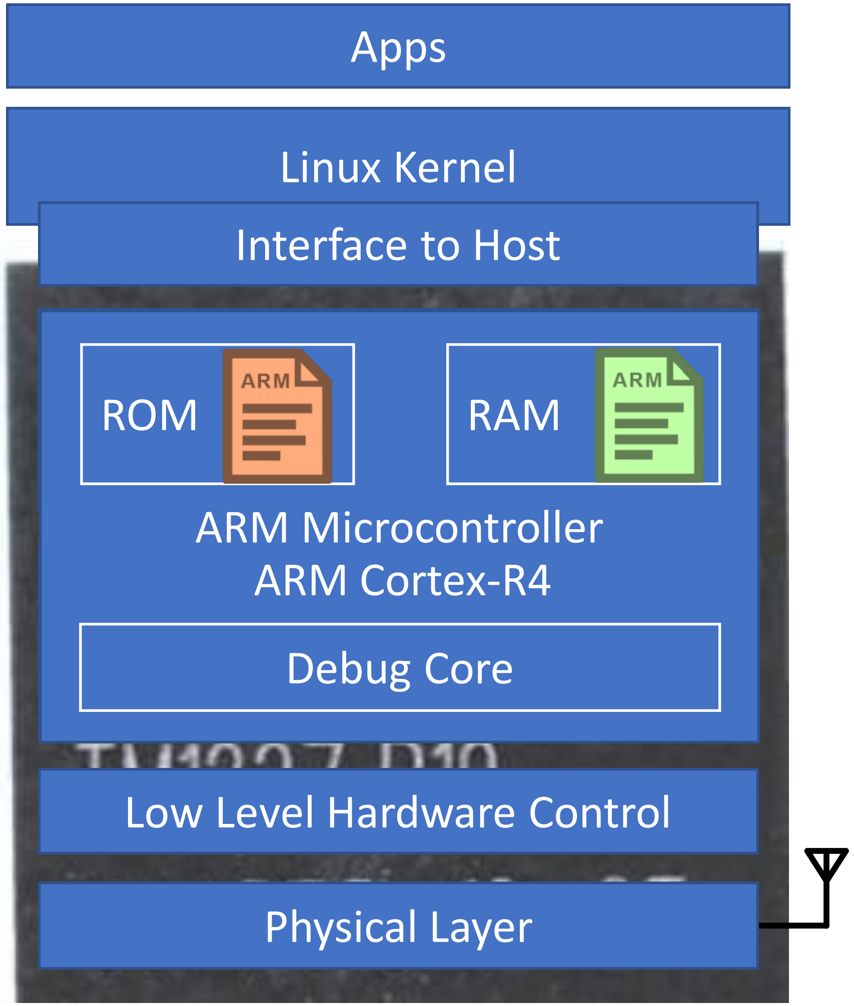
Set breakpoint to trigger on address match

```
        }
    }
    ...
}
```

Continue firmware execution

```
  start:
● MOV R0, #1
➡ MOV R0, #3
  B start
```

Activating Breakpoints



Apps

Linux Kernel

Interface to Host

ROM

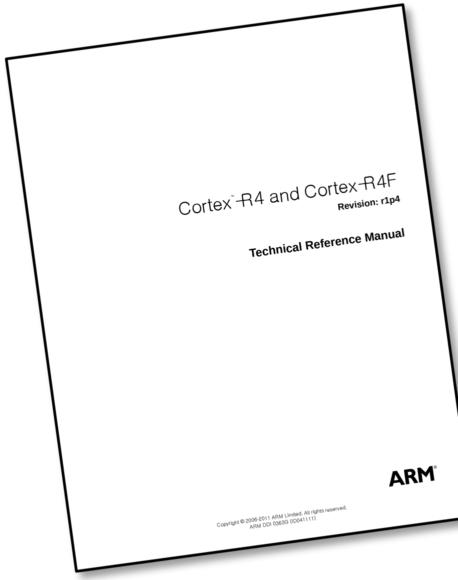
RAM

ARM Microcontroller
ARM Cortex-R4

Debug Core

Low Level Hardware Control

Physical Layer

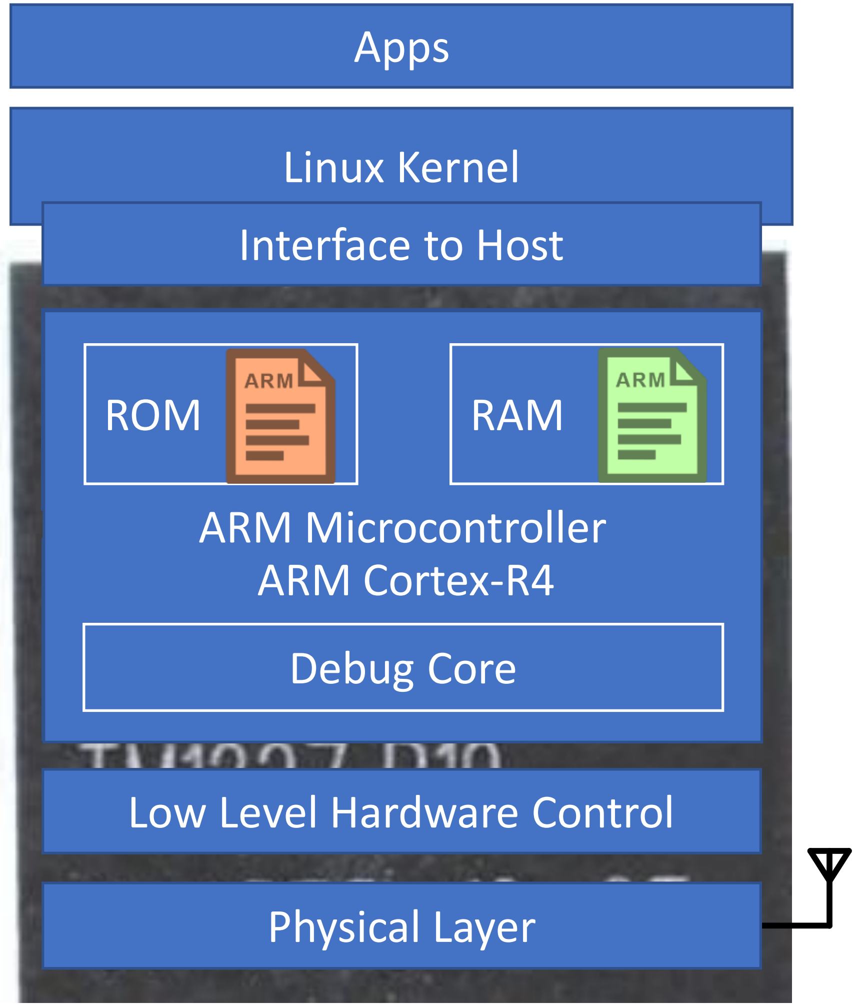


Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

Activating Breakpoints



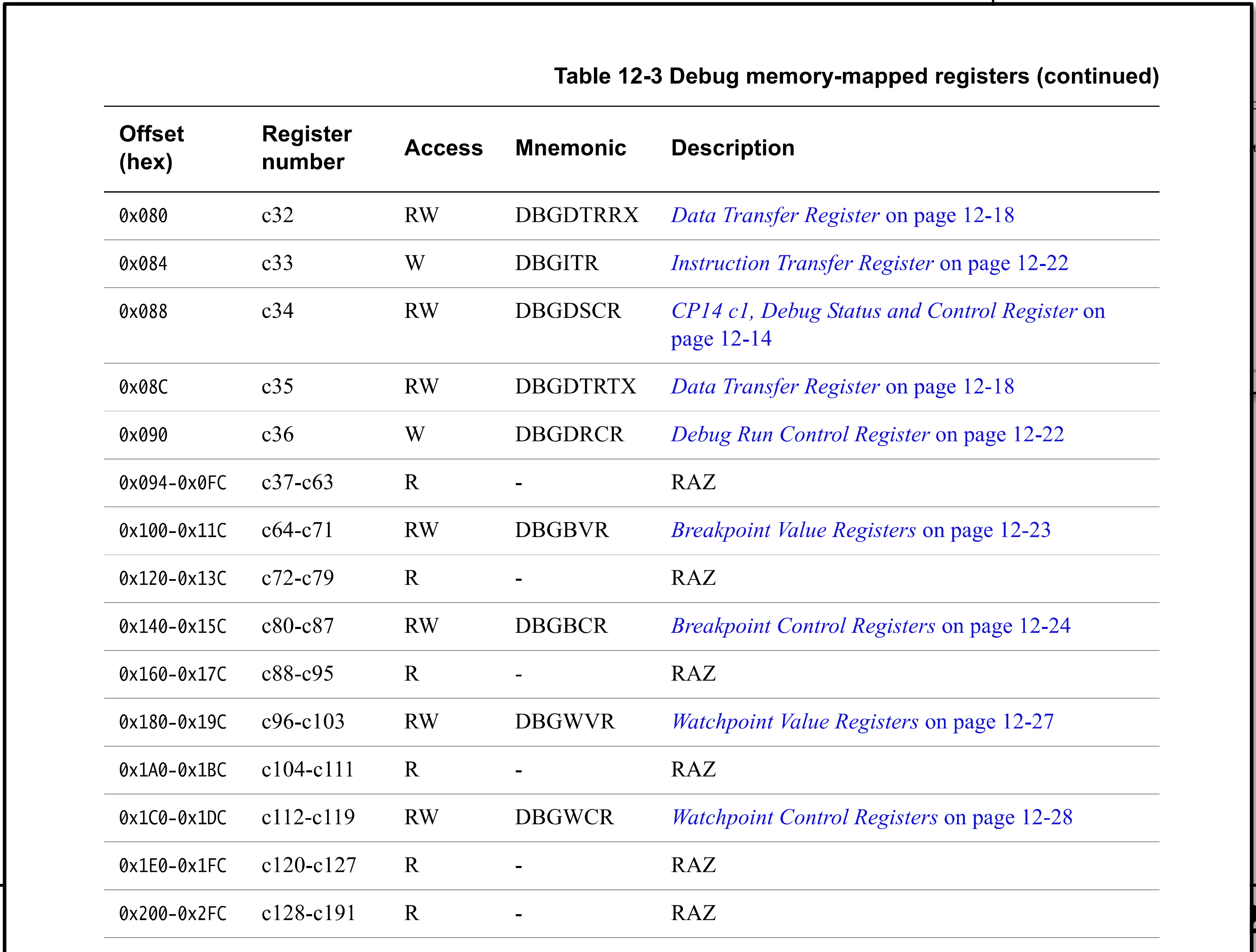
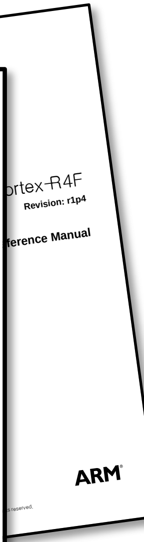


Table 12-3 Debug memory-mapped registers (continued)

Offset (hex)	Register number	Access	Mnemonic	Description
0x080	c32	RW	DBGDTRRX	*Data Transfer Register* on page 12-18
0x084	c33	W	DBGITR	*Instruction Transfer Register* on page 12-22
0x088	c34	RW	DBGDSCR	*CP14 c1, Debug Status and Control Register* on page 12-14
0x08C	c35	RW	DBGDTRTX	*Data Transfer Register* on page 12-18
0x090	c36	W	DBGDRCR	*Debug Run Control Register* on page 12-22
0x094-0x0FC	c37-c63	R	-	RAZ
0x100-0x11C	c64-c71	RW	DBGBVR	*Breakpoint Value Registers* on page 12-23
0x120-0x13C	c72-c79	R	-	RAZ
0x140-0x15C	c80-c87	RW	DBGBCR	*Breakpoint Control Registers* on page 12-24
0x160-0x17C	c88-c95	R	-	RAZ
0x180-0x19C	c96-c103	RW	DBGWVR	*Watchpoint Value Registers* on page 12-27
0x1A0-0x1BC	c104-c111	R	-	RAZ
0x1C0-0x1DC	c112-c119	RW	DBGWCR	*Watchpoint Control Registers* on page 12-28
0x1E0-0x1FC	c120-c127	R	-	RAZ
0x200-0x2FC	c128-c191	R	-	RAZ



Activating Breakpoints

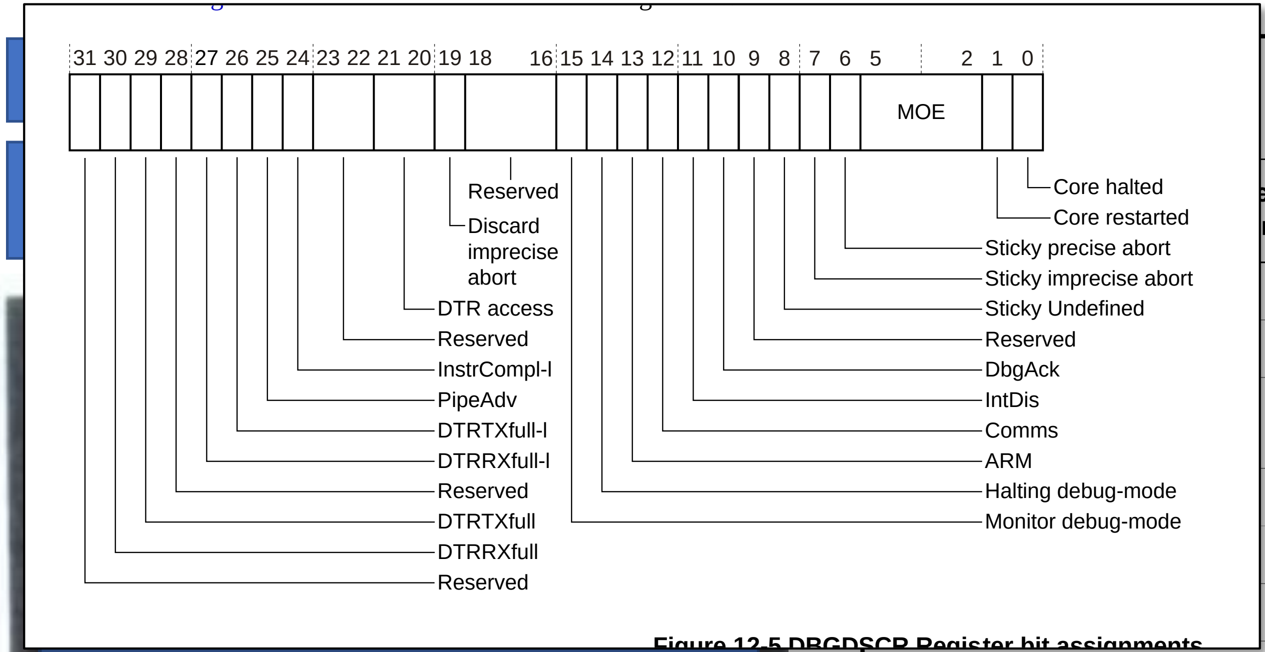


Figure 12-5 DBGDSCR Register bit assignments

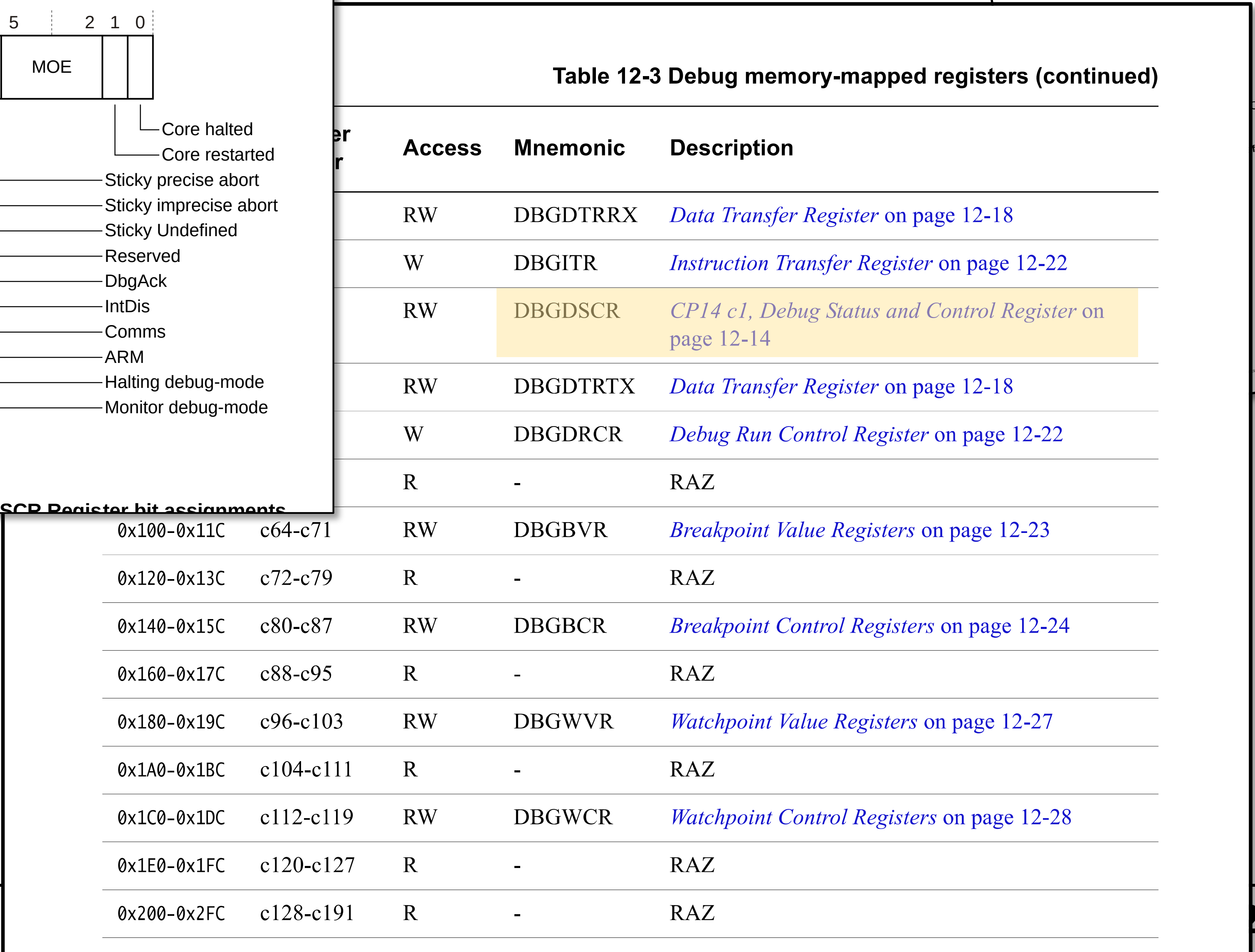
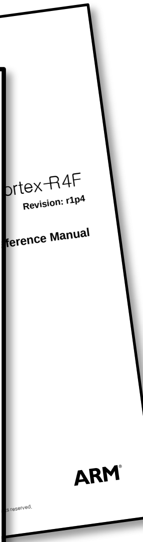
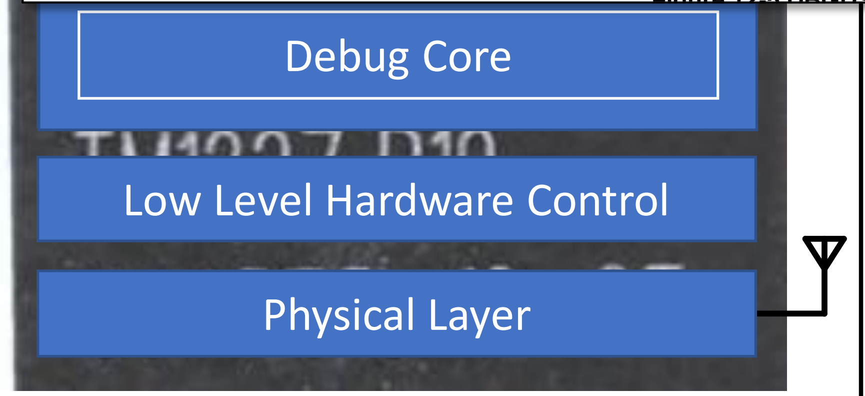


Table 12-3 Debug memory-mapped registers (continued)

		Access	Mnemonic	Description
		RW	DBGDTRRX	*Data Transfer Register* on page 12-18
		W	DBGITR	*Instruction Transfer Register* on page 12-22
		RW	DBGDSCR	*CP14 c1, Debug Status and Control Register* on page 12-14
		RW	DBGDTRTX	*Data Transfer Register* on page 12-18
		W	DBGDRCR	*Debug Run Control Register* on page 12-22
		R	-	RAZ
`0x100-0x11C`	c64-c71	RW	DBGBVR	*Breakpoint Value Registers* on page 12-23
`0x120-0x13C`	c72-c79	R	-	RAZ
`0x140-0x15C`	c80-c87	RW	DBGBCR	*Breakpoint Control Registers* on page 12-24
`0x160-0x17C`	c88-c95	R	-	RAZ
`0x180-0x19C`	c96-c103	RW	DBGWVR	*Watchpoint Value Registers* on page 12-27
`0x1A0-0x1BC`	c104-c111	R	-	RAZ
`0x1C0-0x1DC`	c112-c119	RW	DBGWCR	*Watchpoint Control Registers* on page 12-28
`0x1E0-0x1FC`	c120-c127	R	-	RAZ
`0x200-0x2FC`	c128-c191	R	-	RAZ





Debug Core

Low Level Hardware Control

Physical Layer

Activating Breakpoints

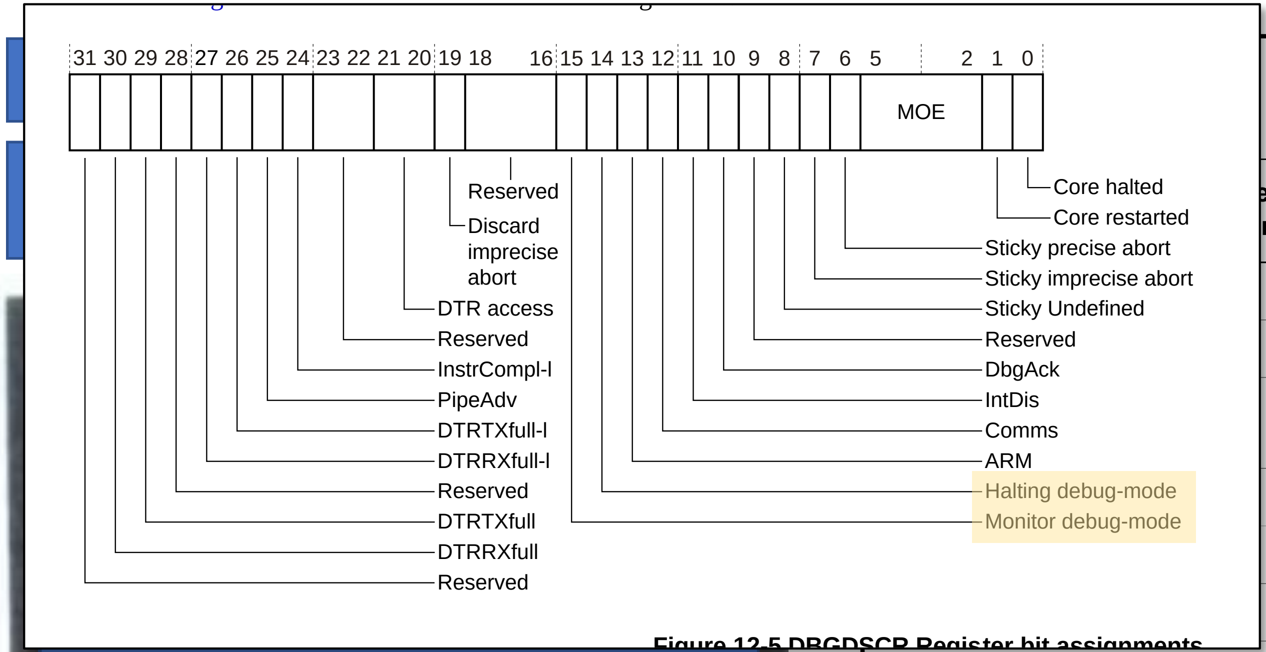


Figure 12-5 DBGDSCR Register bit assignments

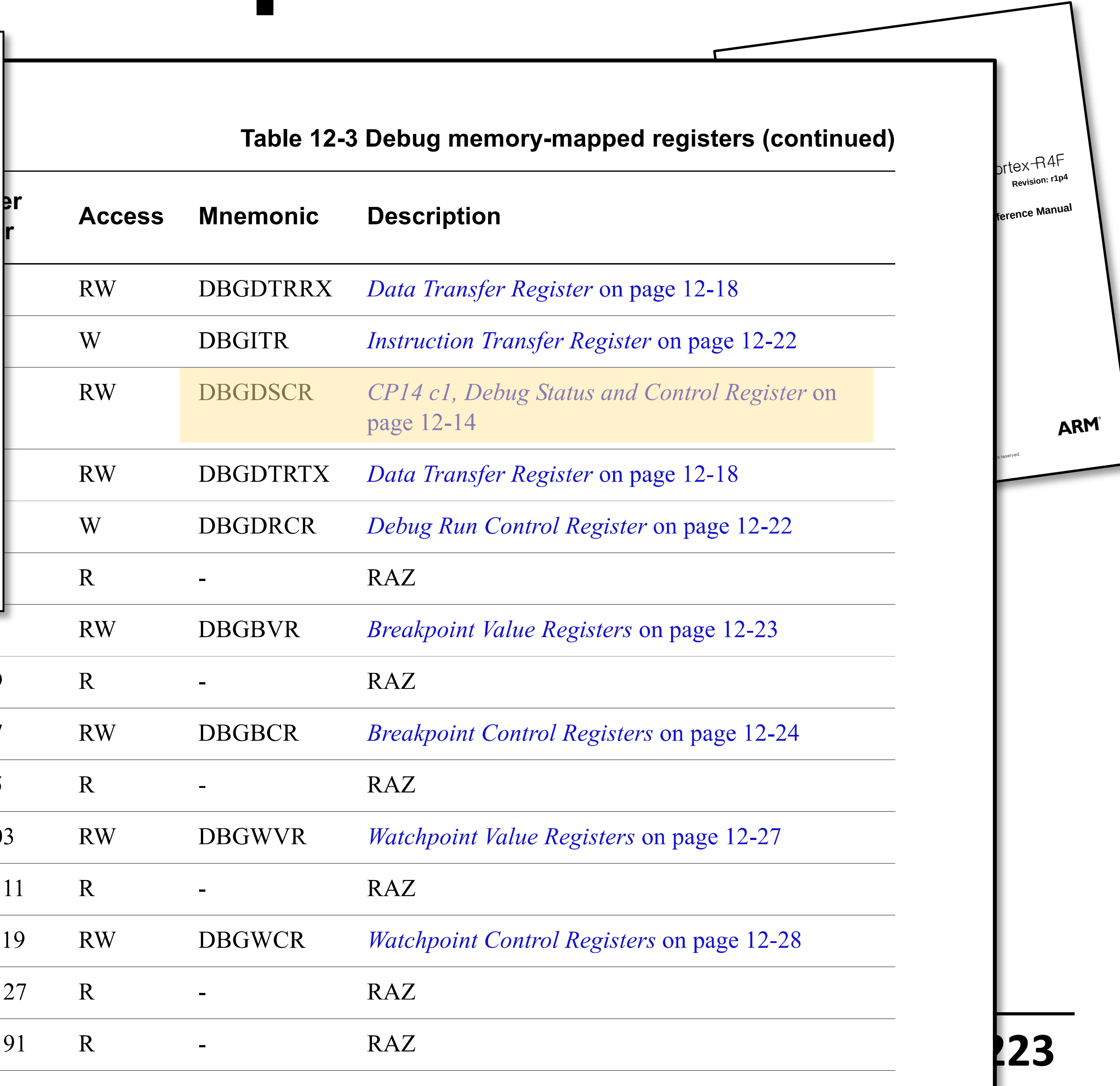
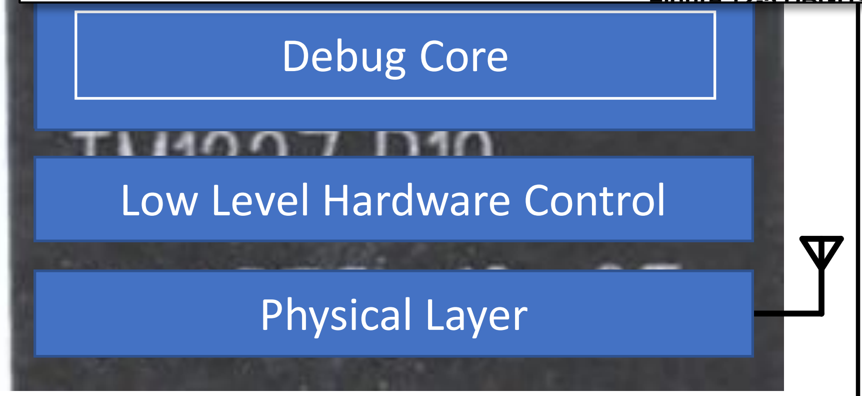
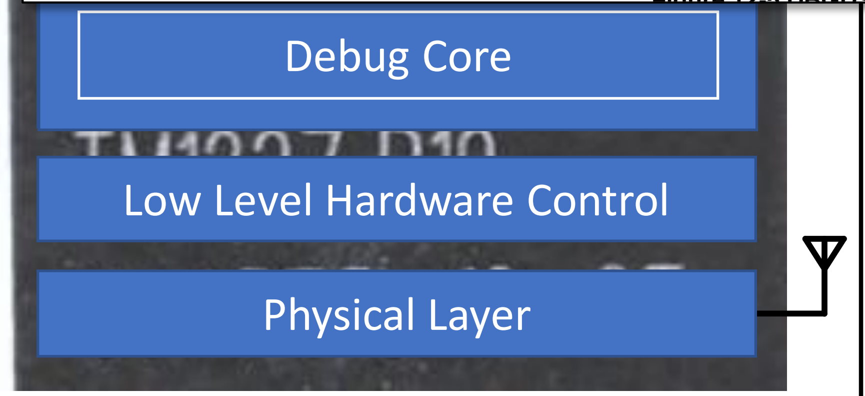


Table 12-3 Debug memory-mapped registers (continued)

	er r	Access	Mnemonic	Description
		RW	DBGDTRRX	*Data Transfer Register* on page 12-18
		W	DBGITR	*Instruction Transfer Register* on page 12-22
		RW	DBGDSCR	*CP14 c1, Debug Status and Control Register* on page 12-14
		RW	DBGDTRTX	*Data Transfer Register* on page 12-18
		W	DBGDRCR	*Debug Run Control Register* on page 12-22
		R	-	RAZ
0x100-0x11C	c64-c71	RW	DBGBVR	*Breakpoint Value Registers* on page 12-23
0x120-0x13C	c72-c79	R	-	RAZ
0x140-0x15C	c80-c87	RW	DBGBCR	*Breakpoint Control Registers* on page 12-24
0x160-0x17C	c88-c95	R	-	RAZ
0x180-0x19C	c96-c103	RW	DBGWVR	*Watchpoint Value Registers* on page 12-27
0x1A0-0x1BC	c104-c111	R	-	RAZ
0x1C0-0x1DC	c112-c119	RW	DBGWCR	*Watchpoint Control Registers* on page 12-28
0x1E0-0x1FC	c120-c127	R	-	RAZ
0x200-0x2FC	c128-c191	R	-	RAZ



Activating Breakpoints



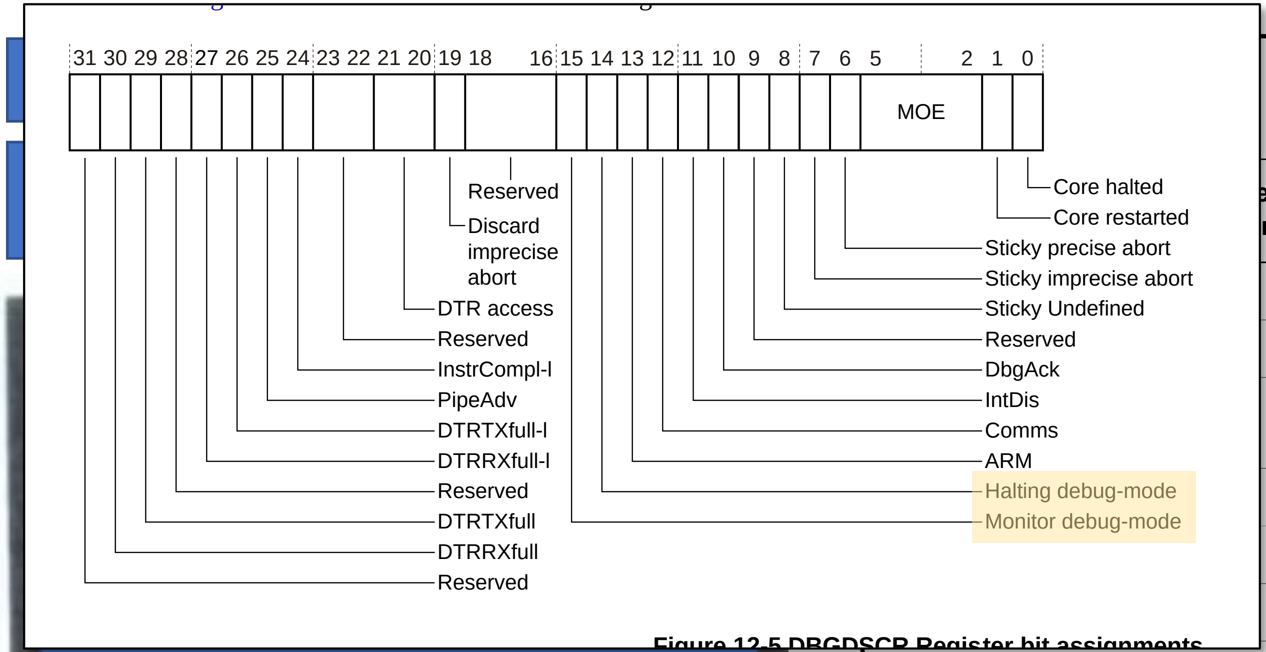


Figure 12-5 DBGDSCR Register bit assignments

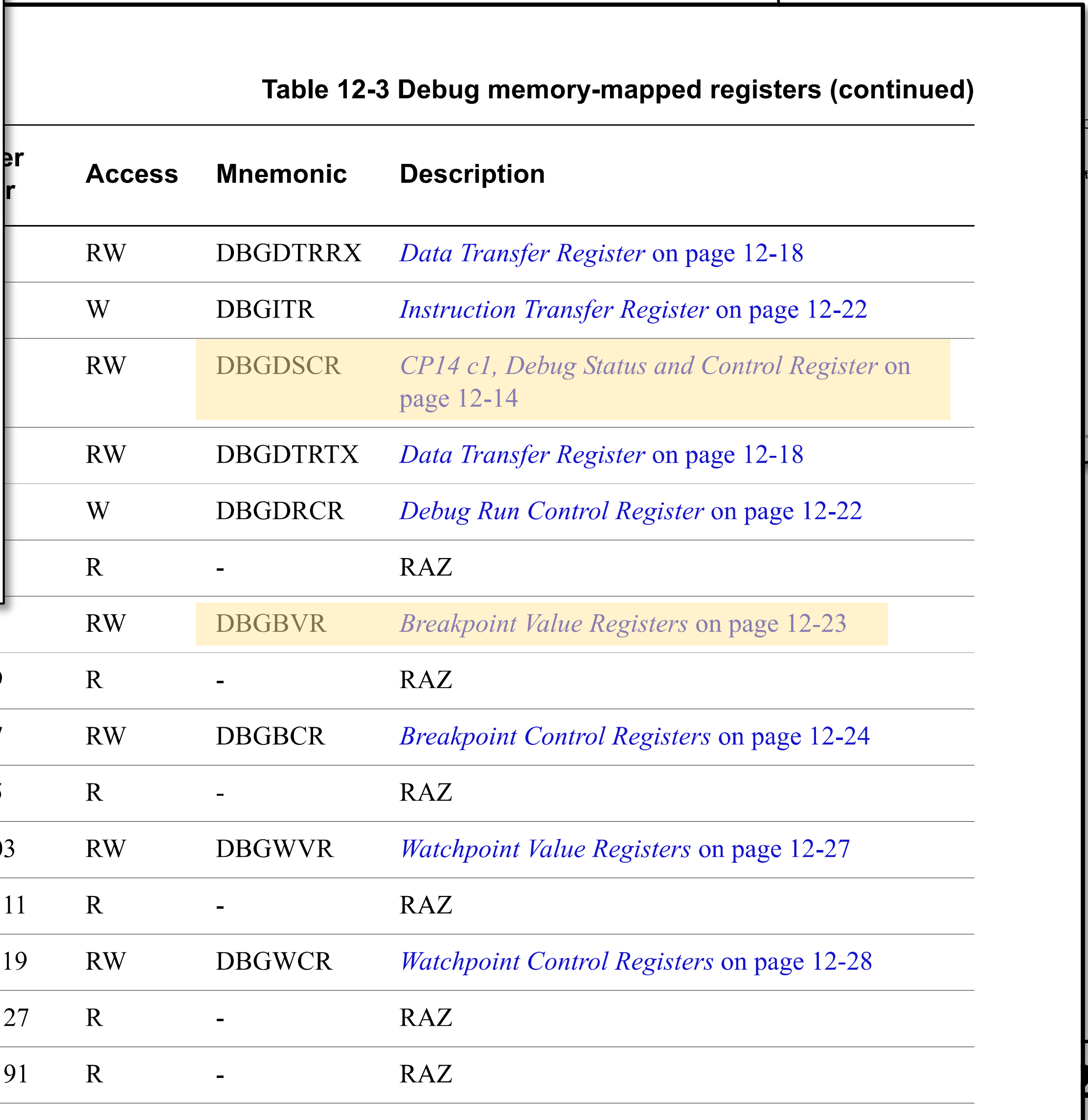
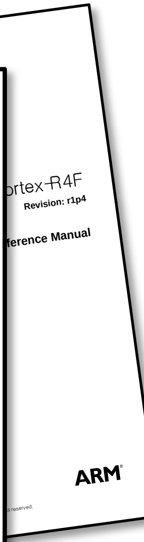


Table 12-3 Debug memory-mapped registers (continued)

		Access	Mnemonic	Description
		RW	DBGDTRRX	*Data Transfer Register* on page 12-18
		W	DBGITR	*Instruction Transfer Register* on page 12-22
		RW	DBGDSCR	*CP14 c1, Debug Status and Control Register* on page 12-14
		RW	DBGDTRTX	*Data Transfer Register* on page 12-18
		W	DBGDRCR	*Debug Run Control Register* on page 12-22
		R	-	RAZ
0x100-0x11C	c64-c71	RW	DBGBVR	*Breakpoint Value Registers* on page 12-23
0x120-0x13C	c72-c79	R	-	RAZ
0x140-0x15C	c80-c87	RW	DBGBCR	*Breakpoint Control Registers* on page 12-24
0x160-0x17C	c88-c95	R	-	RAZ
0x180-0x19C	c96-c103	RW	DBGWVR	*Watchpoint Value Registers* on page 12-27
0x1A0-0x1BC	c104-c111	R	-	RAZ
0x1C0-0x1DC	c112-c119	RW	DBGWCR	*Watchpoint Control Registers* on page 12-28
0x1E0-0x1FC	c120-c127	R	-	RAZ
0x200-0x2FC	c128-c191	R	-	RAZ



Activating Breakpoints

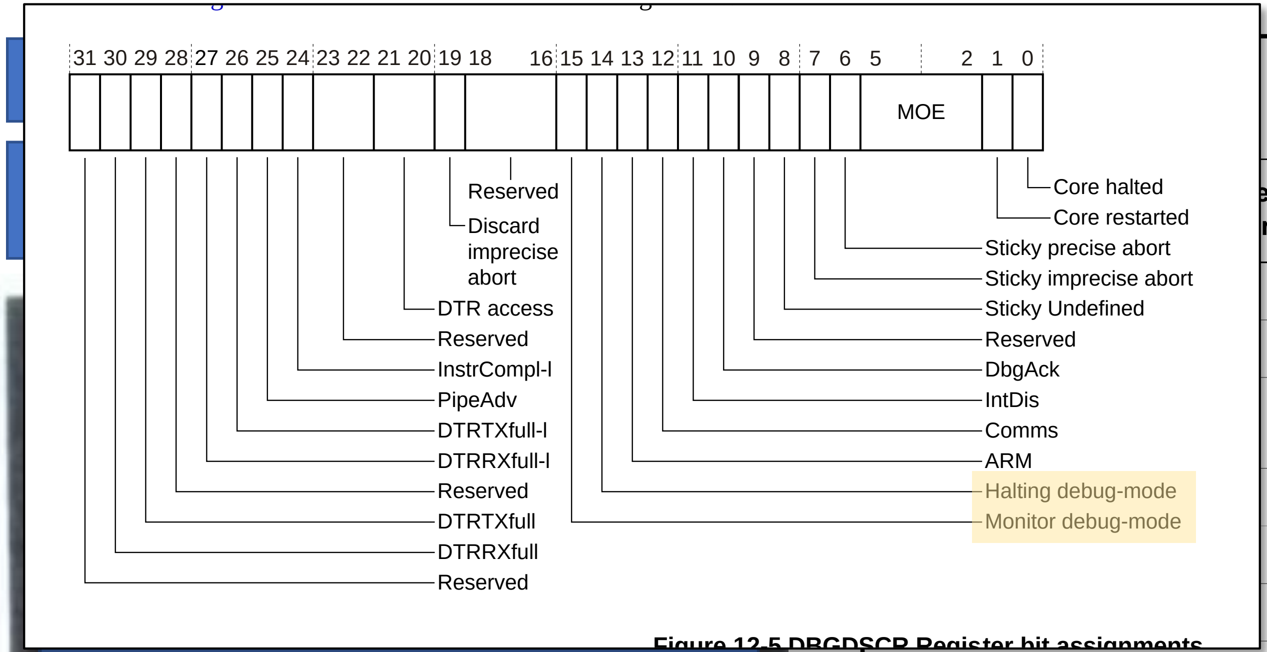
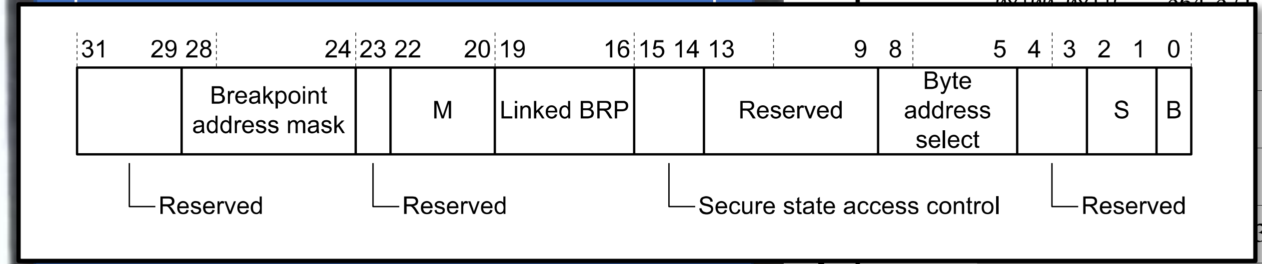
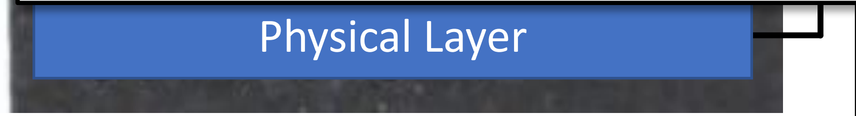


Figure 12-5 DBGDSCR Register bit assignments





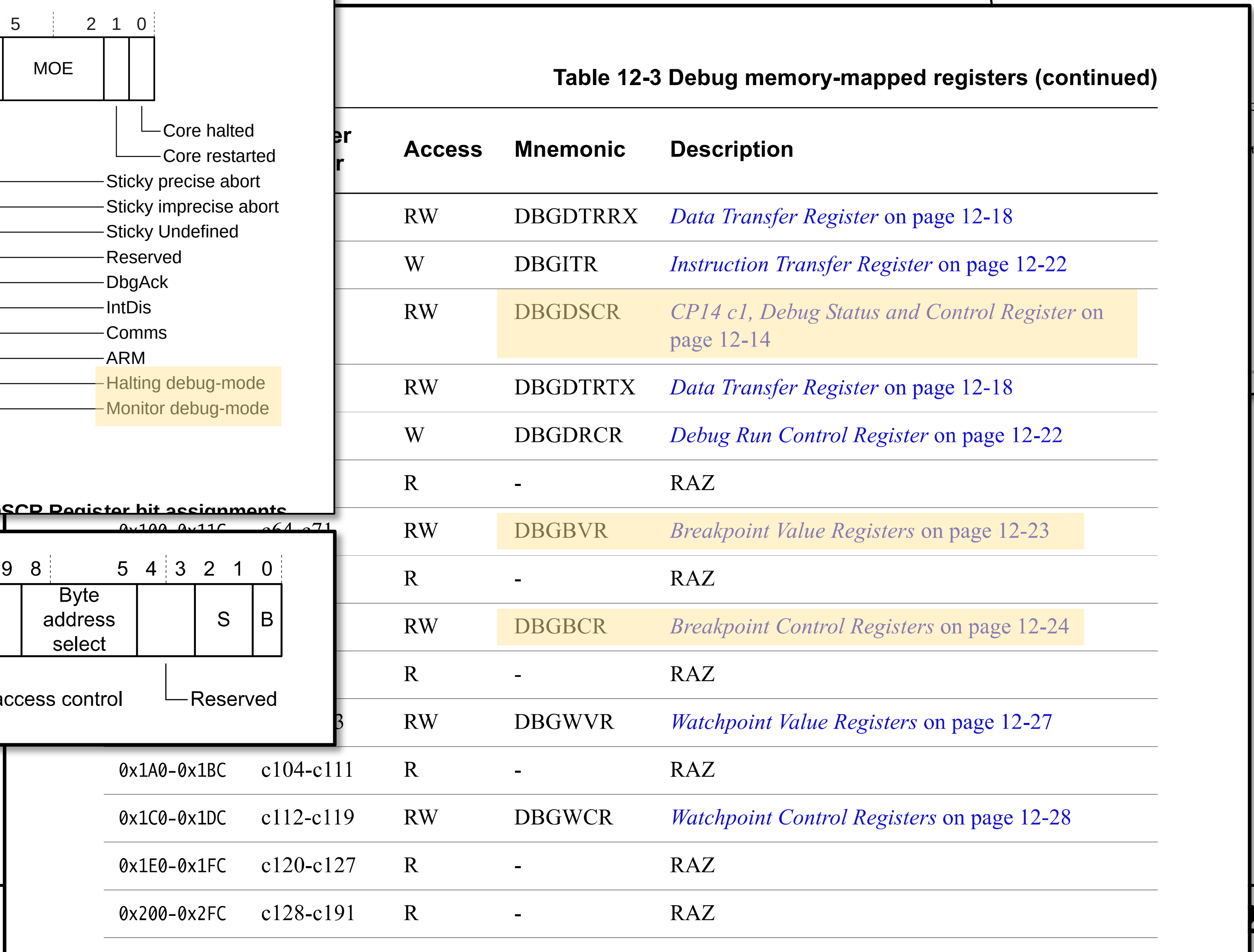
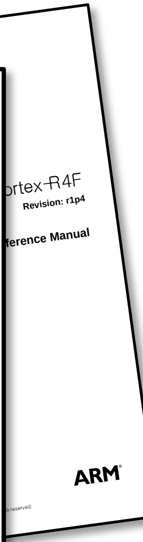
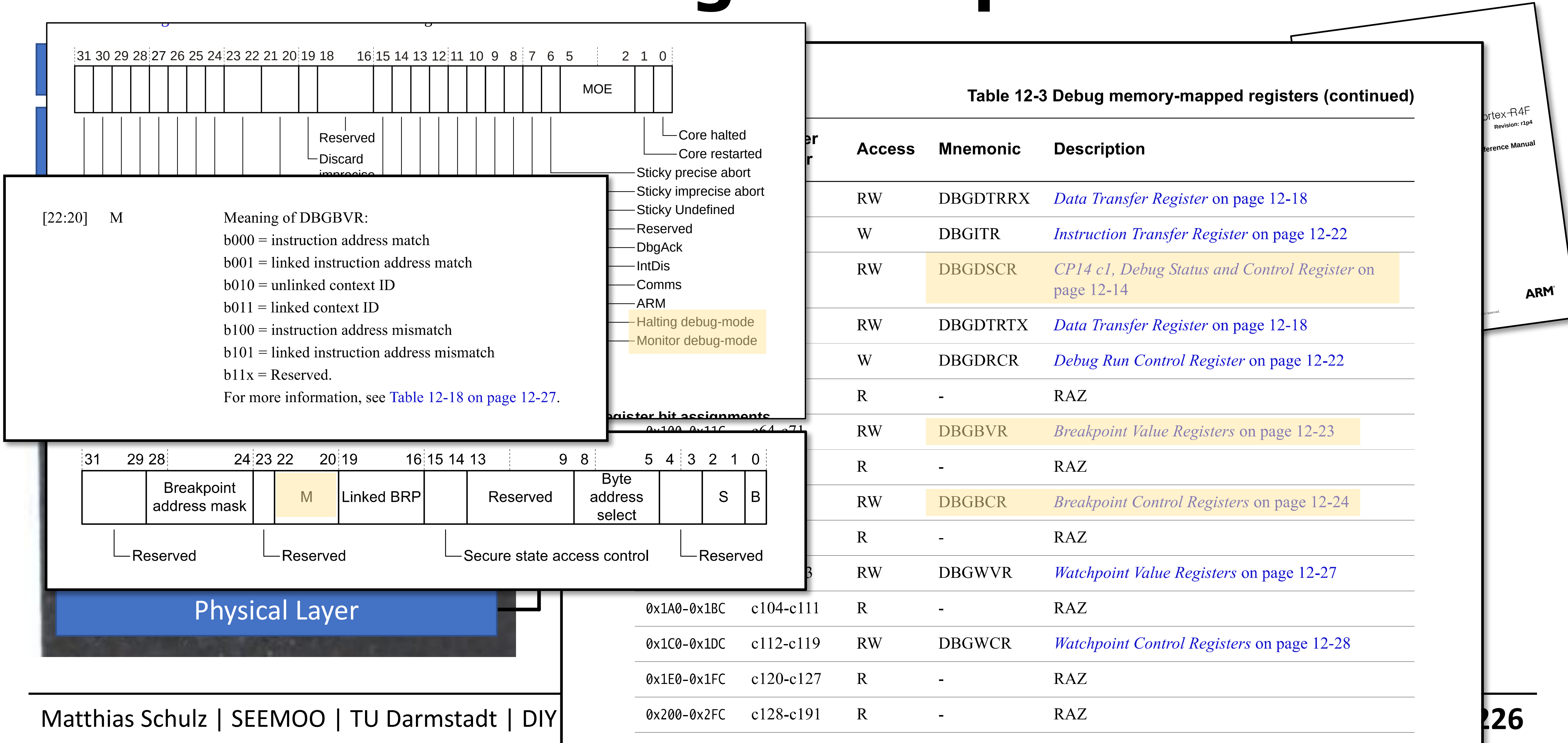


Table 12-3 Debug memory-mapped registers (continued)

		Access	Mnemonic	Description
		RW	DBGDTRRX	*Data Transfer Register* on page 12-18
		W	DBGITR	*Instruction Transfer Register* on page 12-22
		RW	DBGDSCR	*CP14 c1, Debug Status and Control Register* on page 12-14
		RW	DBGDTRTX	*Data Transfer Register* on page 12-18
		W	DBGDRCR	*Debug Run Control Register* on page 12-22
		R	-	RAZ
0x100-0x11C	c64-c71	RW	DBGBVR	*Breakpoint Value Registers* on page 12-23
		R	-	RAZ
		RW	DBGBCR	*Breakpoint Control Registers* on page 12-24
		R	-	RAZ
		RW	DBGWVR	*Watchpoint Value Registers* on page 12-27
0x1A0-0x1BC	c104-c111	R	-	RAZ
0x1C0-0x1DC	c112-c119	RW	DBGWCR	*Watchpoint Control Registers* on page 12-28
0x1E0-0x1FC	c120-c127	R	-	RAZ
0x200-0x2FC	c128-c191	R	-	RAZ



Activating Breakpoints

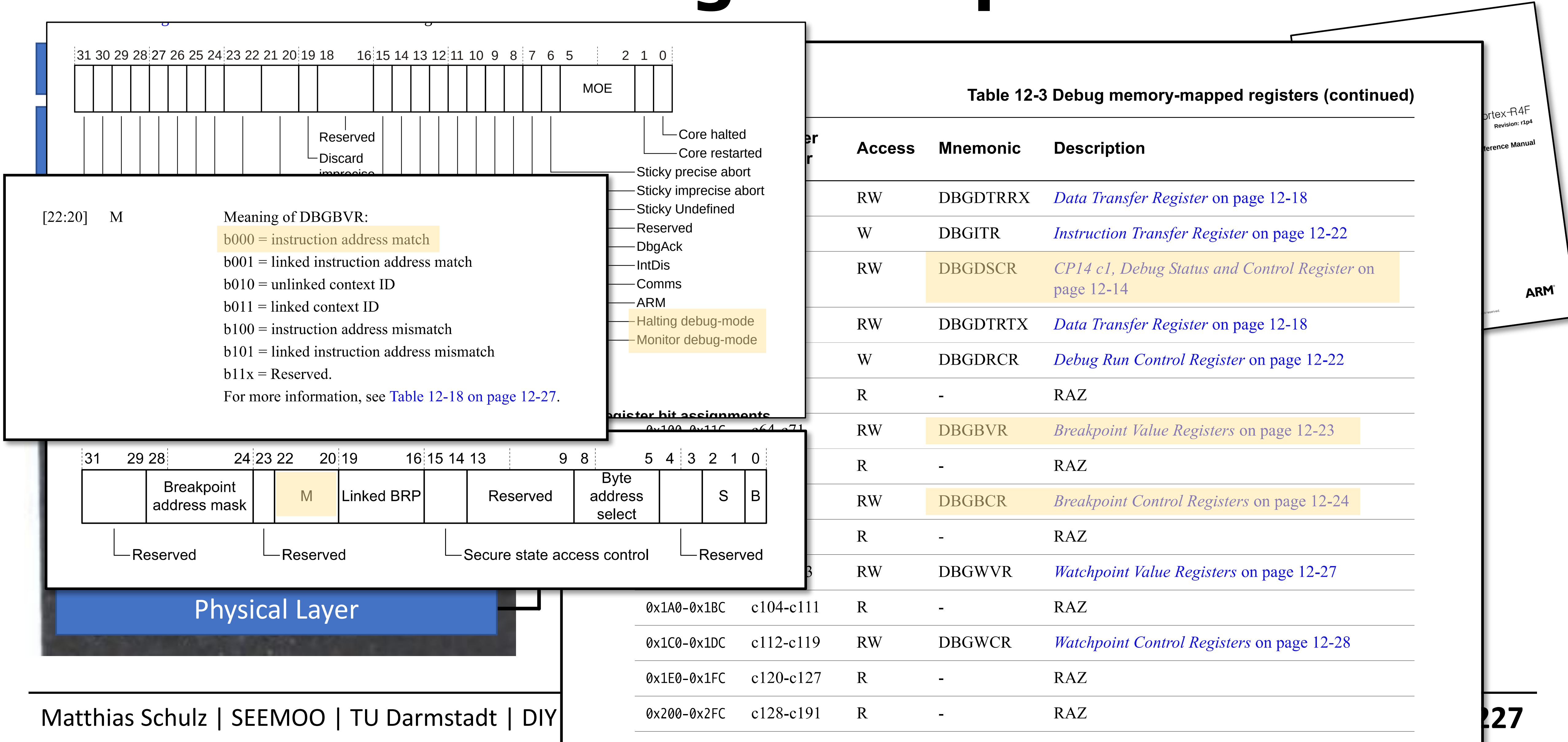


[22:20] M Meaning of DBGBVR:
b000 = instruction address match
b001 = linked instruction address match
b010 = unlinked context ID
b011 = linked context ID
b100 = instruction address mismatch
b101 = linked instruction address mismatch
b11x = Reserved.
For more information, see Table 12-18 on page 12-27.

Table 12-3 Debug memory-mapped registers (continued)

		Access	Mnemonic	Description
		RW	DBGDTRRX	*Data Transfer Register* on page 12-18
		W	DBGITR	*Instruction Transfer Register* on page 12-22
		RW	DBGDSCR	*CP14 c1, Debug Status and Control Register* on page 12-14
		RW	DBGDTRTX	*Data Transfer Register* on page 12-18
		W	DBGDRCR	*Debug Run Control Register* on page 12-22
		R	-	RAZ
0x100-0x11C	c64-c71	RW	DBGBVR	*Breakpoint Value Registers* on page 12-23
		R	-	RAZ
		RW	DBGBCR	*Breakpoint Control Registers* on page 12-24
		R	-	RAZ
		RW	DBGWVR	*Watchpoint Value Registers* on page 12-27
`0x1A0-0x1BC`	c104-c111	R	-	RAZ
`0x1C0-0x1DC`	c112-c119	RW	DBGWCR	*Watchpoint Control Registers* on page 12-28
`0x1E0-0x1FC`	c120-c127	R	-	RAZ
`0x200-0x2FC`	c128-c191	R	-	RAZ

Activating Breakpoints



[22:20] M

Meaning of DBGBVR:

b000 = instruction address match

b001 = linked instruction address match

b010 = unlinked context ID

b011 = linked context ID

b100 = instruction address mismatch

b101 = linked instruction address mismatch

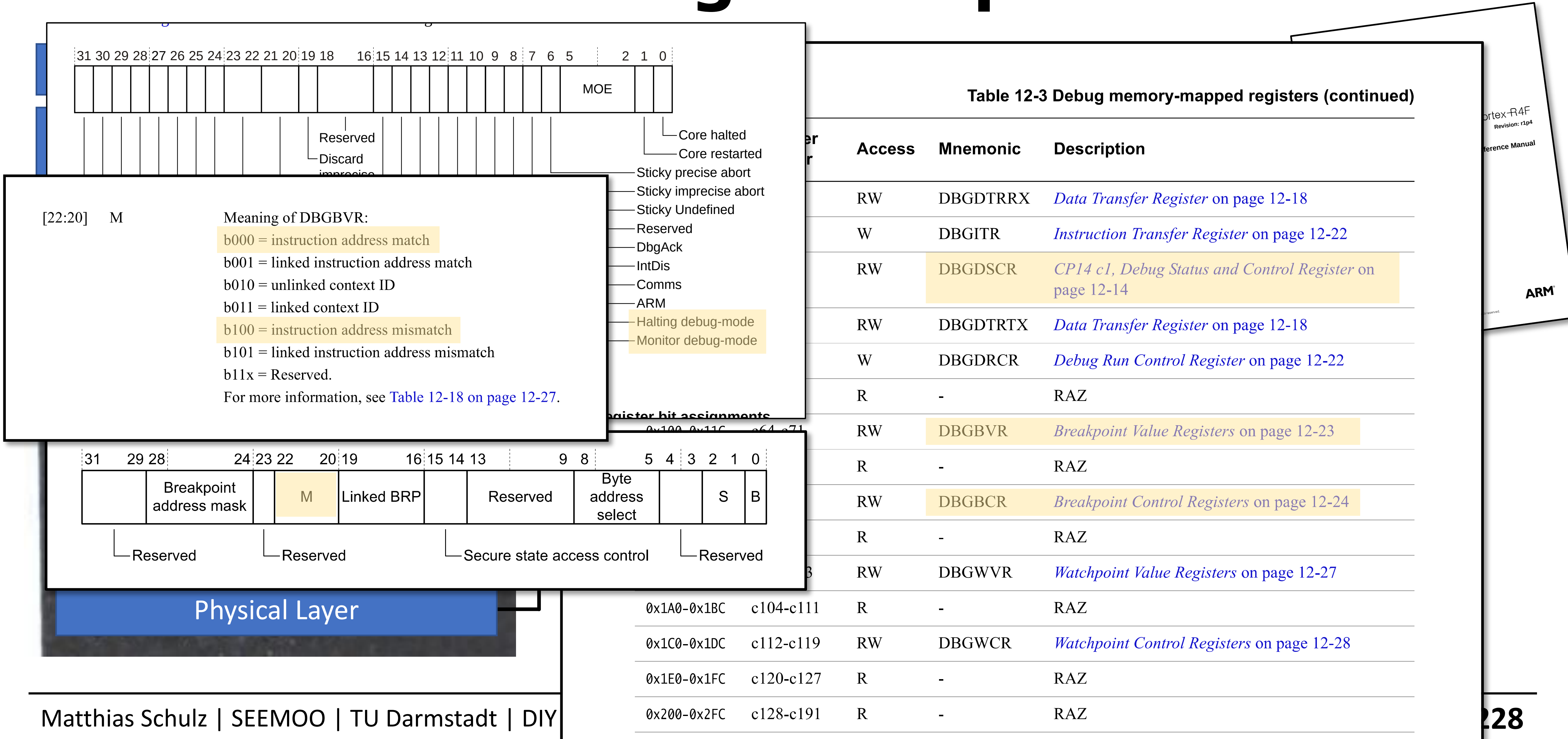
b11x = Reserved.

For more information, see Table 12-18 on page 12-27.

Table 12-3 Debug memory-mapped registers (continued)

		Access	Mnemonic	Description
		RW	DBGDTRRX	*Data Transfer Register* on page 12-18
		W	DBGITR	*Instruction Transfer Register* on page 12-22
		RW	DBGDSCR	*CP14 c1, Debug Status and Control Register* on page 12-14
		RW	DBGDTRTX	*Data Transfer Register* on page 12-18
		W	DBGDRCR	*Debug Run Control Register* on page 12-22
		R	-	RAZ
		RW	DBGBVR	*Breakpoint Value Registers* on page 12-23
		R	-	RAZ
		RW	DBGBCR	*Breakpoint Control Registers* on page 12-24
		R	-	RAZ
		RW	DBGWVR	*Watchpoint Value Registers* on page 12-27
0x1A0-0x1BC	c104-c111	R	-	RAZ
0x1C0-0x1DC	c112-c119	RW	DBGWCR	*Watchpoint Control Registers* on page 12-28
0x1E0-0x1FC	c120-c127	R	-	RAZ
0x200-0x2FC	c128-c191	R	-	RAZ

Activating Breakpoints



[22:20] M Meaning of DBGBVR:

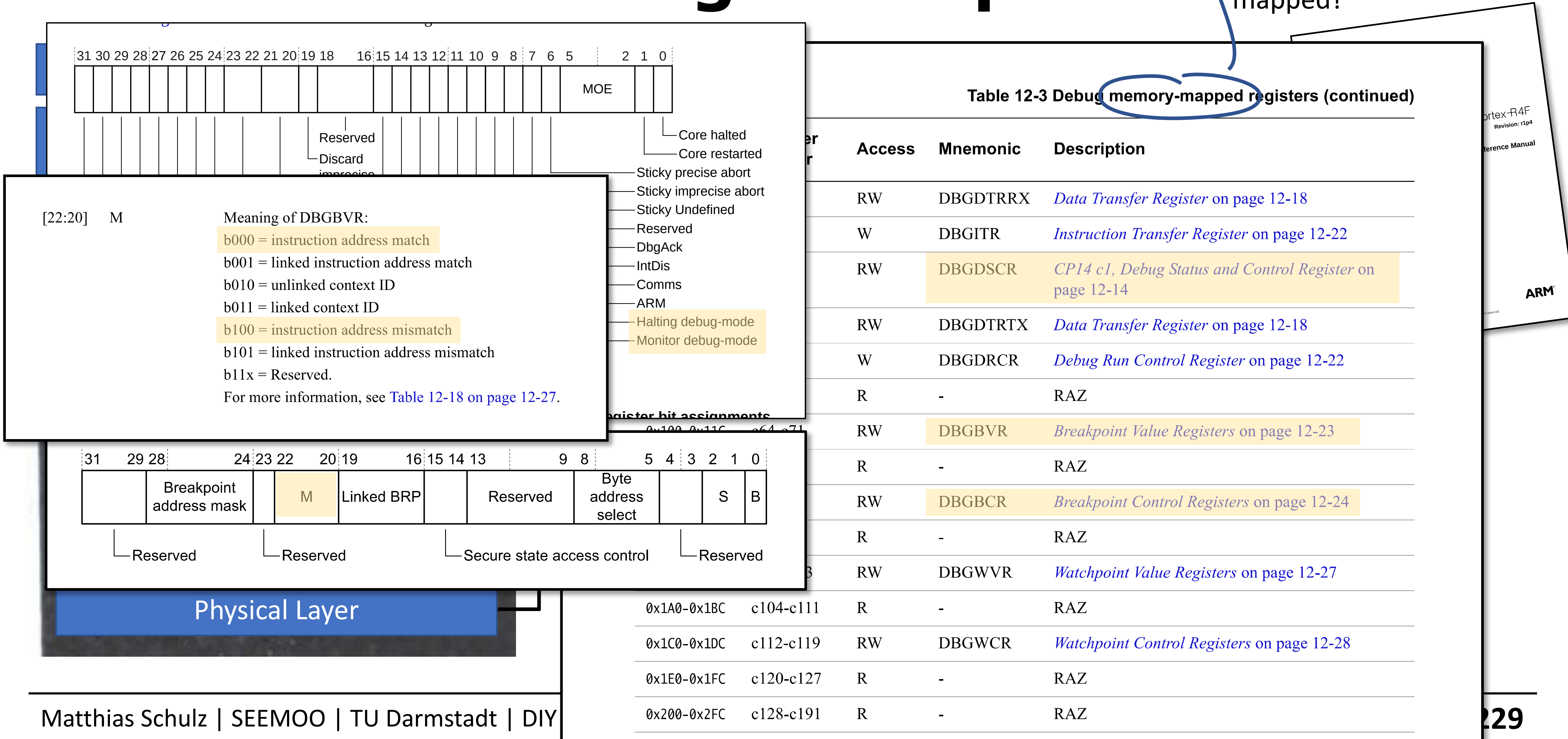
b000 = instruction address match
b001 = linked instruction address match
b010 = unlinked context ID
b011 = linked context ID
b100 = instruction address mismatch
b101 = linked instruction address mismatch
b11x = Reserved.
For more information, see Table 12-18 on page 12-27.

Table 12-3 Debug memory-mapped registers (continued)

	er r	Access	Mnemonic	Description
		RW	DBGDTRRX	*Data Transfer Register* on page 12-18
		W	DBGITR	*Instruction Transfer Register* on page 12-22
		RW	DBGDSCR	*CP14 c1, Debug Status and Control Register* on page 12-14
		RW	DBGDTRTX	*Data Transfer Register* on page 12-18
		W	DBGDRCR	*Debug Run Control Register* on page 12-22
		R	-	RAZ
0x100-0x11C	c64-c71	RW	DBGBVR	*Breakpoint Value Registers* on page 12-23
		R	-	RAZ
		RW	DBGBCR	*Breakpoint Control Registers* on page 12-24
		R	-	RAZ
	3	RW	DBGWVR	*Watchpoint Value Registers* on page 12-27
0x1A0-0x1BC	c104-c111	R	-	RAZ
0x1C0-0x1DC	c112-c119	RW	DBGWCR	*Watchpoint Control Registers* on page 12-28
0x1E0-0x1FC	c120-c127	R	-	RAZ
0x200-0x2FC	c128-c191	R	-	RAZ

Activating Breakpoints

To which addresses are the debugging registers mapped?



[22:20] M

Meaning of DBGBVR:

b000 = instruction address match
b001 = linked instruction address match
b010 = unlinked context ID
b011 = linked context ID
b100 = instruction address mismatch
b101 = linked instruction address mismatch
b11x = Reserved.

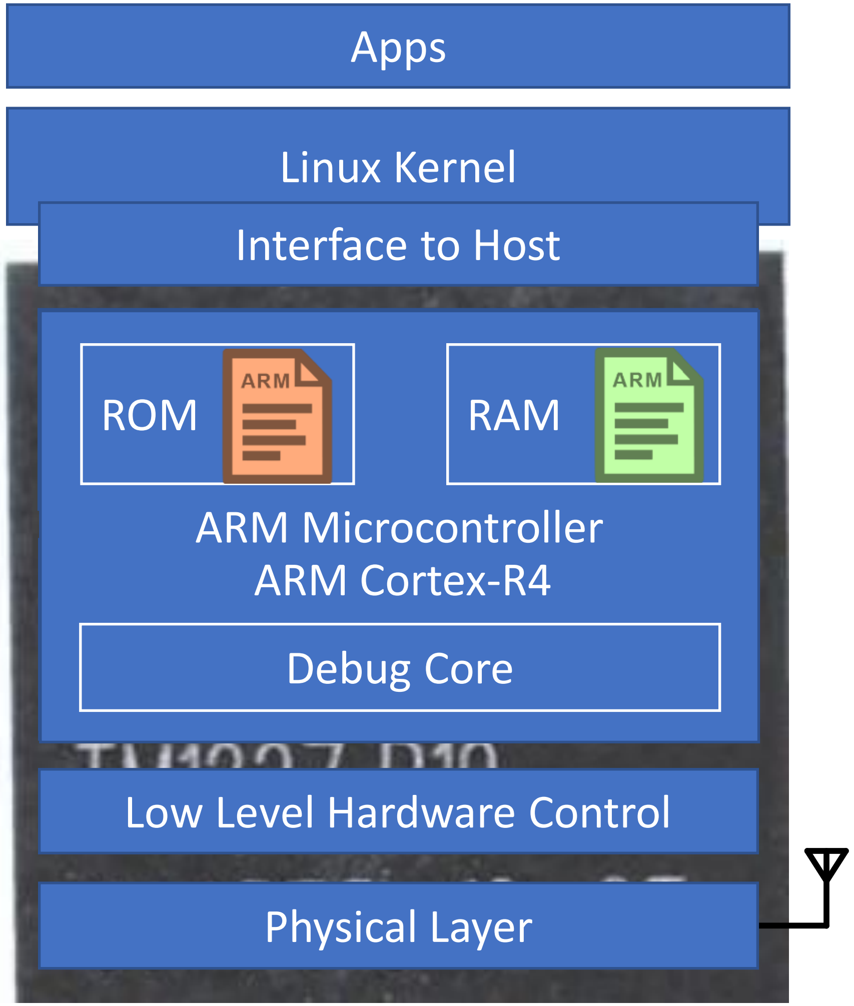
For more information, see Table 12-18 on page 12-27.

Table 12-3 Debug memory-mapped registers (continued)

		Access	Mnemonic	Description
		RW	DBGDTRRX	*Data Transfer Register* on page 12-18
		W	DBGITR	*Instruction Transfer Register* on page 12-22
		RW	DBGDSCR	*CP14 c1, Debug Status and Control Register* on page 12-14
		RW	DBGDTRTX	*Data Transfer Register* on page 12-18
		W	DBGDRCR	*Debug Run Control Register* on page 12-22
		R	-	RAZ
0x100-0x11C	c64-c71	RW	DBGBVR	*Breakpoint Value Registers* on page 12-23
		R	-	RAZ
		RW	DBGBCR	*Breakpoint Control Registers* on page 12-24
		R	-	RAZ
		RW	DBGWVR	*Watchpoint Value Registers* on page 12-27
0x1A0-0x1BC	c104-c111	R	-	RAZ
0x1C0-0x1DC	c112-c119	RW	DBGWCR	*Watchpoint Control Registers* on page 12-28
0x1E0-0x1FC	c120-c127	R	-	RAZ
0x200-0x2FC	c128-c191	R	-	RAZ

Activating Breakpoints

To which addresses are the debugging registers mapped?



Apps

Linux Kernel

Interface to Host

ROM

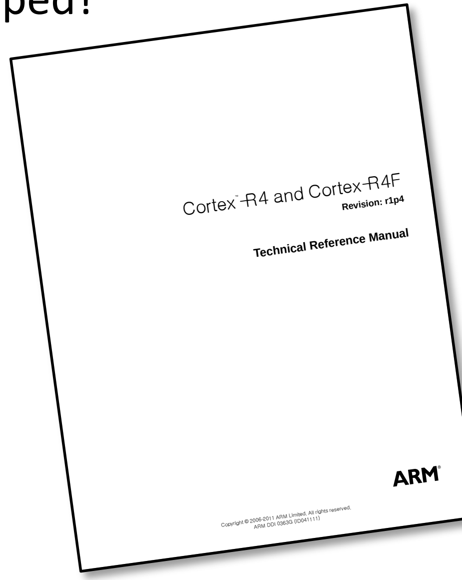
RAM

ARM Microcontroller
ARM Cortex-R4

Debug Core

Low Level Hardware Control

Physical Layer

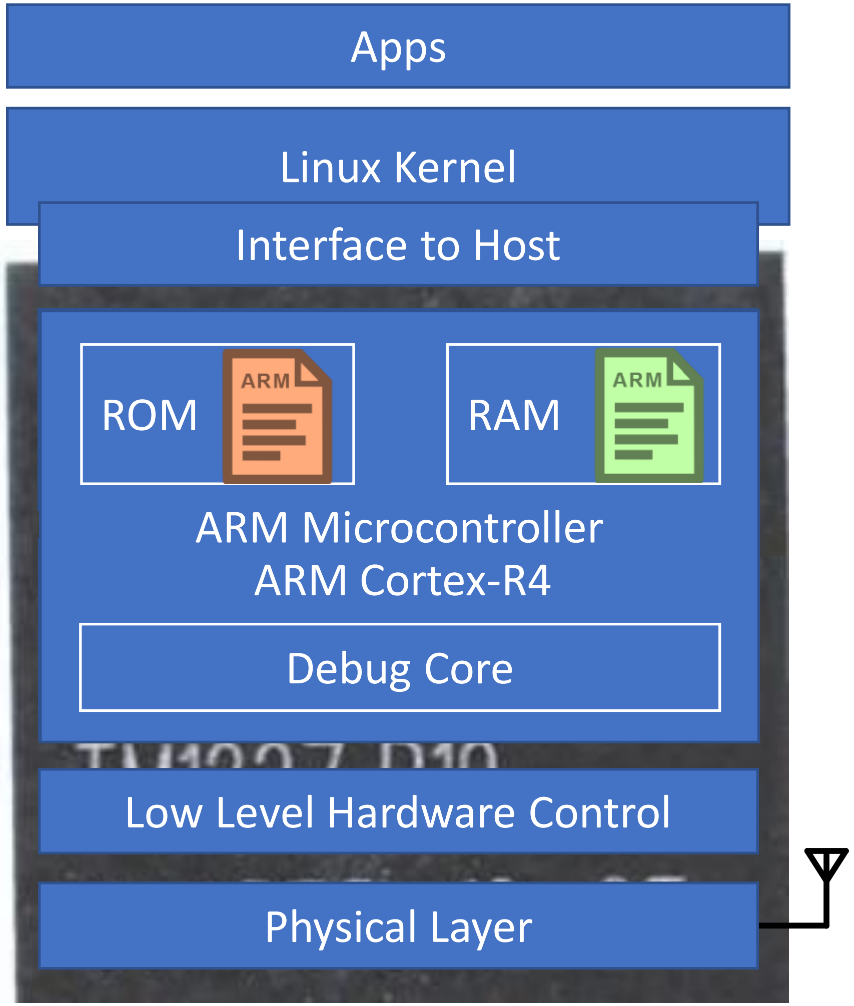


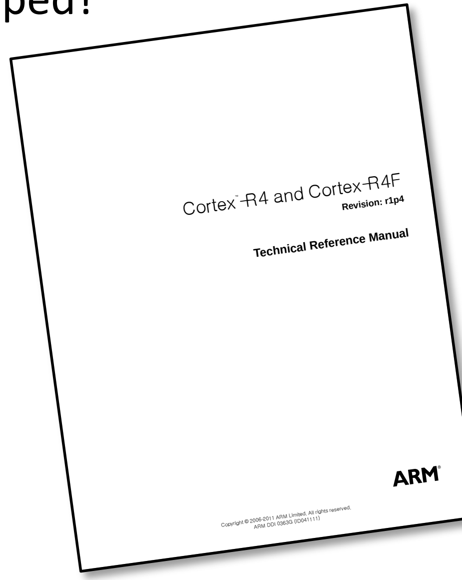
Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

Memory Map

To which addresses are the debugging registers mapped?





Cortex™-R4 and Cortex-R4F

Revision: r1p4

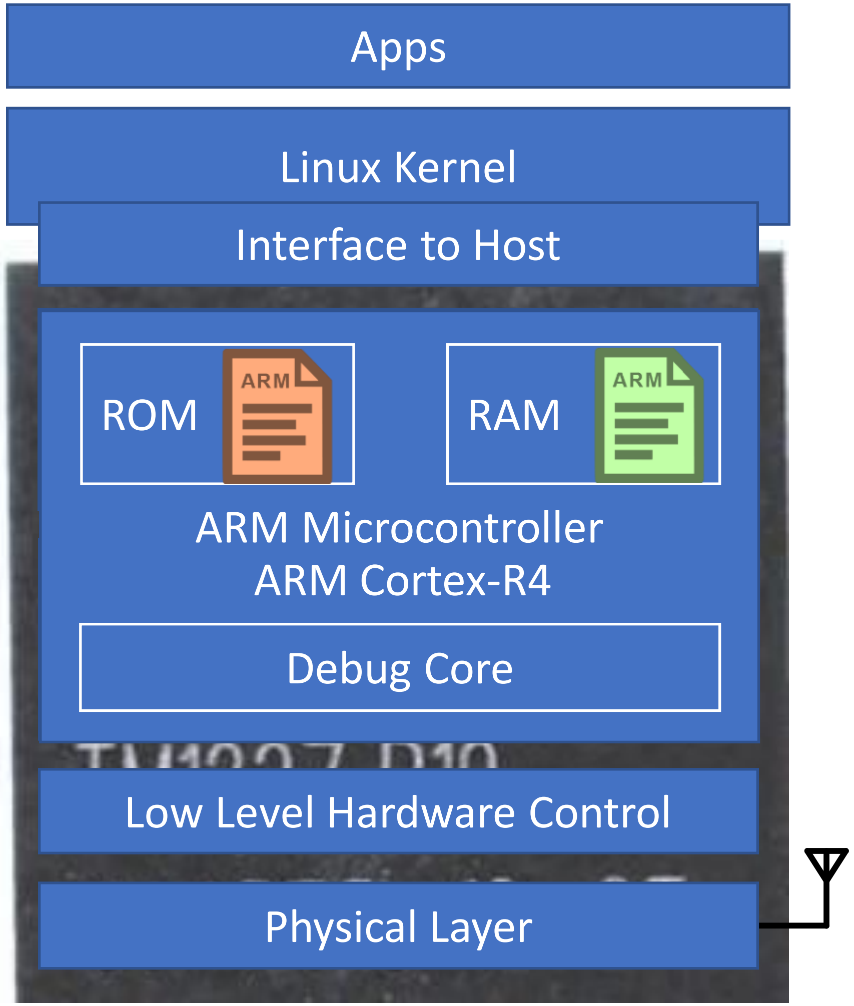
Technical Reference Manual

ARM®

Copyright © 2006-2011 ARM Limited. All rights reserved.
ARM DDI 0363G (ID041111)

Memory Map

To which addresses are the debugging registers mapped?



Apps

Linux Kernel

Interface to Host

ROM

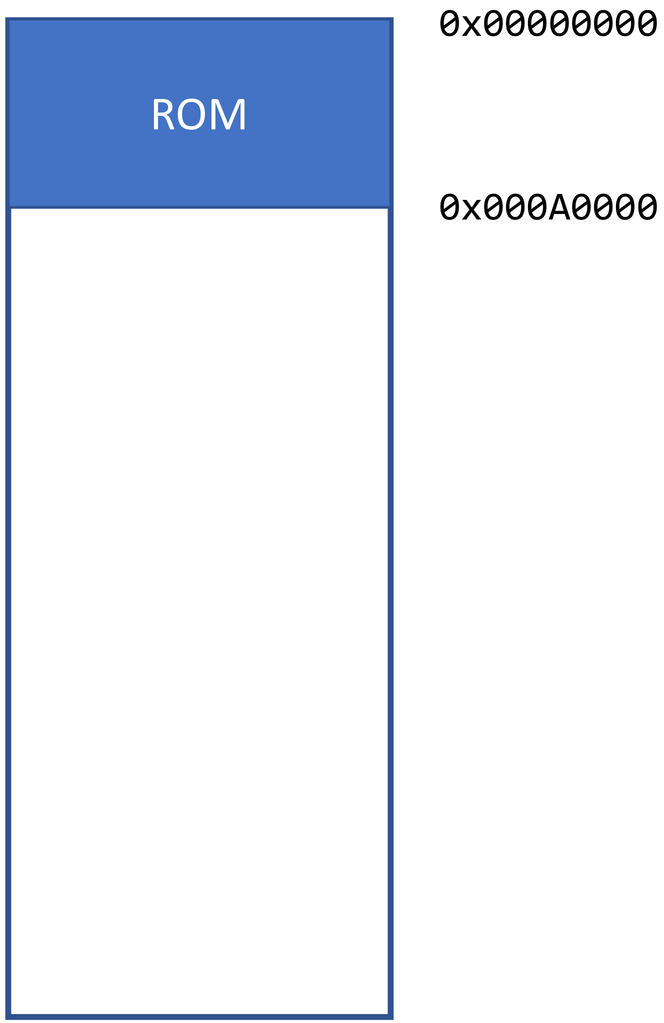
RAM

ARM Microcontroller
ARM Cortex-R4

Debug Core

Low Level Hardware Control

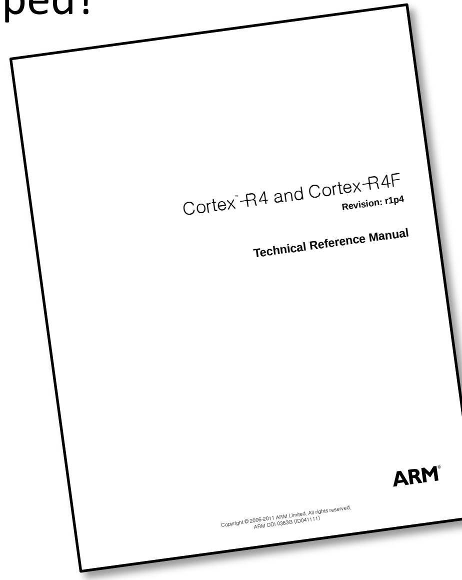
Physical Layer



ROM

0x00000000

0x000A0000



Cortex™-R4 and Cortex-R4F
Revision: r1p4

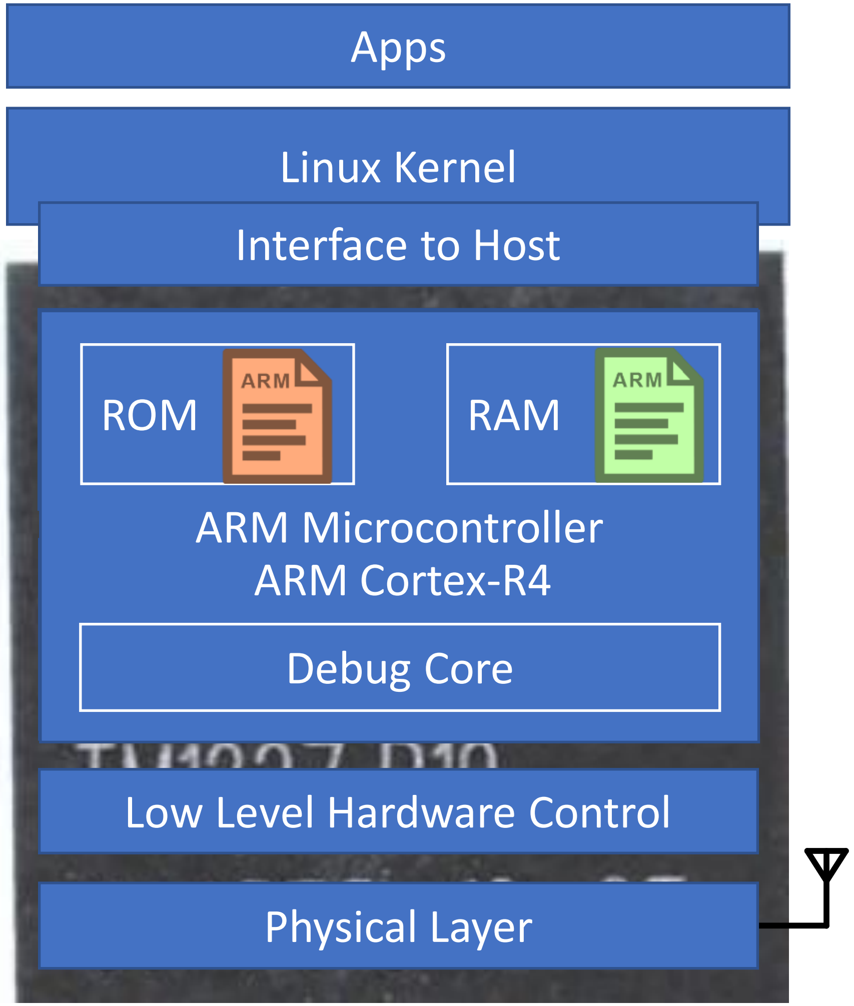
Technical Reference Manual

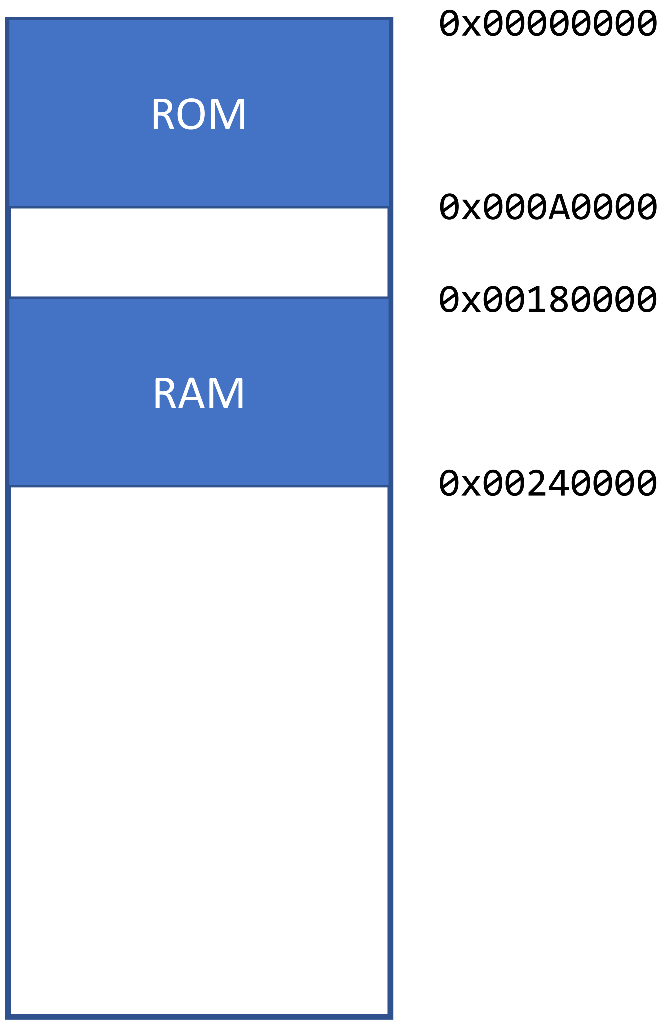
ARM®

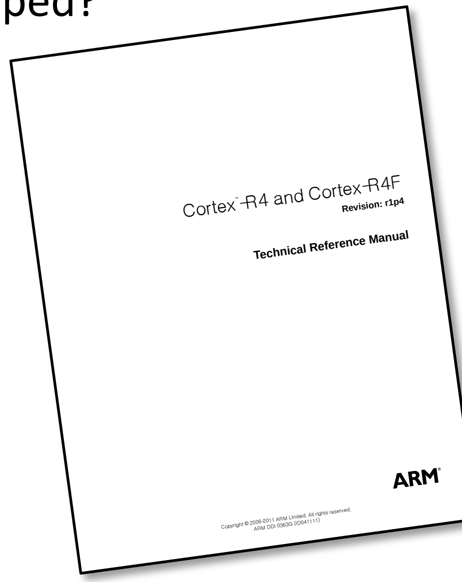
Copyright © 2006-2011 ARM Limited. All rights reserved.
ARM DDI 0363G (ID041111)

Memory Map

To which addresses are the debugging registers mapped?





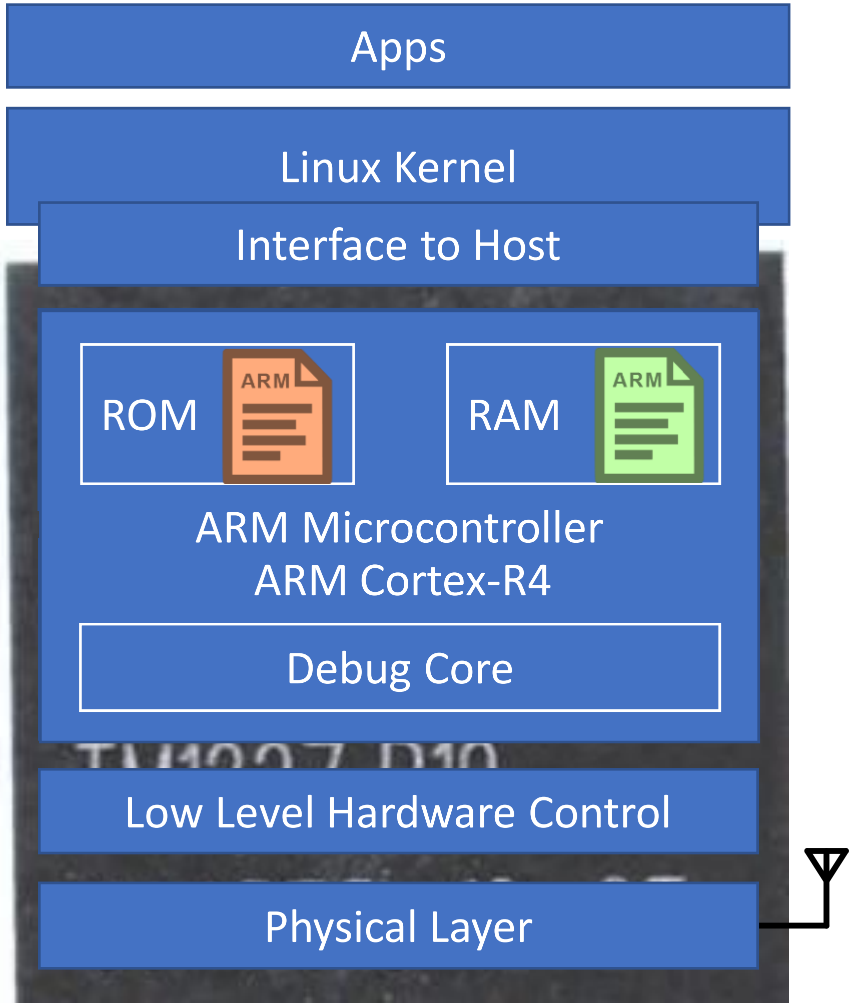

Cortex-R4 and Cortex-R4F
Revision: r1p4

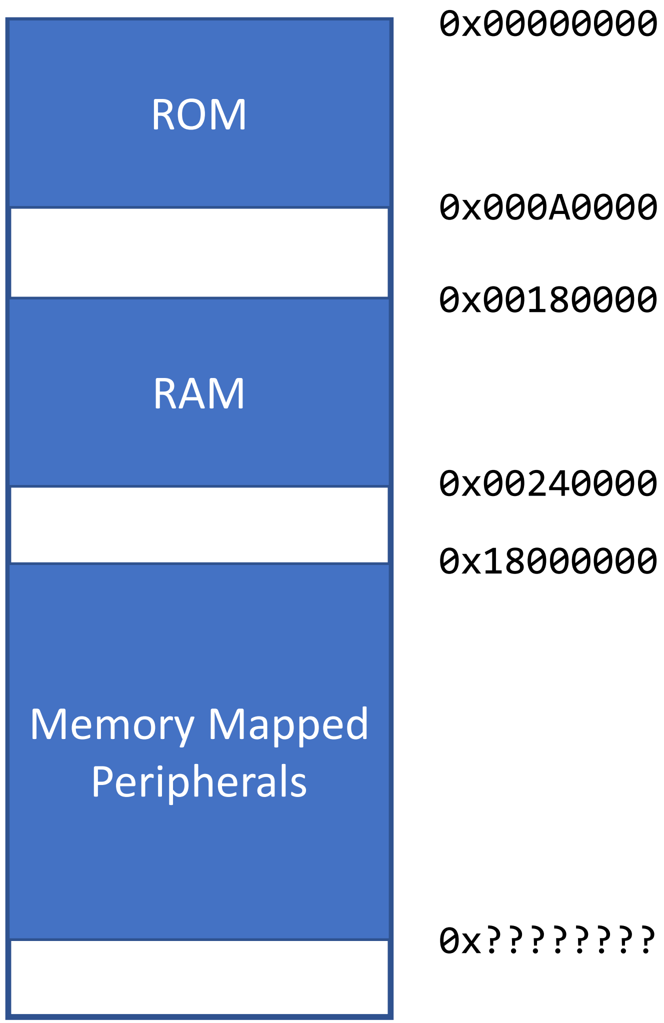
Technical Reference Manual

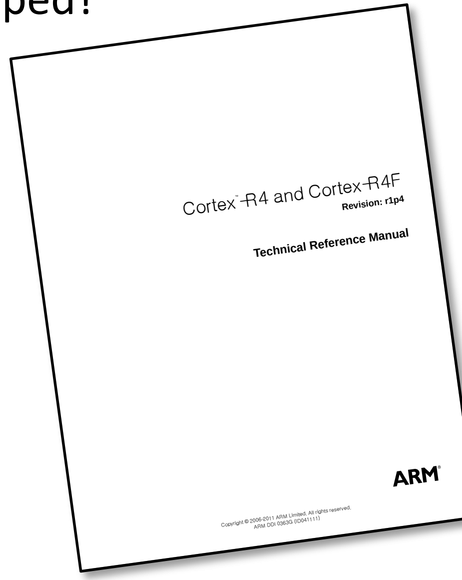
ARM

Memory Map

To which addresses are the debugging registers mapped?



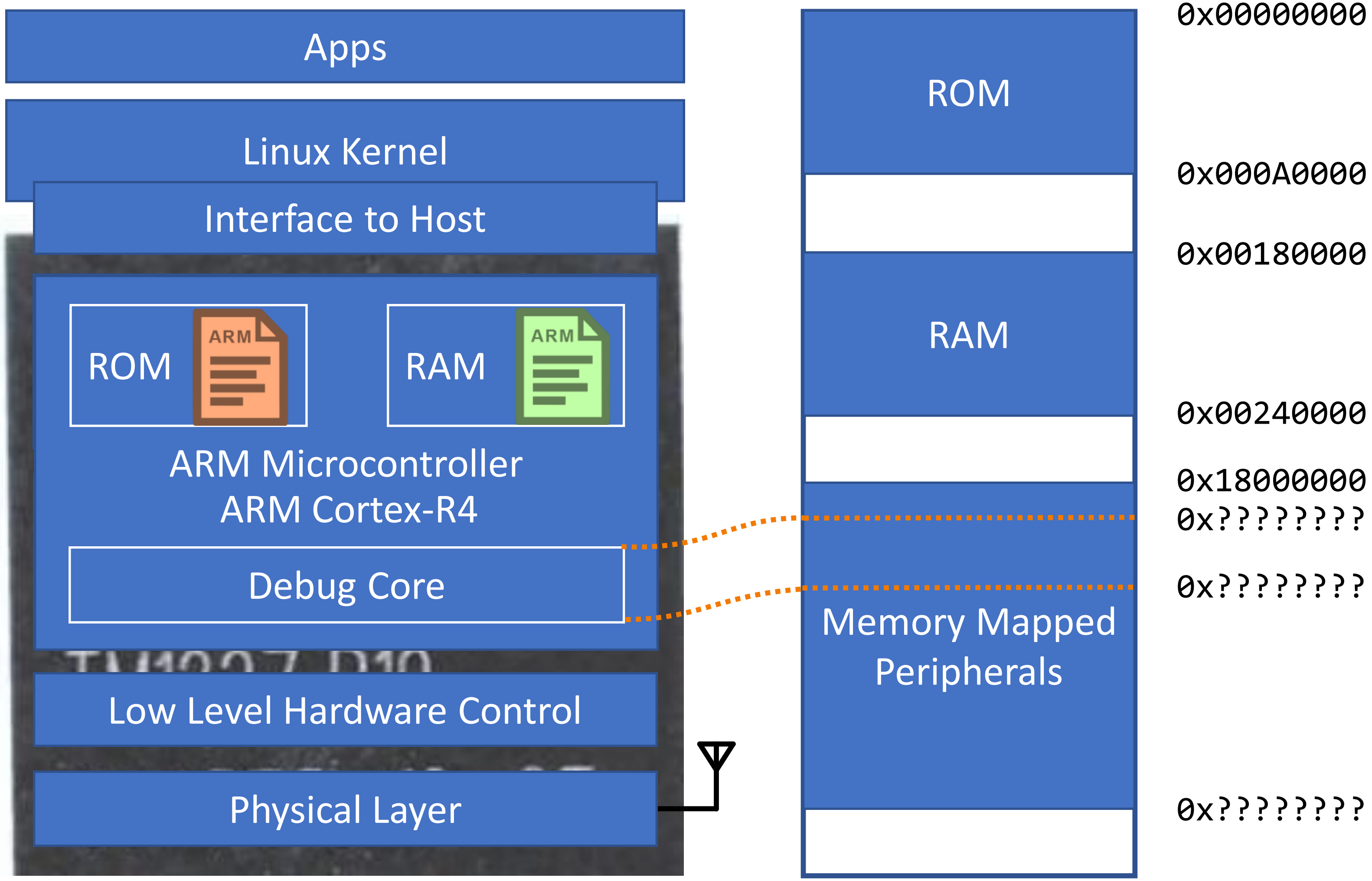




Cortex™-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM®

Memory Map

To which addresses are the debugging registers mapped?



Apps

Linux Kernel

Interface to Host

ROM

RAM

ARM Microcontroller
ARM Cortex-R4

Debug Core

Low Level Hardware Control

Physical Layer

ROM

RAM

Memory Mapped Peripherals

0x00000000

0x000A0000

0x00180000

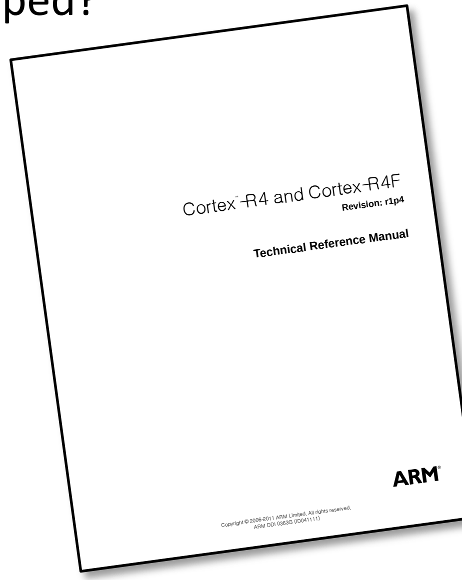
0x00240000

0x18000000

0x????????

0x????????

0x????????



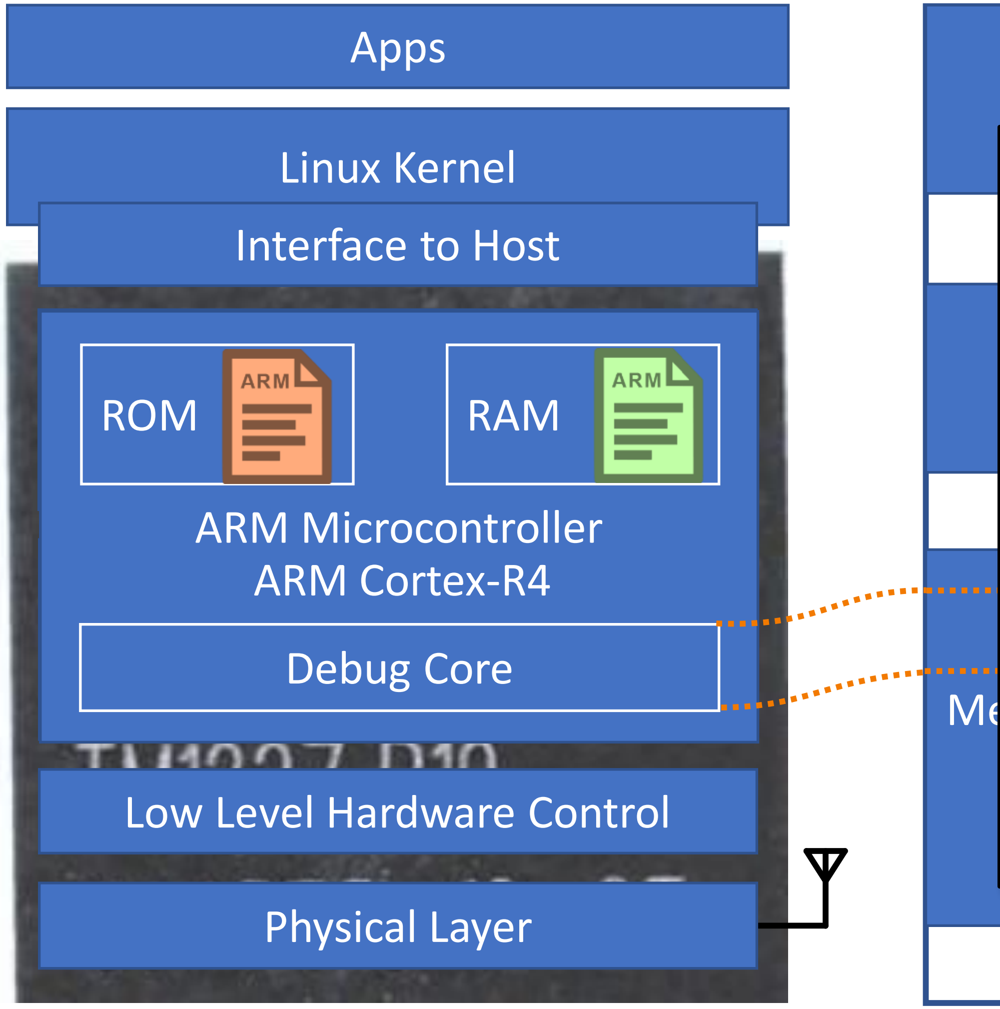
Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

Memory Map

To which addresses are the debugging registers mapped?



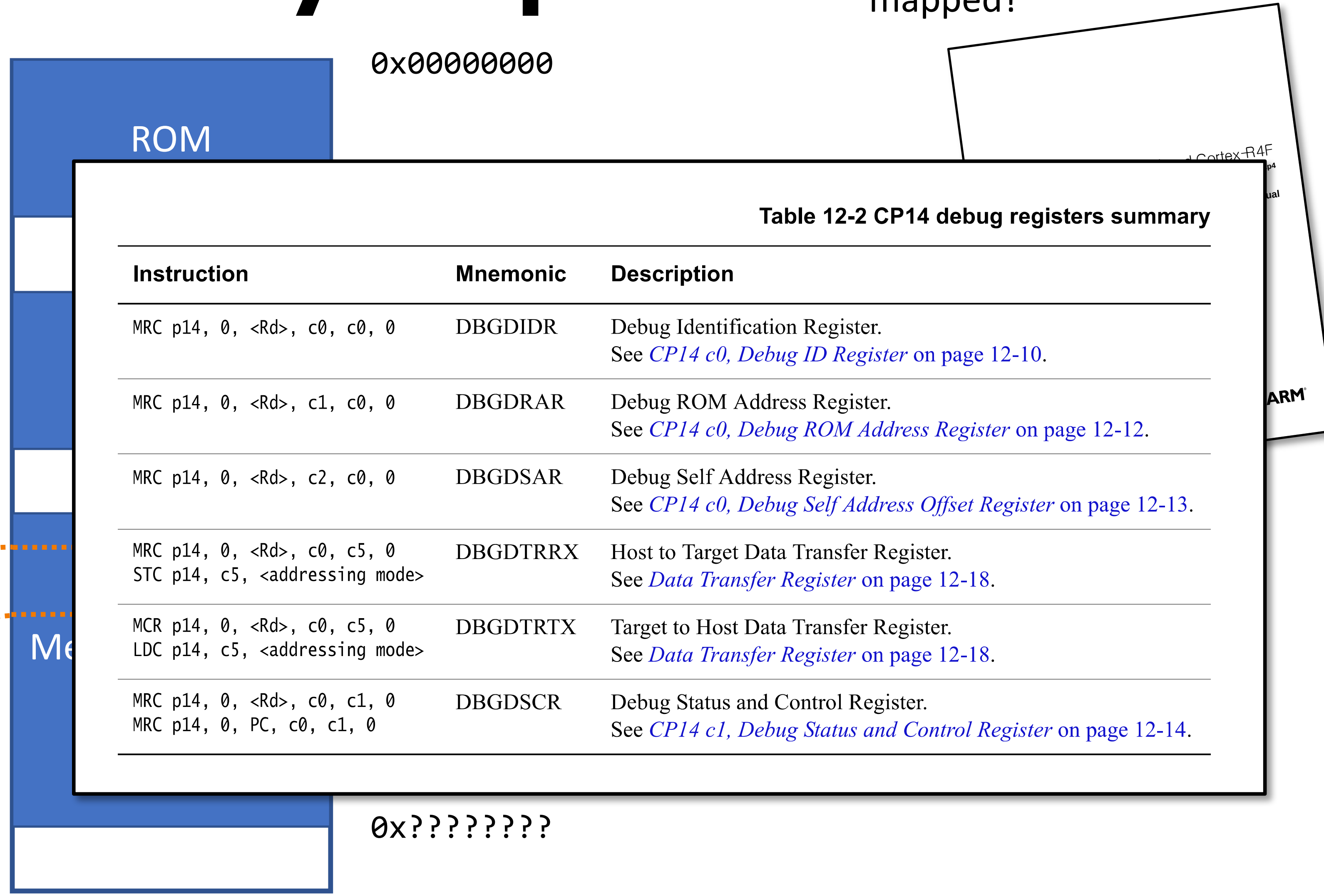
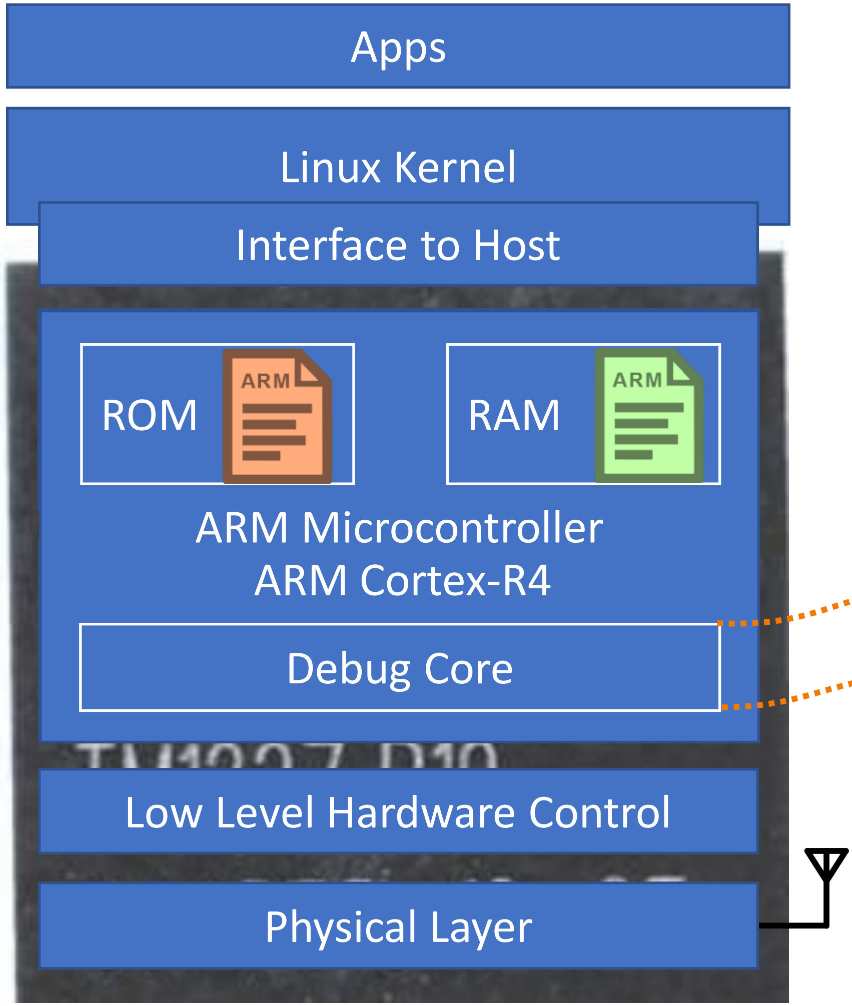


Table 12-2 CP14 debug registers summary

Instruction	Mnemonic	Description
`MRC p14, 0, <Rd>, c0, c0, 0`	DBGDIDR	Debug Identification Register. See *CP14 c0, Debug ID Register* on page 12-10.
`MRC p14, 0, <Rd>, c1, c0, 0`	DBGDRAR	Debug ROM Address Register. See *CP14 c0, Debug ROM Address Register* on page 12-12.
`MRC p14, 0, <Rd>, c2, c0, 0`	DBGDSAR	Debug Self Address Register. See *CP14 c0, Debug Self Address Offset Register* on page 12-13.
`MRC p14, 0, <Rd>, c0, c5, 0` `STC p14, c5, <addressing mode>`	DBGDTRRX	Host to Target Data Transfer Register. See *Data Transfer Register* on page 12-18.
`MCR p14, 0, <Rd>, c0, c5, 0` `LDC p14, c5, <addressing mode>`	DBGDTRTX	Target to Host Data Transfer Register. See *Data Transfer Register* on page 12-18.
`MRC p14, 0, <Rd>, c0, c1, 0` `MRC p14, 0, PC, c0, c1, 0`	DBGDSCR	Debug Status and Control Register. See *CP14 c1, Debug Status and Control Register* on page 12-14.

Memory Map

To which addresses are the debugging registers mapped?



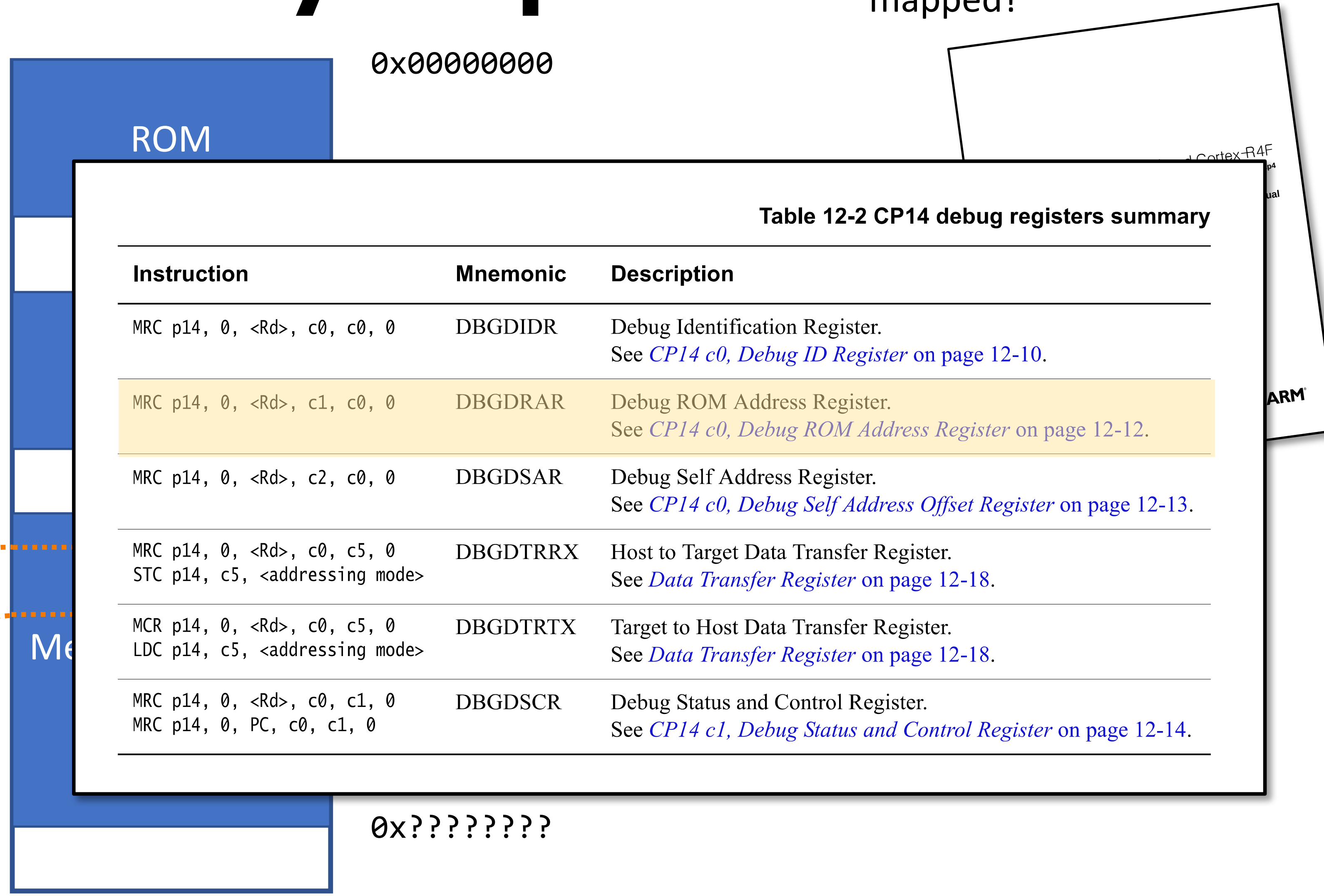
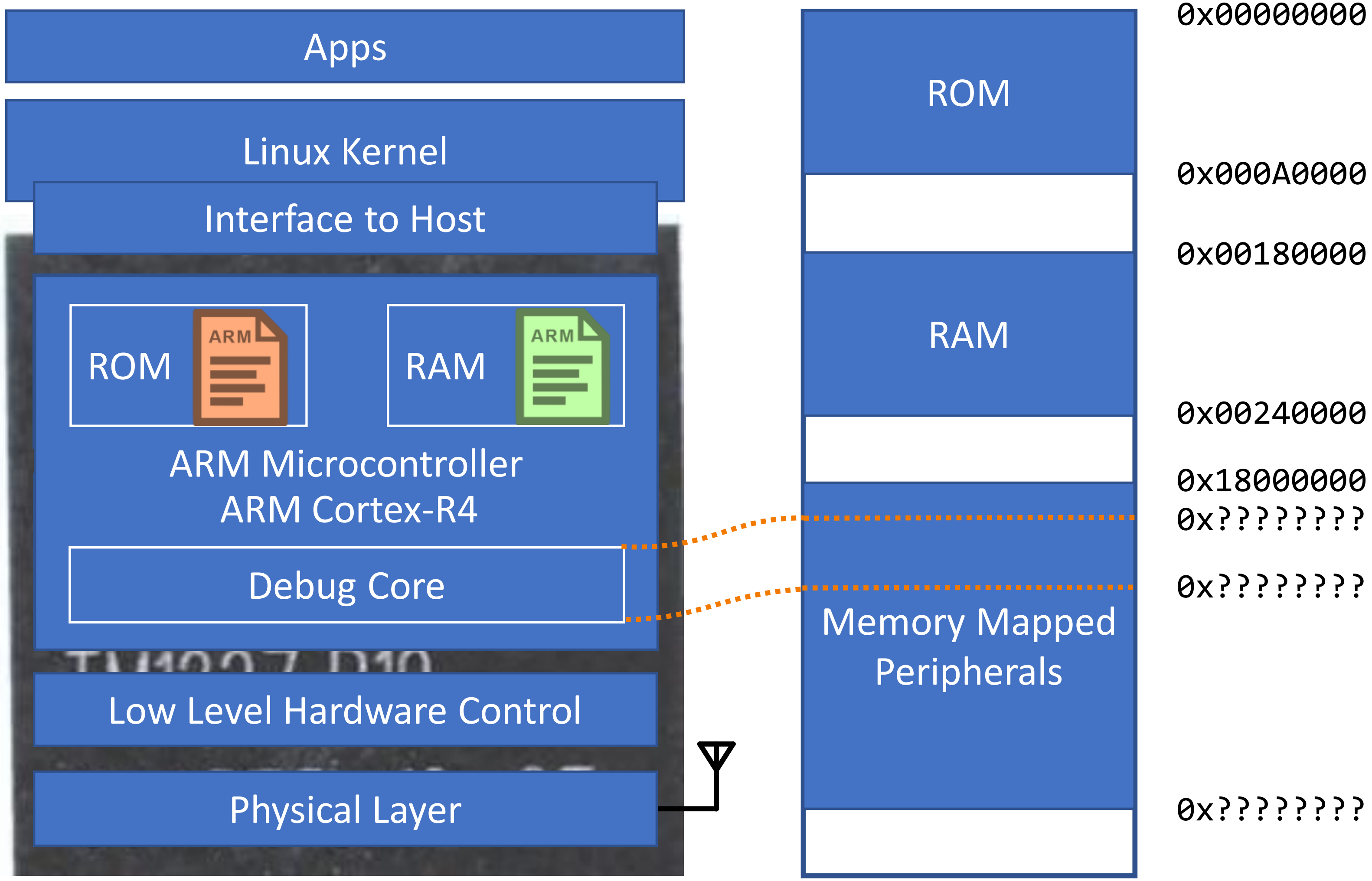


Table 12-2 CP14 debug registers summary

Instruction	Mnemonic	Description
`MRC p14, 0, <Rd>, c0, c0, 0`	DBGDIDR	Debug Identification Register. See *CP14 c0, Debug ID Register* on page 12-10.
`MRC p14, 0, <Rd>, c1, c0, 0`	DBGDRAR	Debug ROM Address Register. See *CP14 c0, Debug ROM Address Register* on page 12-12.
`MRC p14, 0, <Rd>, c2, c0, 0`	DBGDSAR	Debug Self Address Register. See *CP14 c0, Debug Self Address Offset Register* on page 12-13.
`MRC p14, 0, <Rd>, c0, c5, 0` `STC p14, c5, <addressing mode>`	DBGDTRRX	Host to Target Data Transfer Register. See *Data Transfer Register* on page 12-18.
`MCR p14, 0, <Rd>, c0, c5, 0` `LDC p14, c5, <addressing mode>`	DBGDTRTX	Target to Host Data Transfer Register. See *Data Transfer Register* on page 12-18.
`MRC p14, 0, <Rd>, c0, c1, 0` `MRC p14, 0, PC, c0, c1, 0`	DBGDSCR	Debug Status and Control Register. See *CP14 c1, Debug Status and Control Register* on page 12-14.

Memory Map

To which addresses are the debugging registers mapped?



Apps

Linux Kernel

Interface to Host

ROM ARM

RAM ARM

ARM Microcontroller
ARM Cortex-R4

Debug Core

Low Level Hardware Control

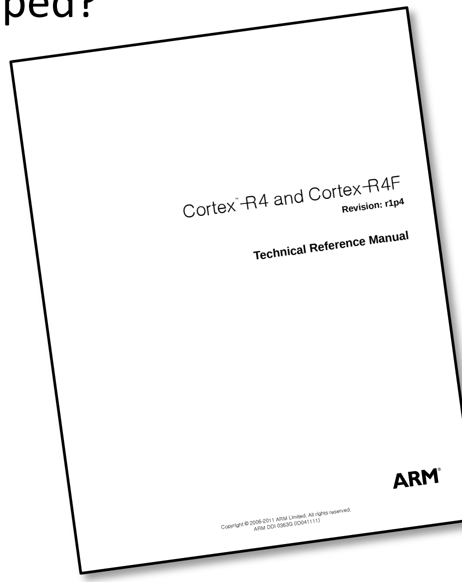
Physical Layer

ROM

RAM

Memory Mapped Peripherals

0x00000000
0x000A0000
0x00180000
0x00240000
0x18000000
0x????????
0x????????
0x????????



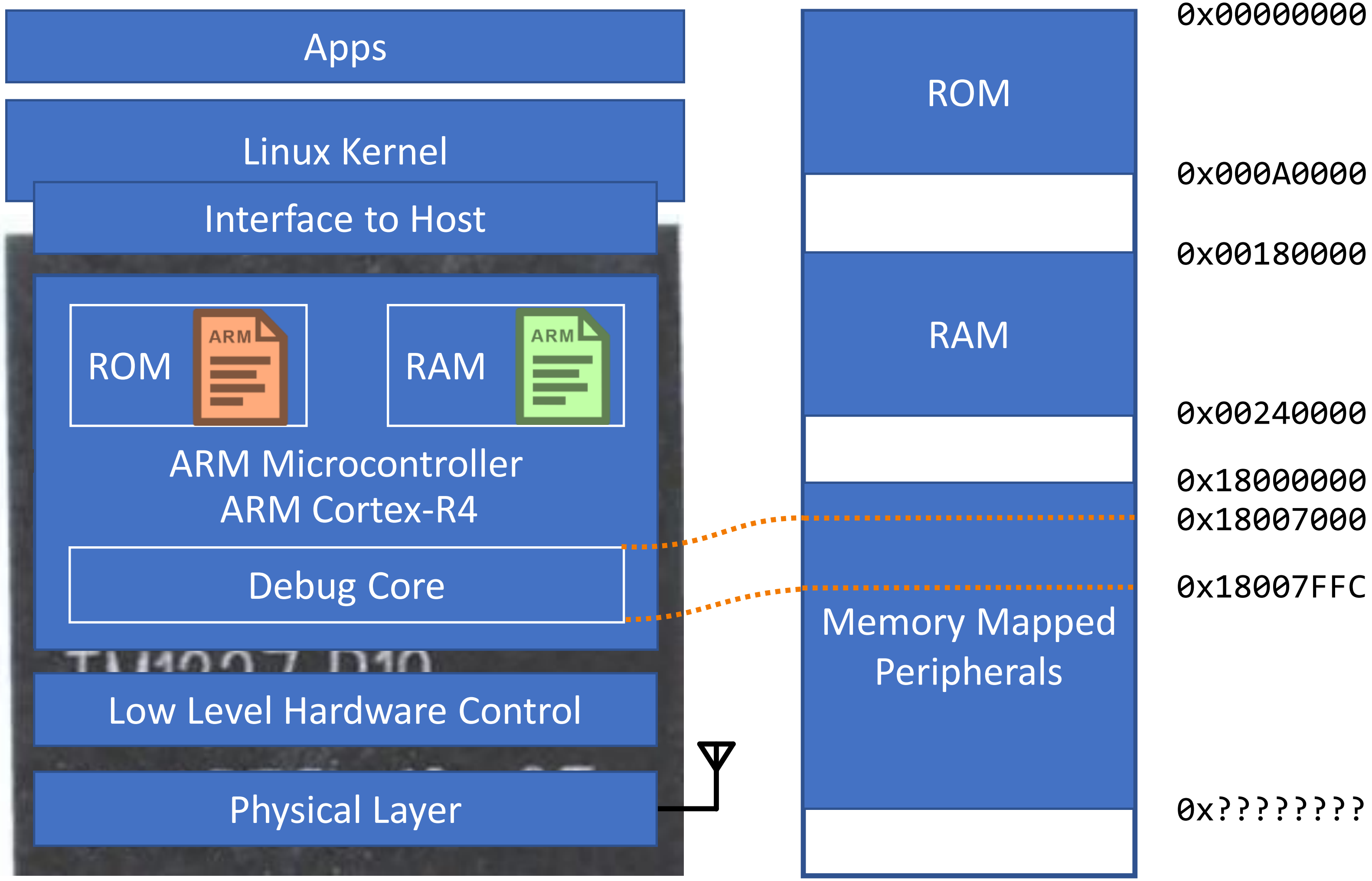
Cortex-R4 and Cortex-R4F
Revision: r1p4

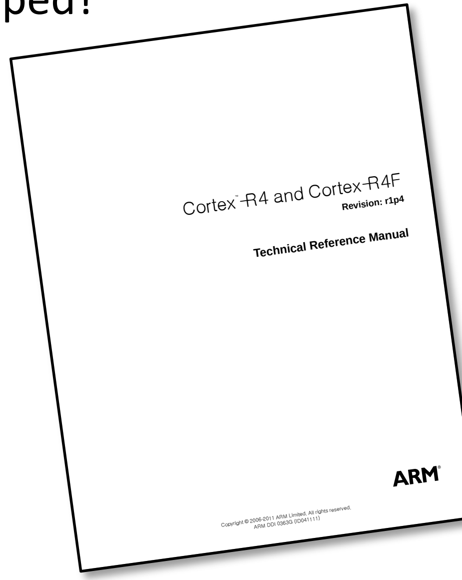
Technical Reference Manual

ARM

Memory Map

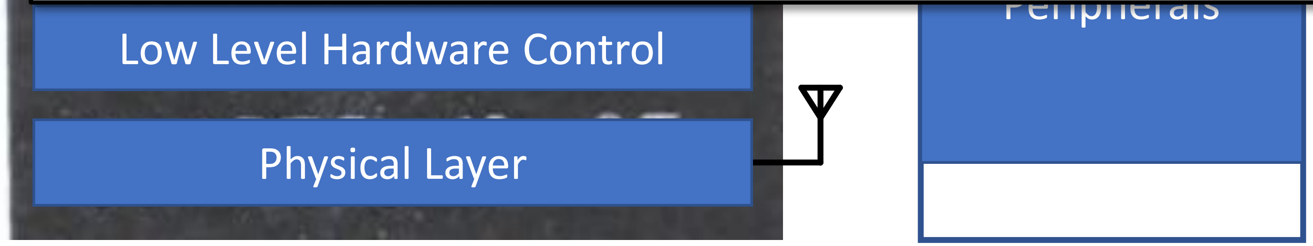
✓ To which addresses are the debugging registers mapped?

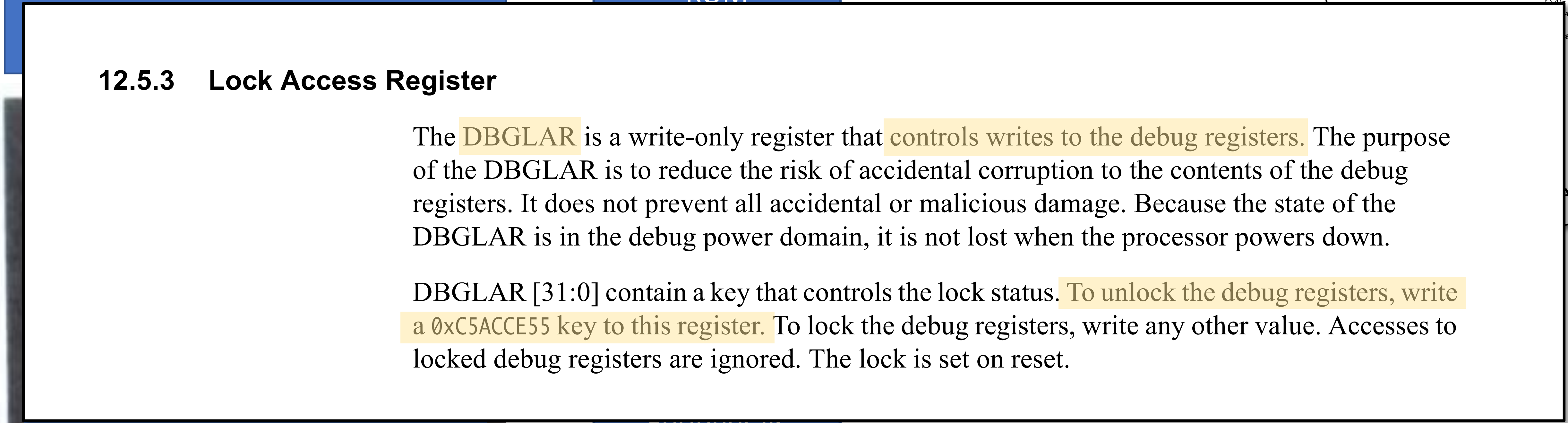



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

Memory Map

✓ To which addresses are the debugging registers mapped?





12.5.3 Lock Access Register

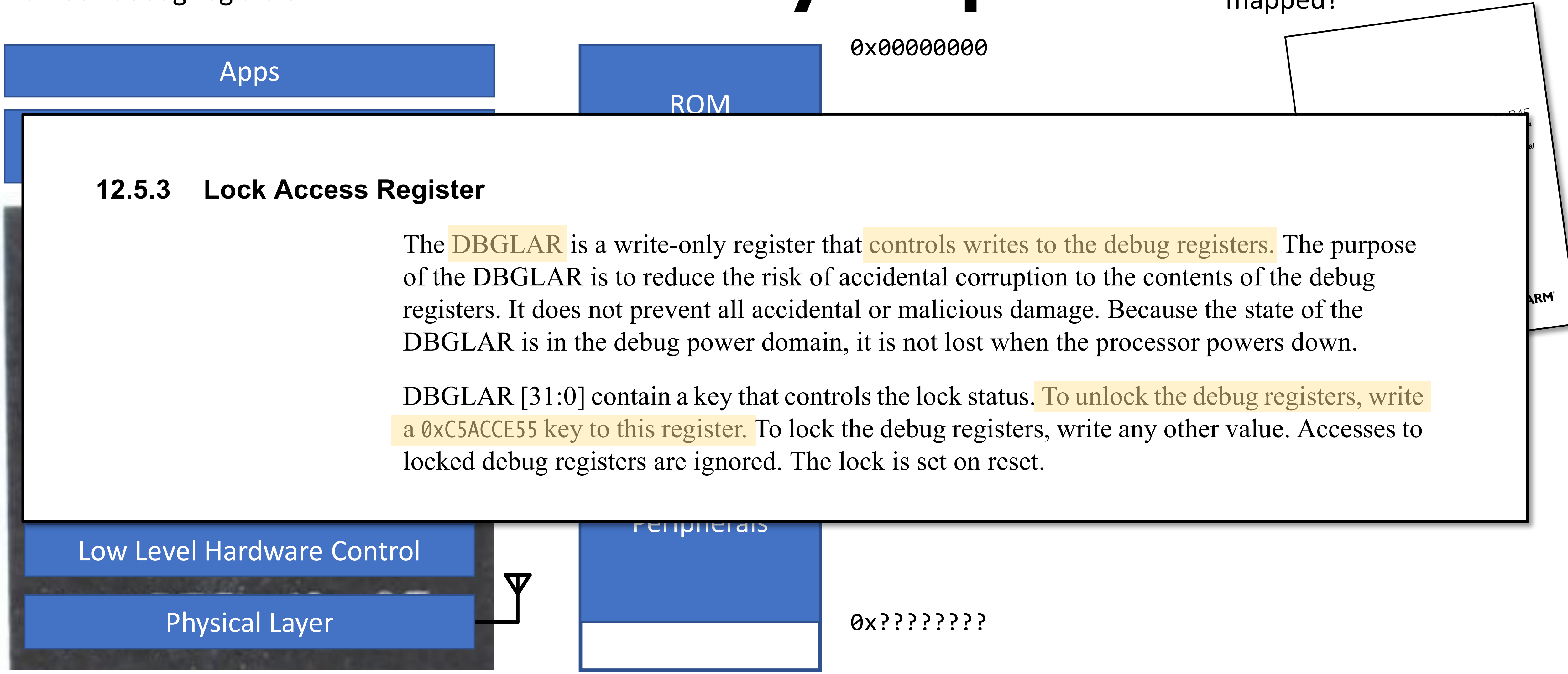
The DBGLAR is a write-only register that controls writes to the debug registers. The purpose of the DBGLAR is to reduce the risk of accidental corruption to the contents of the debug registers. It does not prevent all accidental or malicious damage. Because the state of the DBGLAR is in the debug power domain, it is not lost when the processor powers down.

DBGLAR [31:0] contain a key that controls the lock status. To unlock the debug registers, write a `0xC5ACCE55` key to this register. To lock the debug registers, write any other value. Accesses to locked debug registers are ignored. The lock is set on reset.

Reminder: We need to unlock debug registers!

Memory Map

✓ To which addresses are the debugging registers mapped?



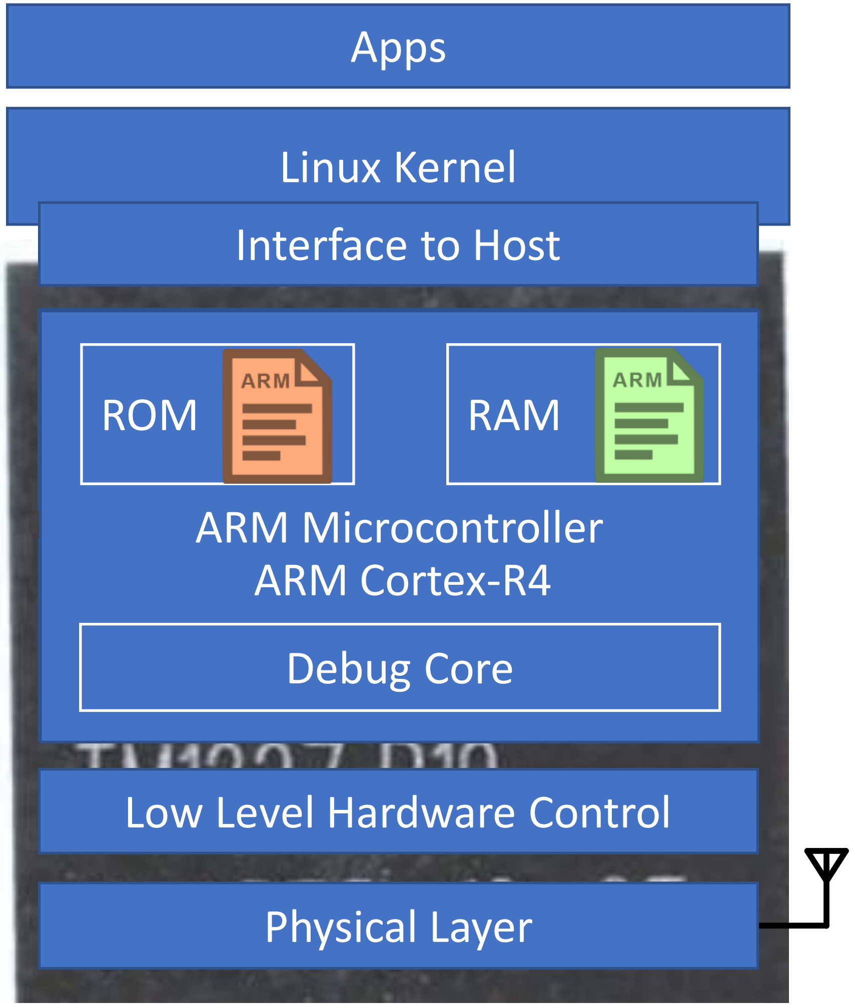
12.5.3 Lock Access Register

The DBGLAR is a write-only register that controls writes to the debug registers. The purpose of the DBGLAR is to reduce the risk of accidental corruption to the contents of the debug registers. It does not prevent all accidental or malicious damage. Because the state of the DBGLAR is in the debug power domain, it is not lost when the processor powers down.

DBGLAR [31:0] contain a key that controls the lock status. To unlock the debug registers, write a `0xC5ACCE55` key to this register. To lock the debug registers, write any other value. Accesses to locked debug registers are ignored. The lock is set on reset.

Reminder: We need to unlock debug registers!

Accessing Debug Core


Apps

Linux Kernel

Interface to Host

ROM

ARM

RAM

ARM

ARM Microcontroller
ARM Cortex-R4

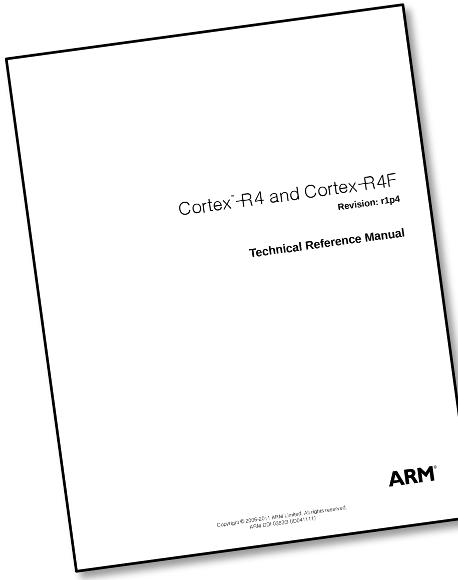
Debug Core

Low Level Hardware Control

Physical Layer

Firmware Execution

Events


Cortex-R4 and Cortex-R4F
Revision: r1p4

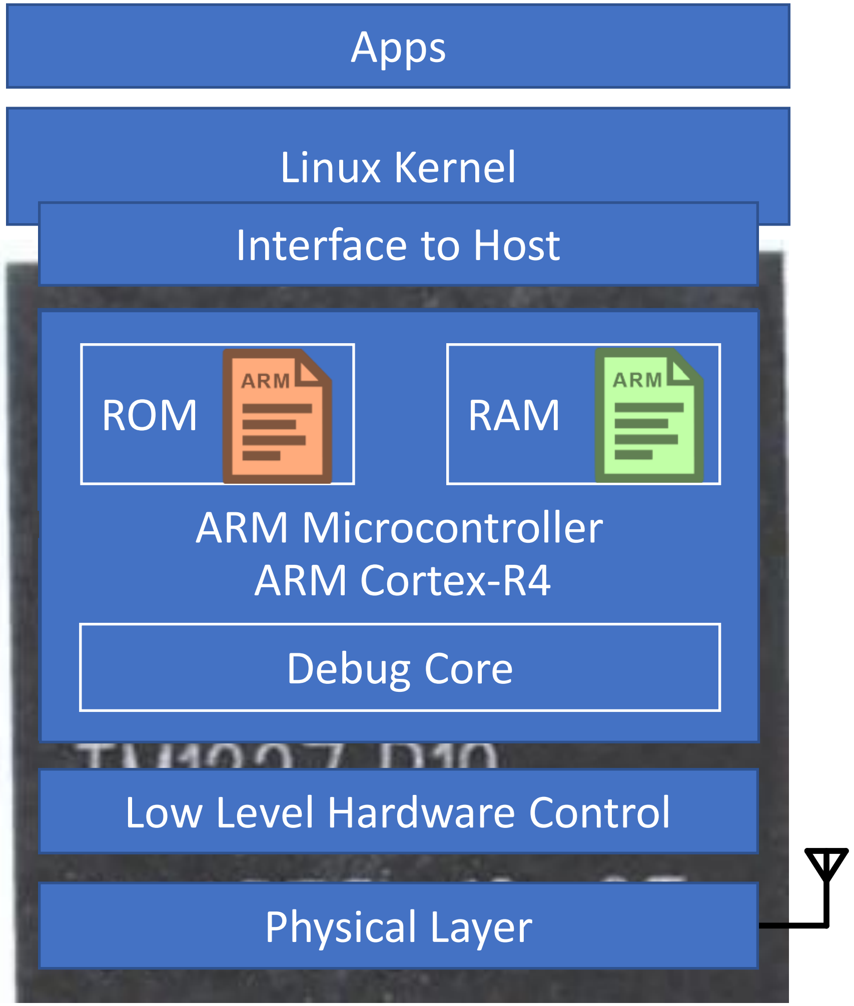
Technical Reference Manual

ARM

Copyright © 2006-2011 ARM Limited. All rights reserved.
ARM DDI 0363G (ID041111)

Reminder: We need to unlock debug registers!

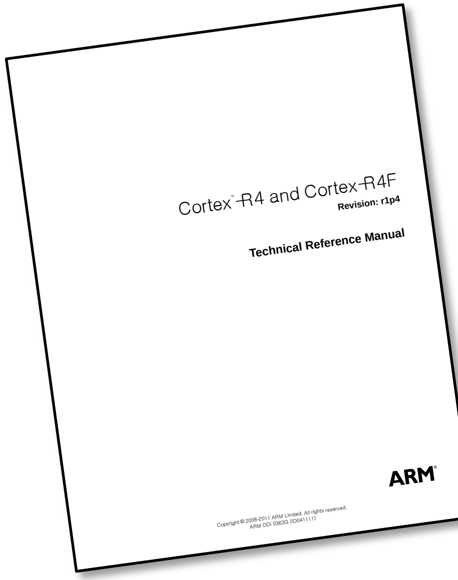
Accessing Debug Core



Firmware Execution

Initialize Hardware

Events

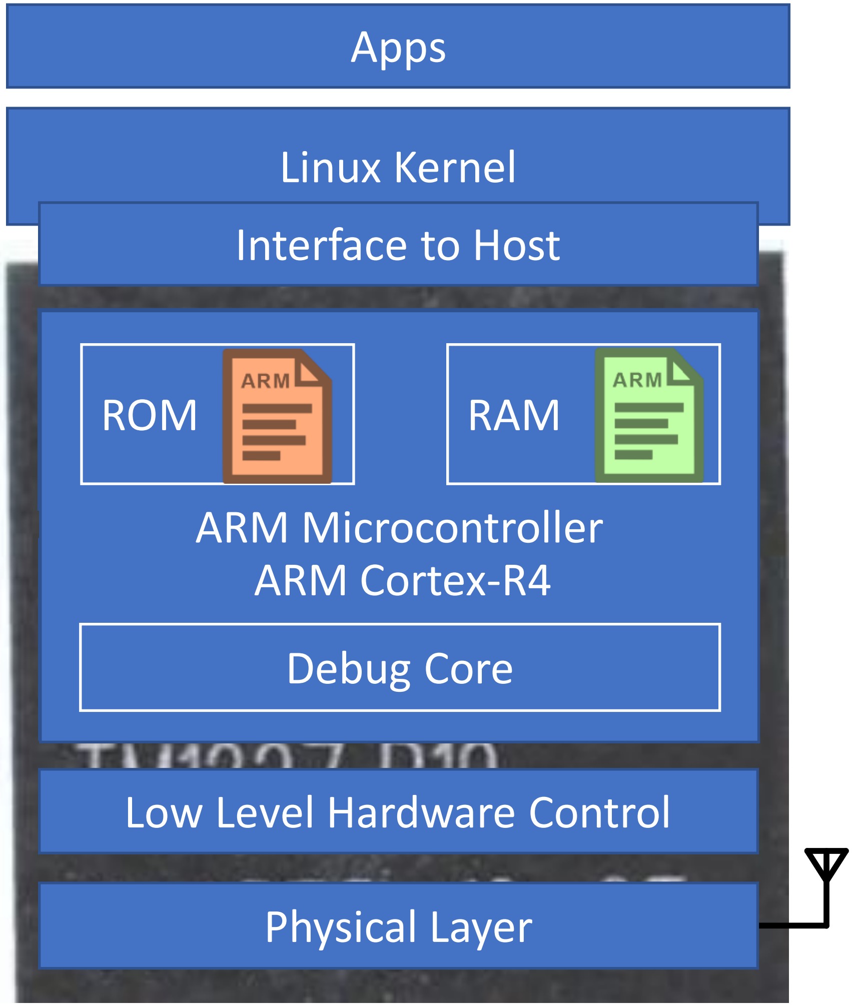

Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

Reminder: We need to unlock debug registers!

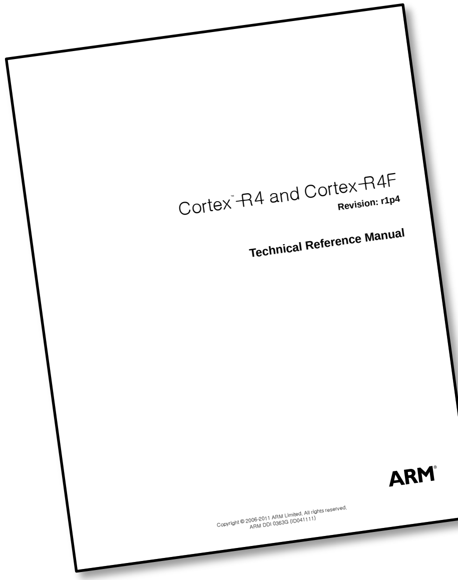
Accessing Debug Core



Firmware Execution

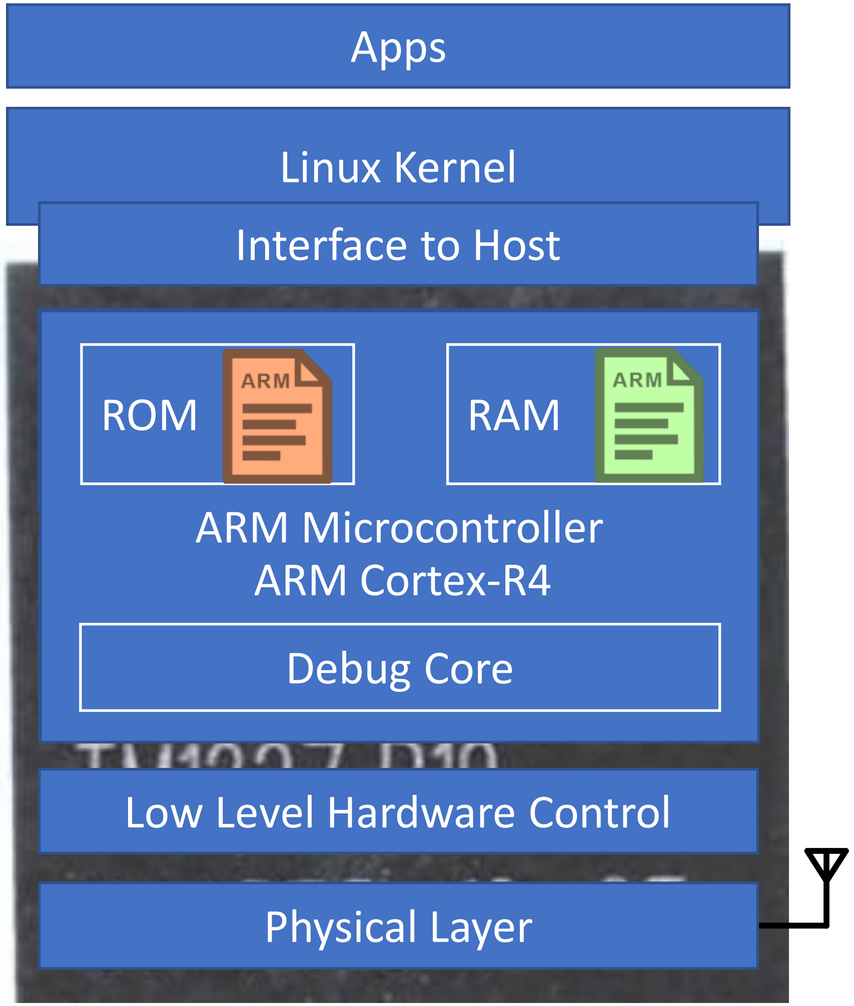
- Initialize Hardware
- Enable Interrupts

Events


Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

Reminder: We need to unlock debug registers!

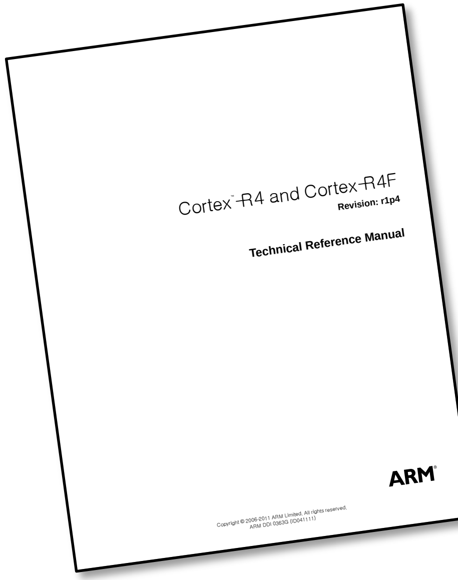
Accessing Debug Core



Firmware Execution

- Initialize Hardware
- Enable Interrupts
- Go to Sleep
  → Wait for Interrupts

Events


Cortex™-R4 and Cortex-R4F
Revision: r1p4

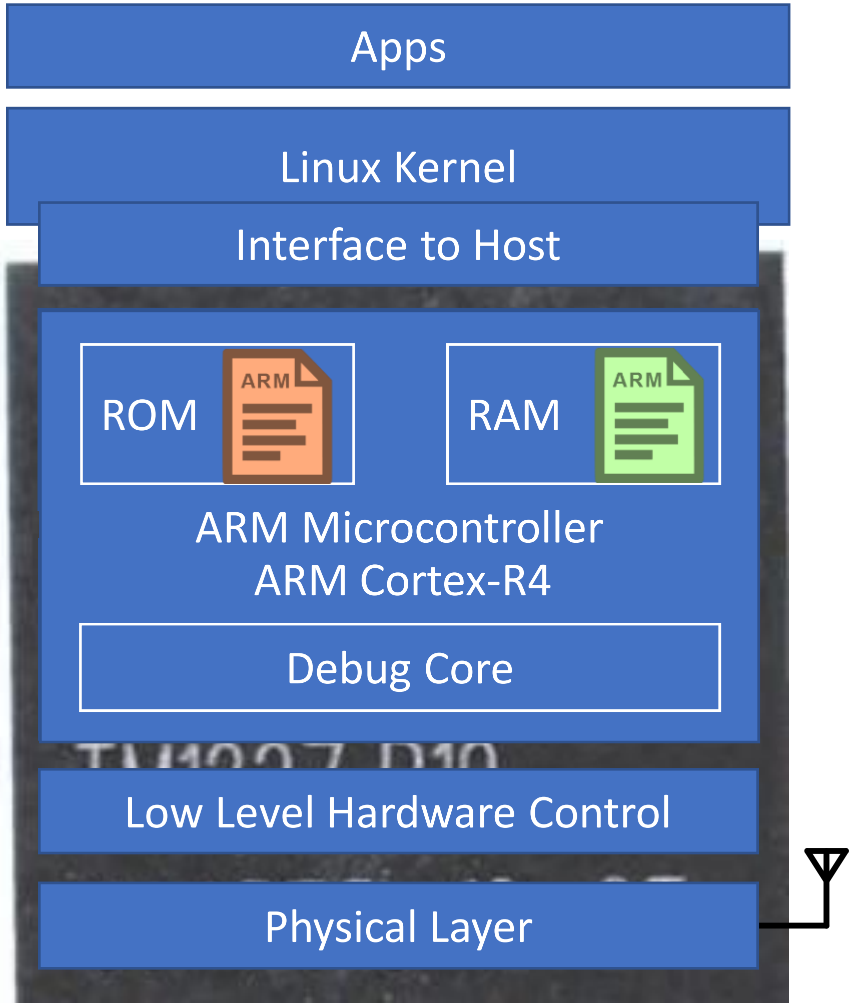
Technical Reference Manual

ARM®

Copyright © 2006-2011 ARM Limited. All rights reserved.
ARM DDI 0363G (ID041111)

Reminder: We need to unlock debug registers!

Accessing Debug Core

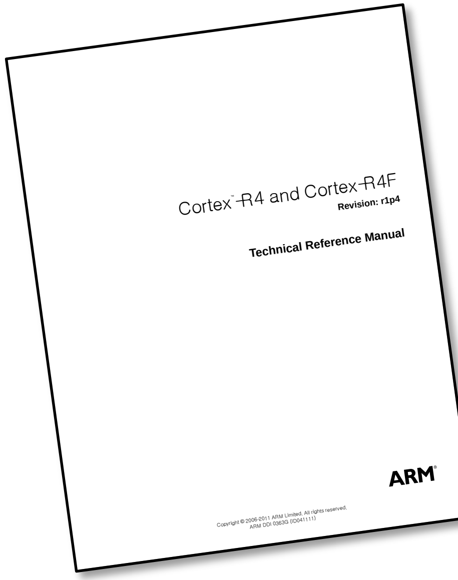


Firmware Execution

- Initialize Hardware
- Enable Interrupts
- Go to Sleep
  → Wait for Interrupts

Events

← Send IOCTL

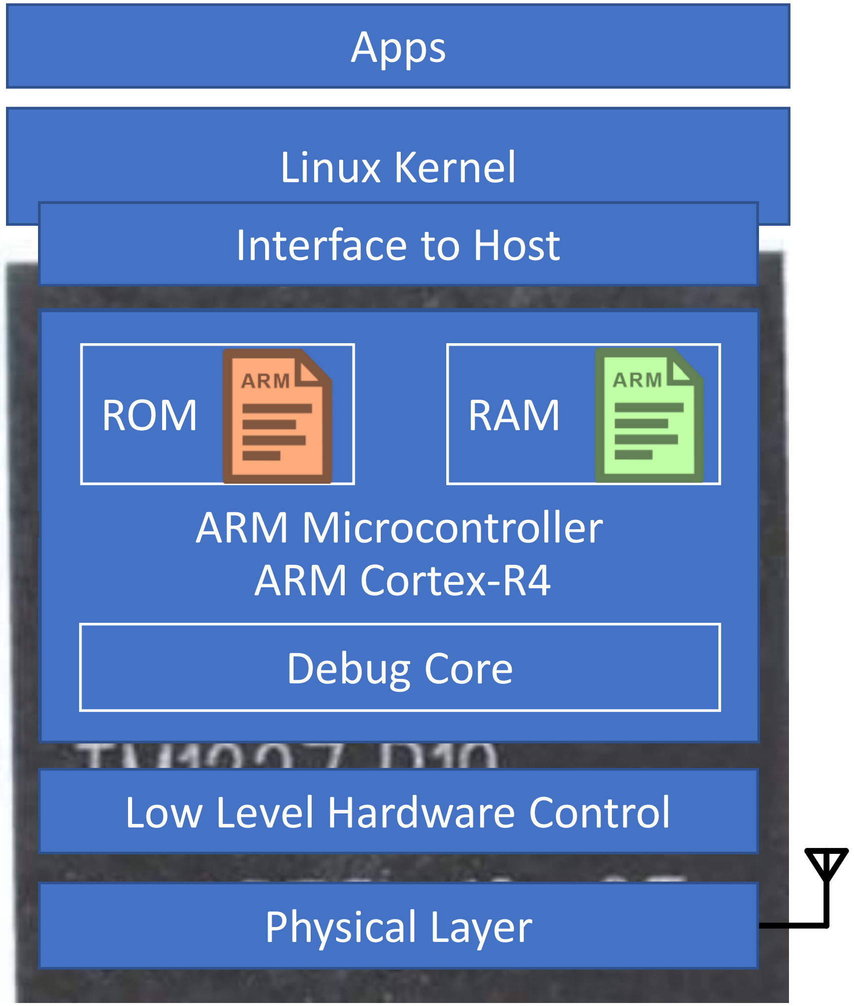

Cortex™-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM®

Reminder: We need to unlock debug registers!

Accessing Debug Core

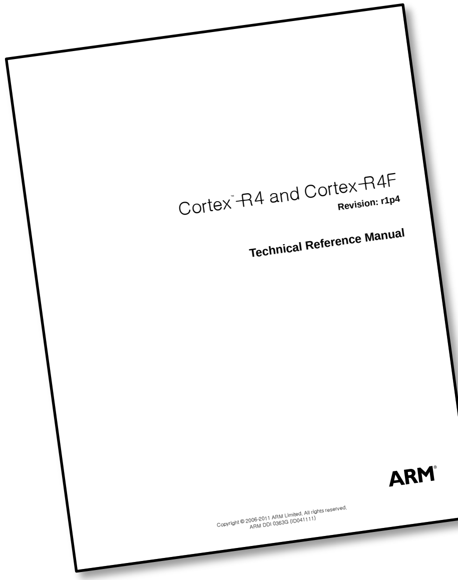


Firmware Execution

- Initialize Hardware
- Enable Interrupts
- Go to Sleep → Wait for Interrupts
- Wake up to handle IOCTL

Events

- ← Send IOCTL

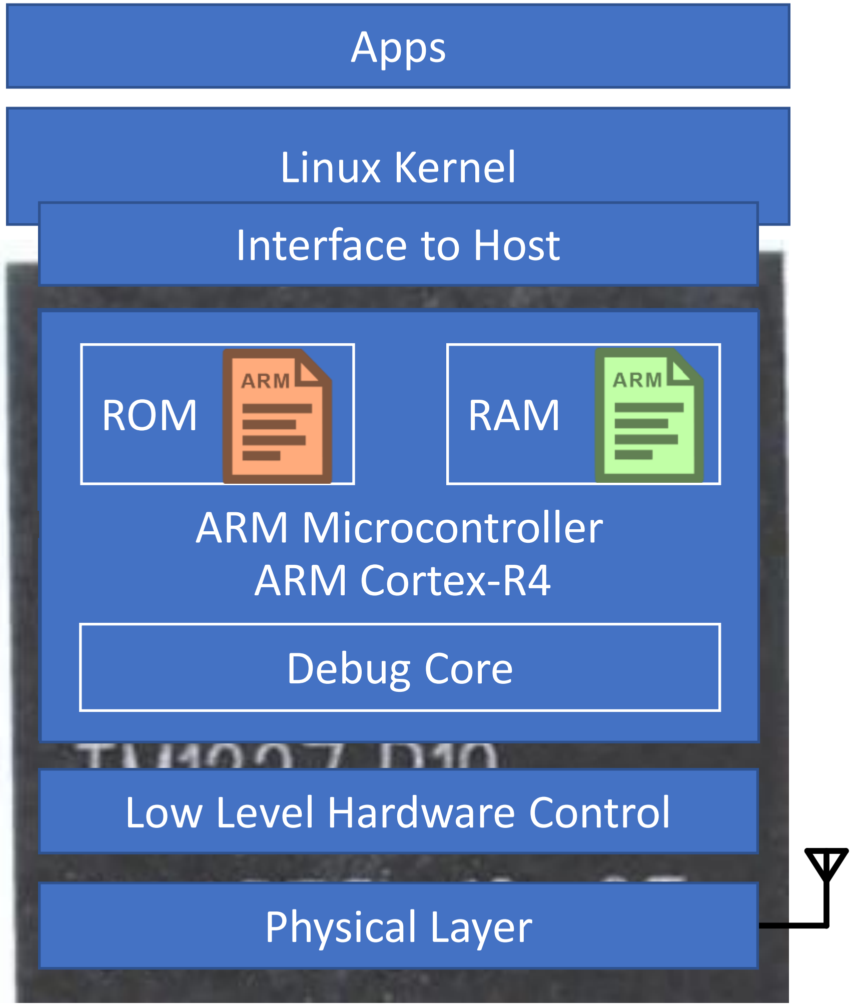

Cortex™-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM®

Reminder: We need to unlock debug registers!

Accessing Debug Core



Apps

Linux Kernel

Interface to Host

ROM

RAM

ARM Microcontroller
ARM Cortex-R4

Debug Core

Low Level Hardware Control

Physical Layer

Firmware Execution

Initialize Hardware

Enable Interrupts

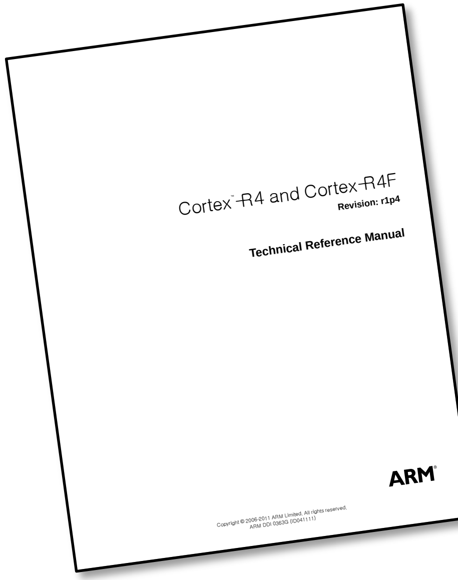
Go to Sleep
→ Wait for Interrupts

Wake up to handle IOCTL

Events

← Send IOCTL

→ Handle Response



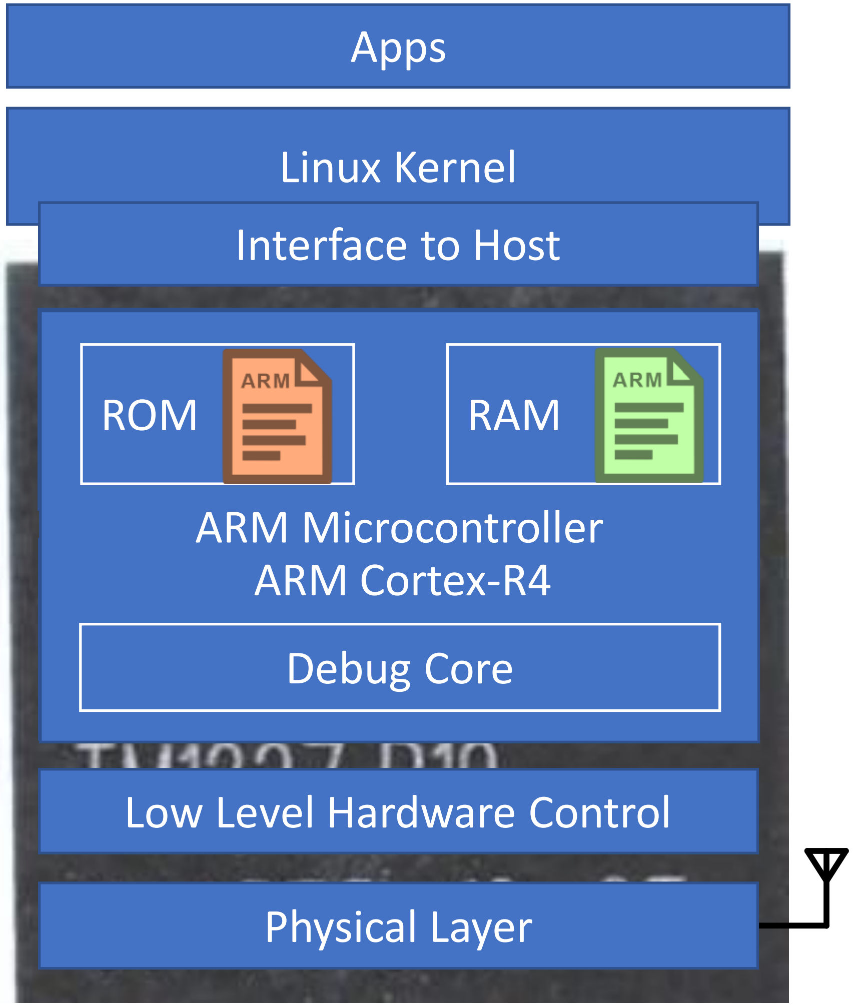
Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

Reminder: We need to unlock debug registers!

Accessing Debug Core



Firmware Execution

Initialize Hardware

Enable Interrupts

Go to Sleep
→ Wait for Interrupts

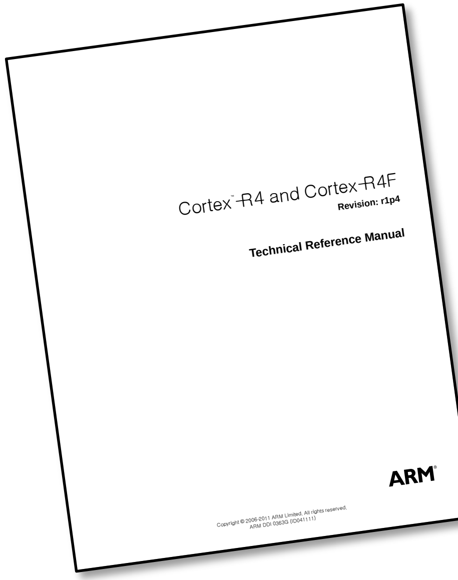
Wake up to handle IOCTL

Events

← Send IOCTL

→ Handle Response

← Send IOCTL



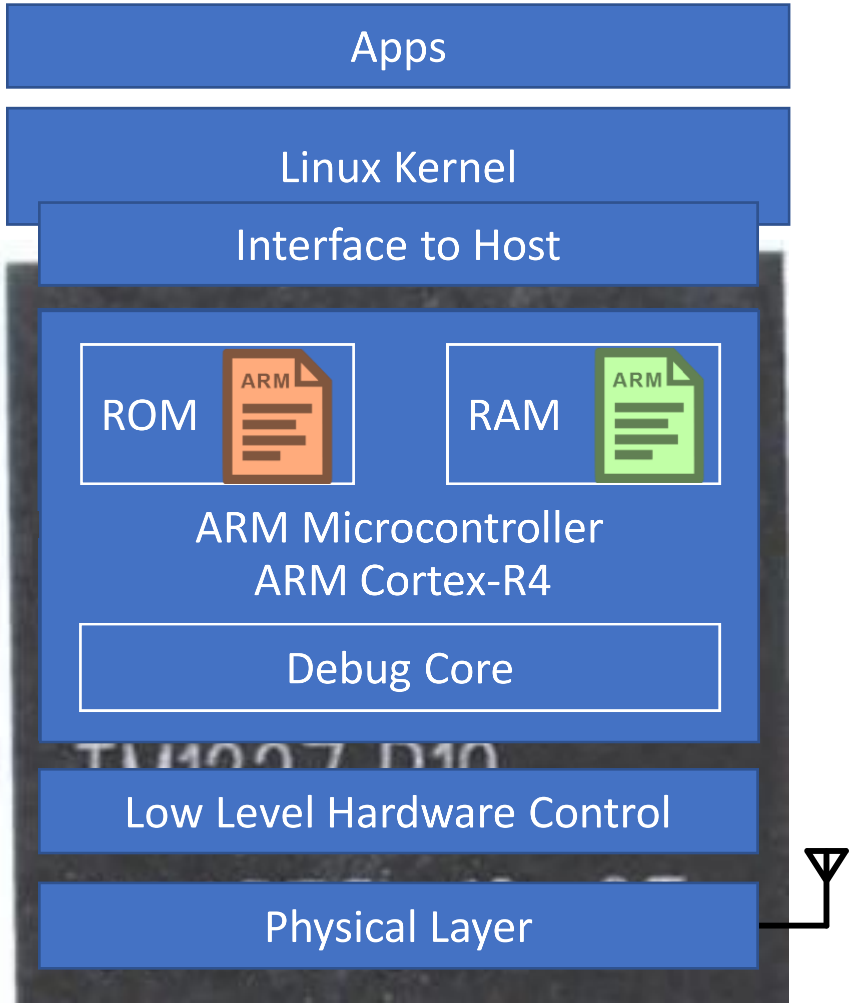
Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

Reminder: We need to unlock debug registers!

Accessing Debug Core



Firmware Execution

Initialize Hardware

Enable Interrupts

Go to Sleep
→ Wait for Interrupts

Wake up to handle IOCTL

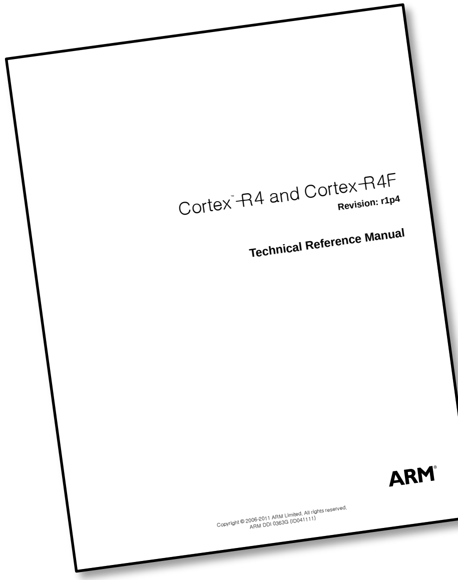
Wake up to handle IOCTL:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Events

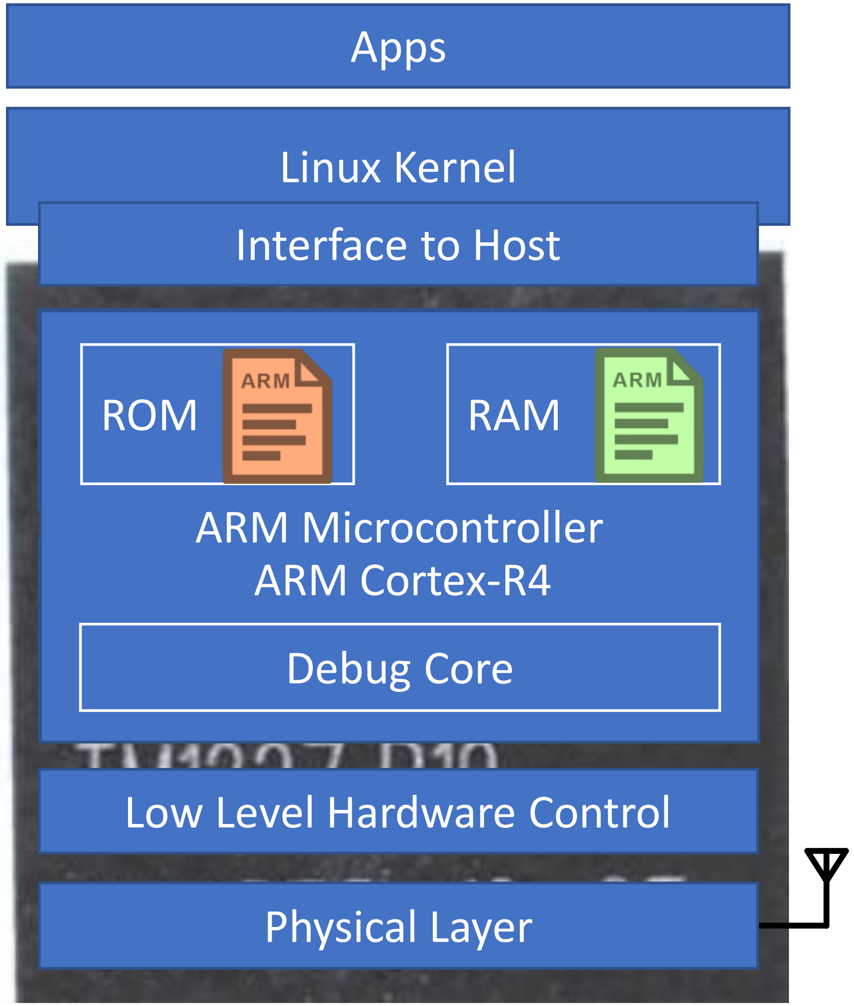
← Send IOCTL

→ Handle Response
← Send IOCTL


Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

Reminder: We need to unlock debug registers!

Accessing Debug Core



Firmware Execution

Initialize Hardware

Enable Interrupts

Go to Sleep
→ Wait for Interrupts

Wake up to handle IOCTL

Wake up to handle IOCTL:

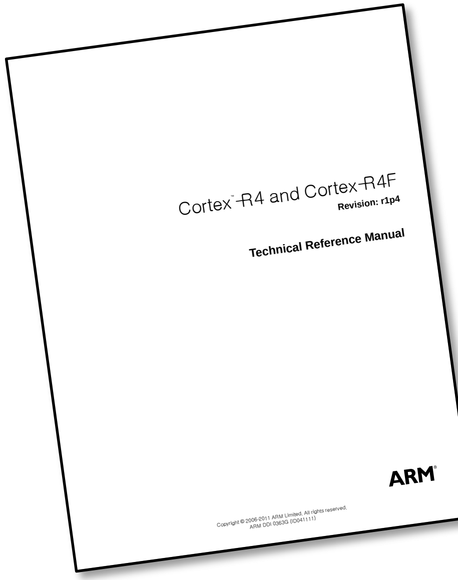
```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware crashes ☹

Events

← Send IOCTL

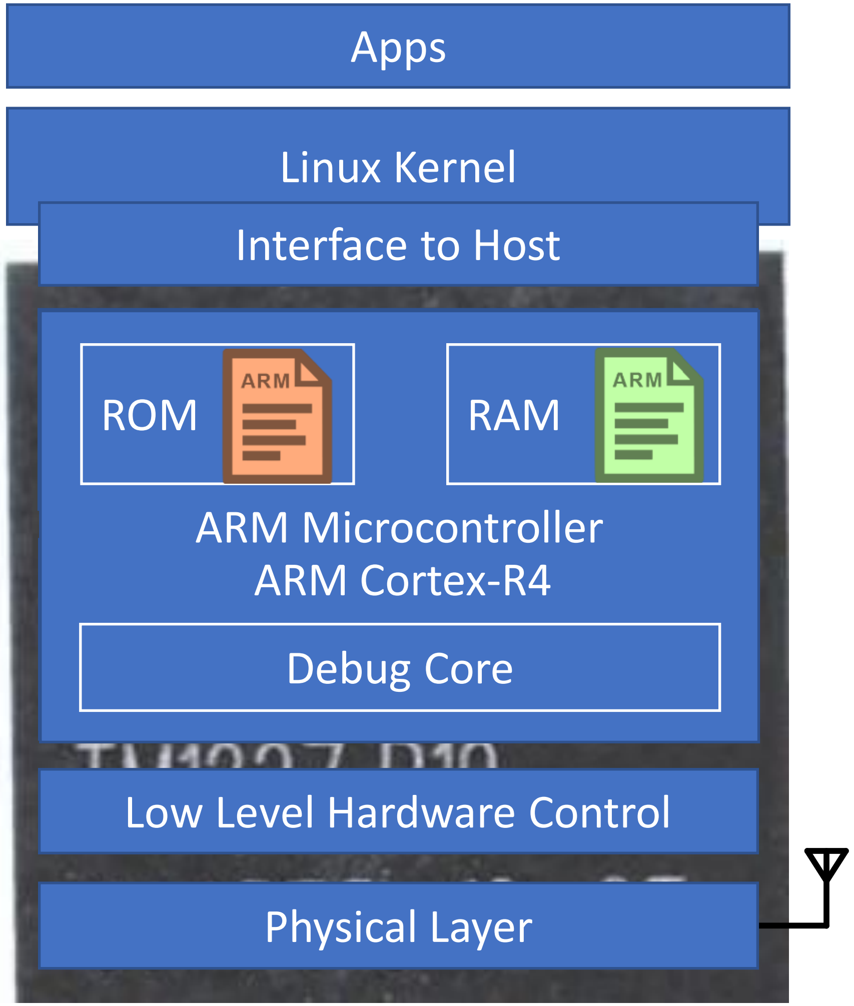
→ Handle Response
← Send IOCTL



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

Reminder: We need to unlock debug registers!

Accessing Debug Core



Apps

Linux Kernel

Interface to Host

ROM

RAM

ARM Microcontroller
ARM Cortex-R4

Debug Core

Low Level Hardware Control

Physical Layer

Firmware Execution

Initialize Hardware

Enable Interrupts

Go to Sleep
→ Wait for Interrupts

Wake up to handle IOCTL

Wake up to handle IOCTL:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

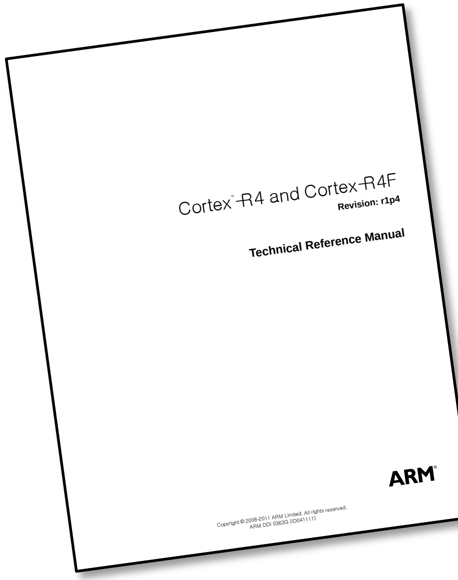
Firmware crashes ☹

Events

← Send IOCTL

→ Handle Response

← Send IOCTL



Cortex-R4 and Cortex-R4F
Revision: r1p4

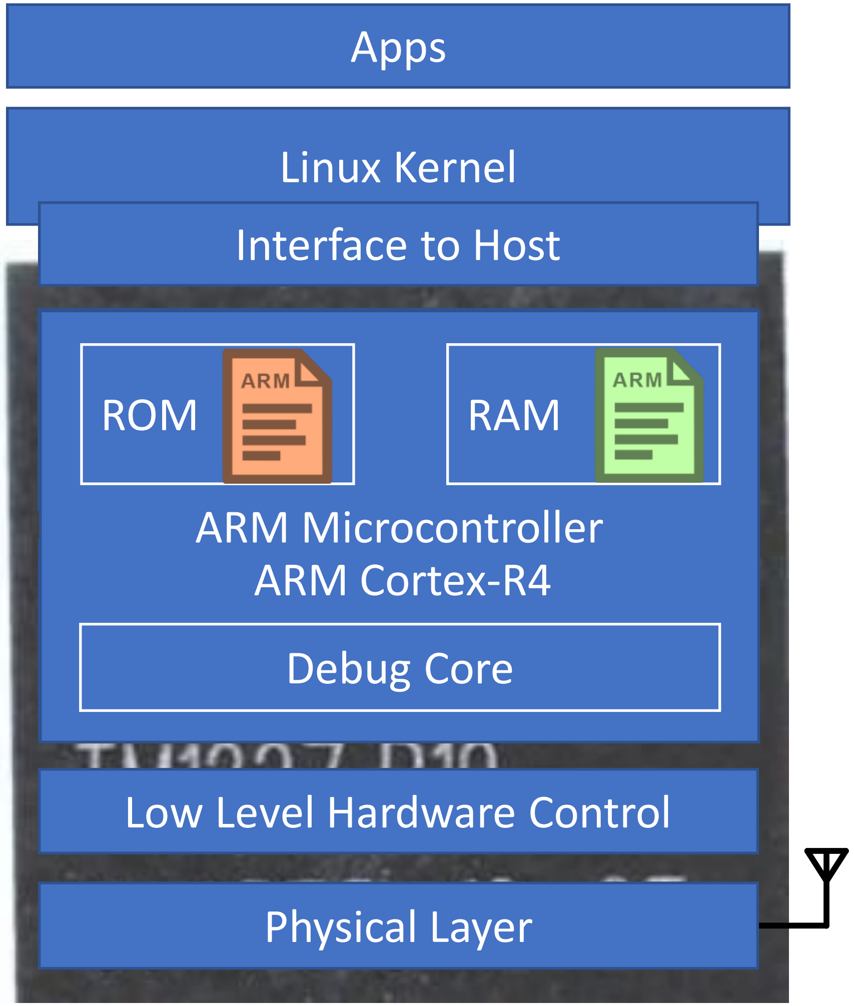
Technical Reference Manual

ARM

→ Why can‘t we access the debug registers? Is the debug core even available?

Reminder: We need to unlock debug registers!

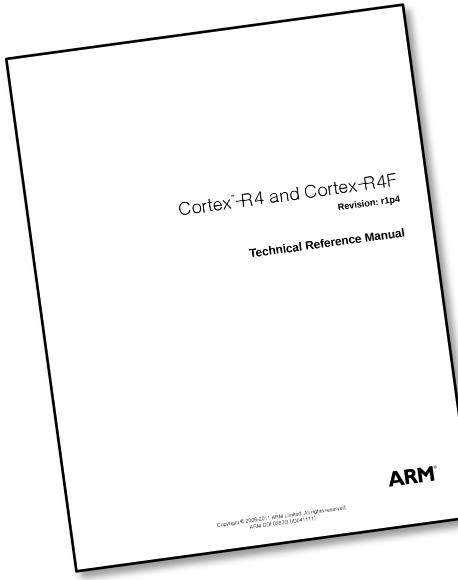
Accessing Debug Core



Firmware Execution

Initialize Hardware

Events


Cortex-R4 and Cortex-R4F
Revision: r1p4

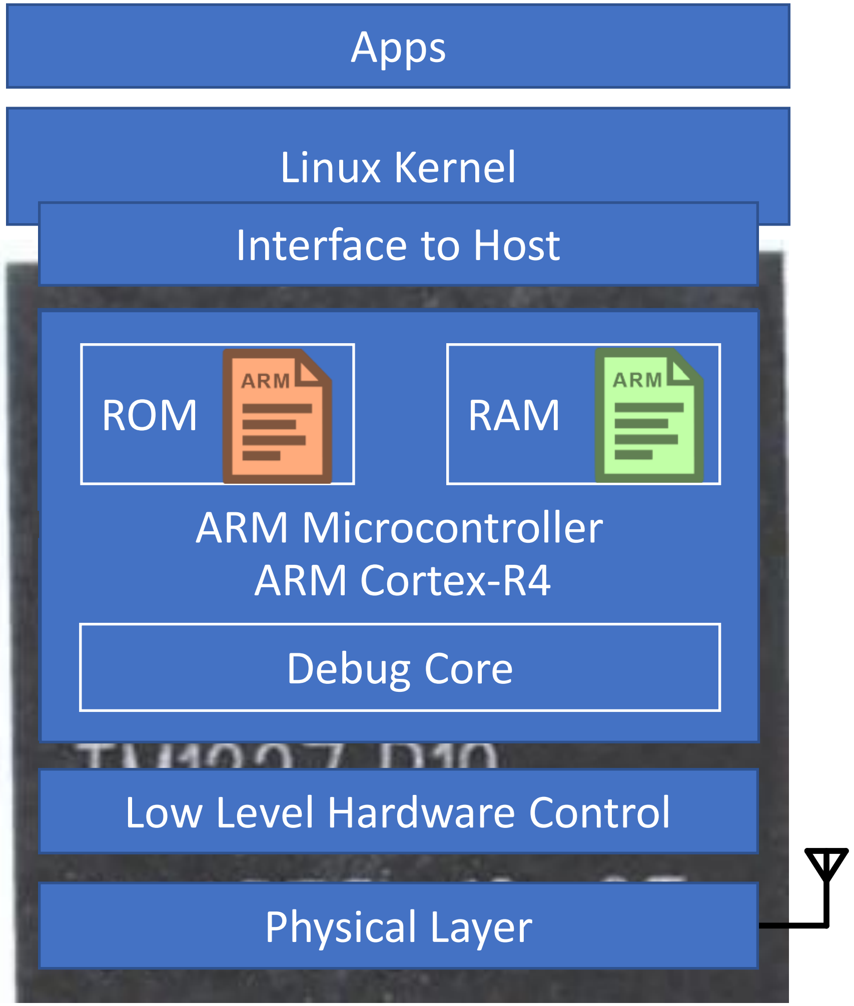
Technical Reference Manual

ARM

➔ **Why can‘t we access the debug registers? Is the debug core even available?**

Reminder: We need to unlock debug registers!

Accessing Debug Core



Apps

Linux Kernel

Interface to Host

ROM

RAM

ARM Microcontroller
ARM Cortex-R4

Debug Core

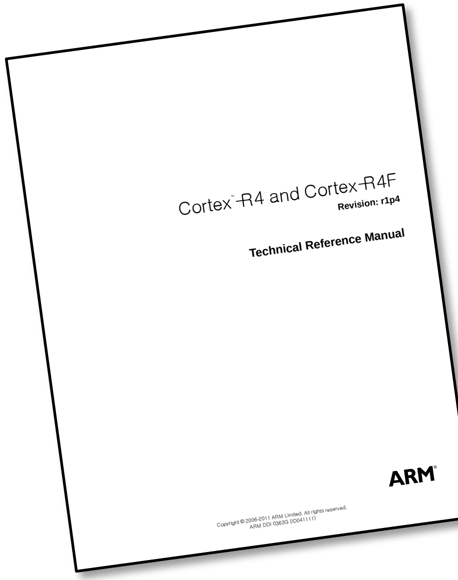
Low Level Hardware Control

Physical Layer

Firmware Execution

Initialize Hardware

Events



Cortex-R4 and Cortex-R4F
Revision: r1p4

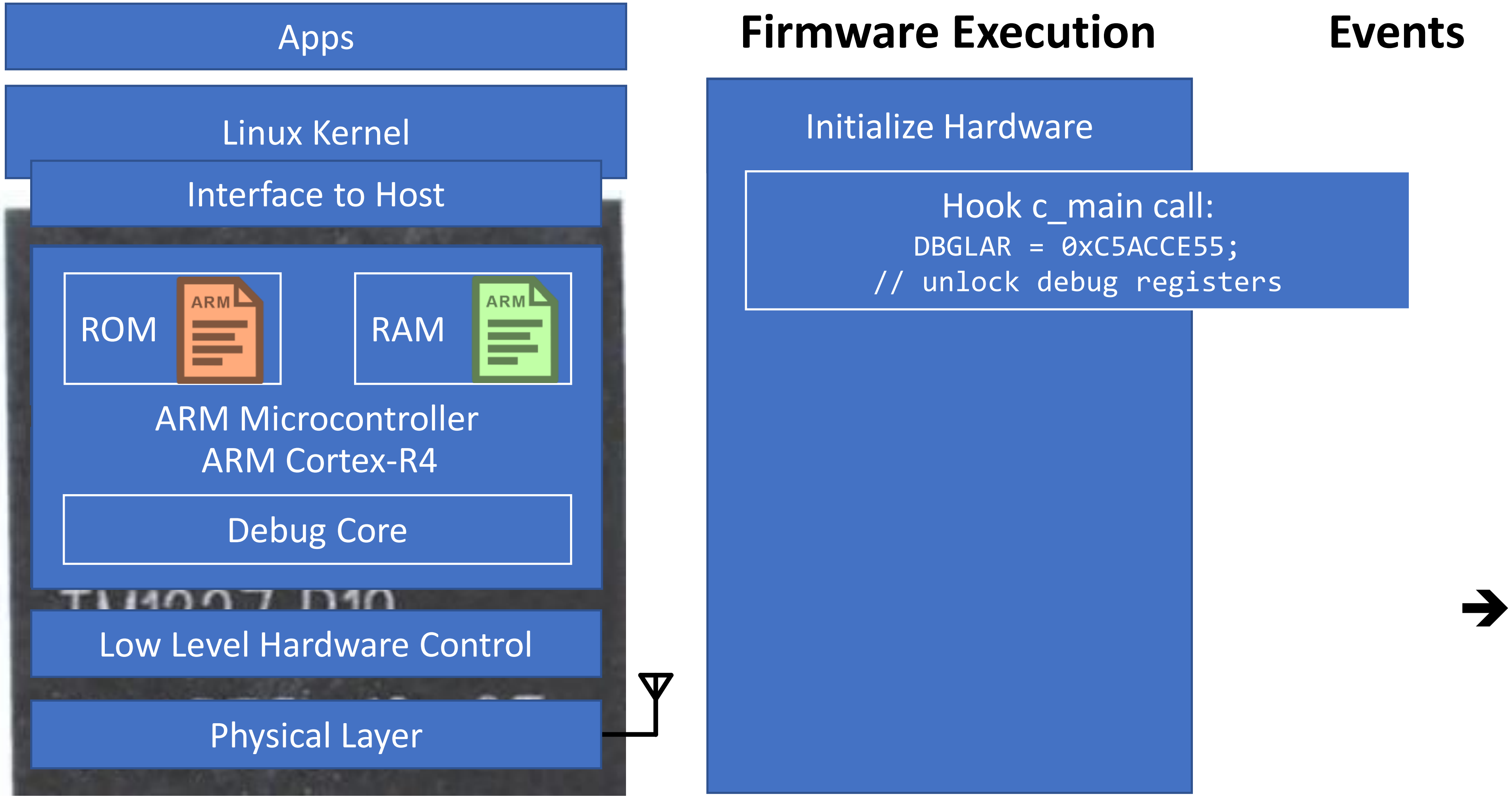
Technical Reference Manual

ARM

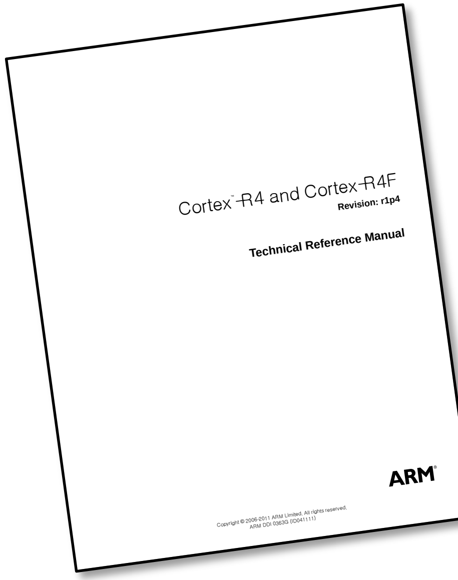
➔ **Why can‘t we access the debug registers? Is the debug core even available?**

Reminder: We need to unlock debug registers!

Accessing Debug Core



```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```



Cortex™-R4 and Cortex-R4F
Revision: r1p4

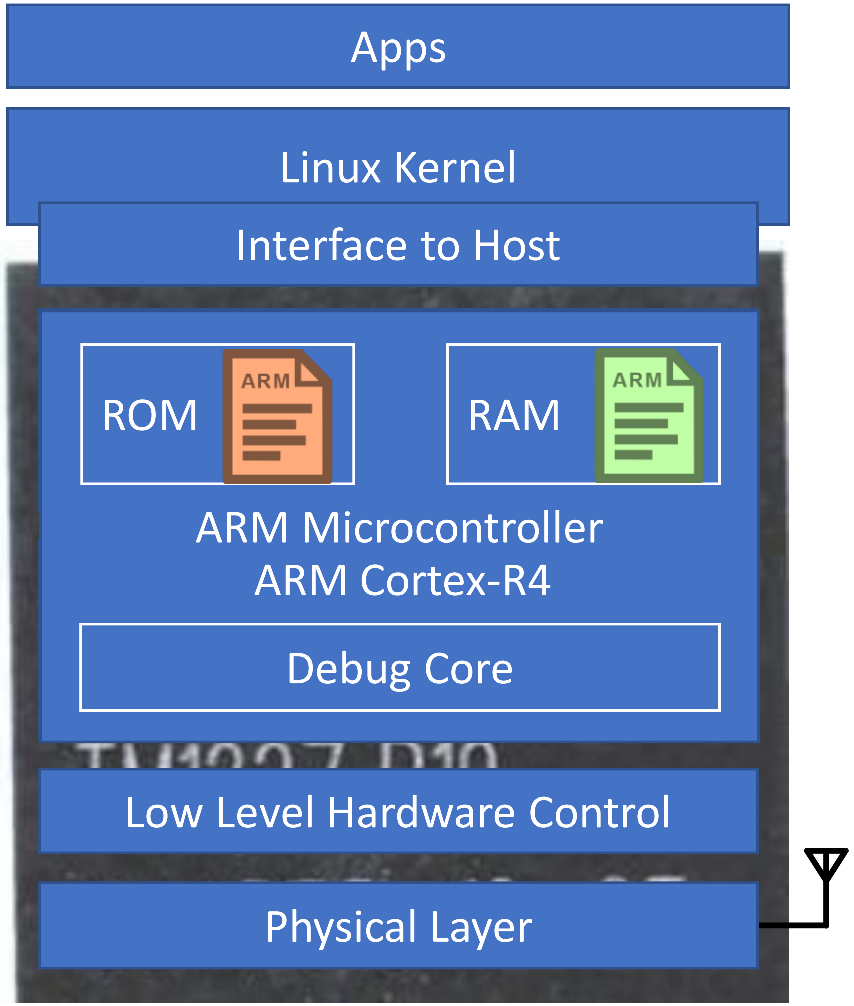
Technical Reference Manual

ARM®

→ **Why can‘t we access the debug registers? Is the debug core even available?**

Reminder: We need to unlock debug registers!

Accessing Debug Core



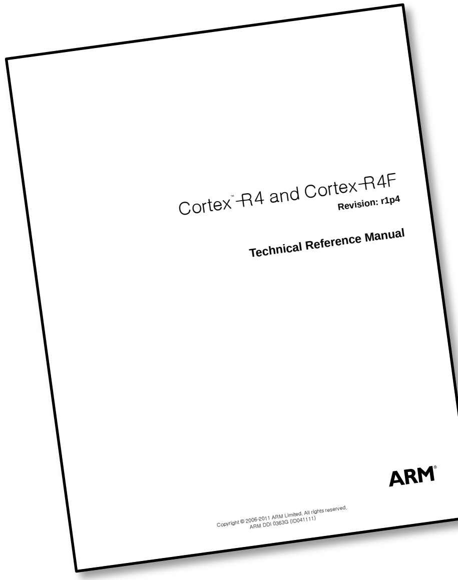
Firmware Execution **Events**

Initialize Hardware

Hook c_main call:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware keeps running ☺

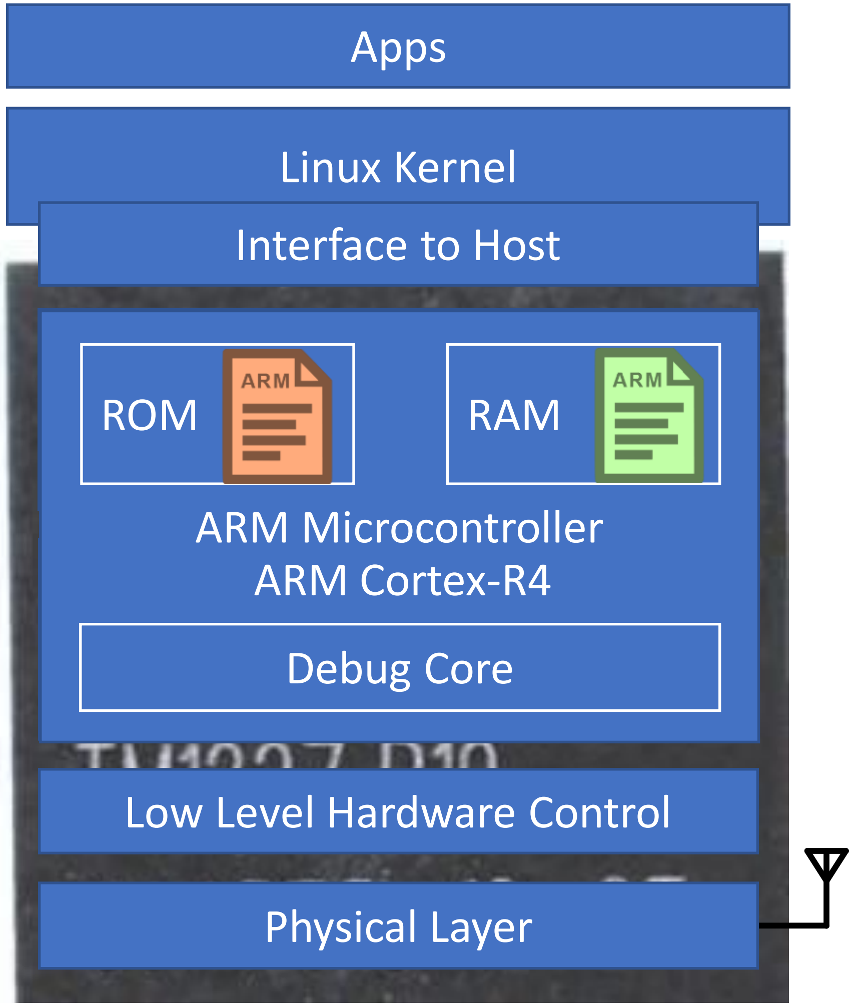


Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

→ **Why can‘t we access the debug registers? Is the debug core even available?**

Reminder: We need to unlock debug registers! ✓

Accessing Debug Core



Apps

Linux Kernel

Interface to Host

ROM ARM

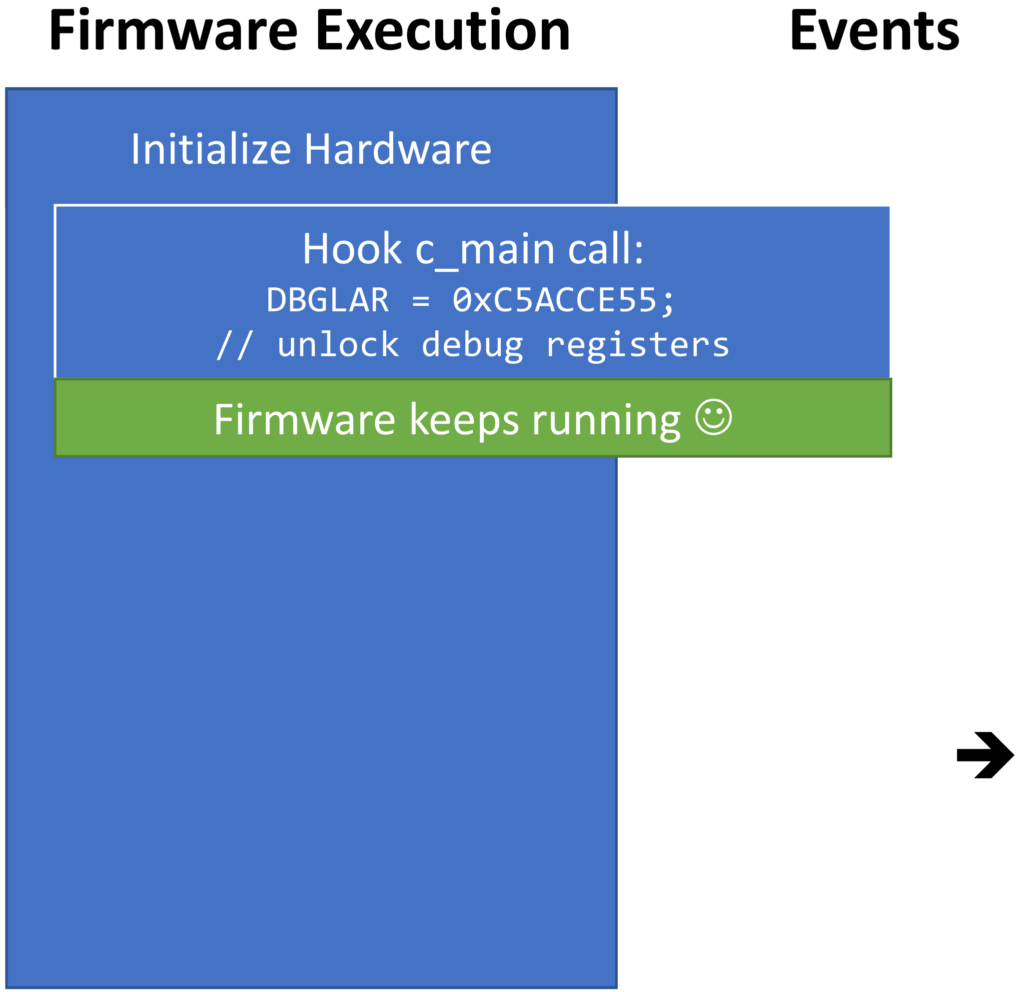
RAM ARM

ARM Microcontroller
ARM Cortex-R4

Debug Core

Low Level Hardware Control

Physical Layer



Firmware Execution

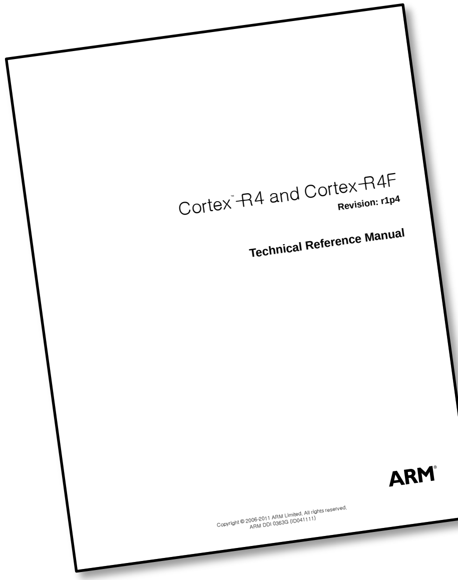
Events

Initialize Hardware

Hook c_main call:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware keeps running ☺



Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

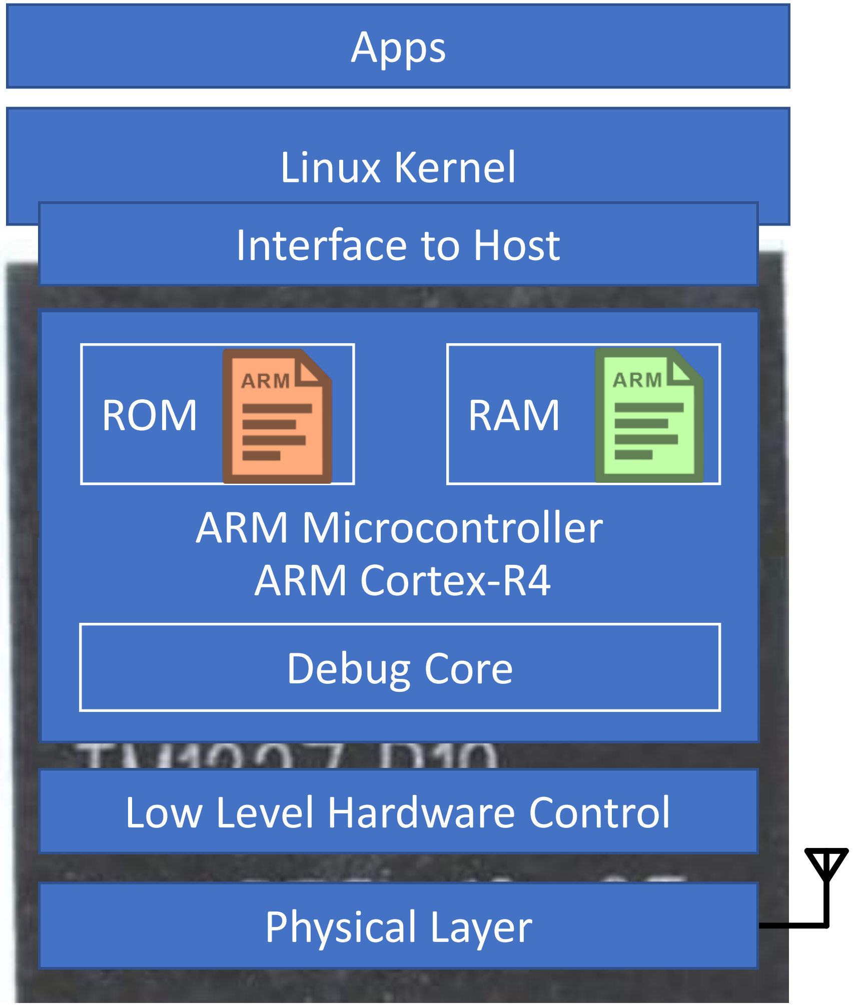
ARM

➔ **Why can‘t we access the debug registers?**

~~**Is the debug core even available?**~~

Reminder: We need to unlock debug registers! ✓

Accessing Debug Core



Firmware Execution

Events

Initialize Hardware

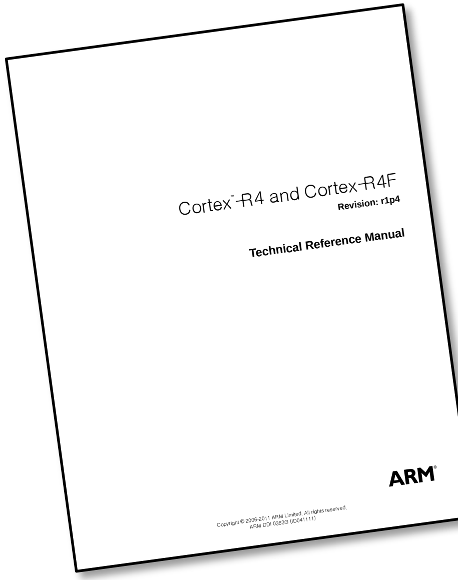
Hook c_main call:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware keeps running ☺

Add code before enabling interrupts:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```



Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

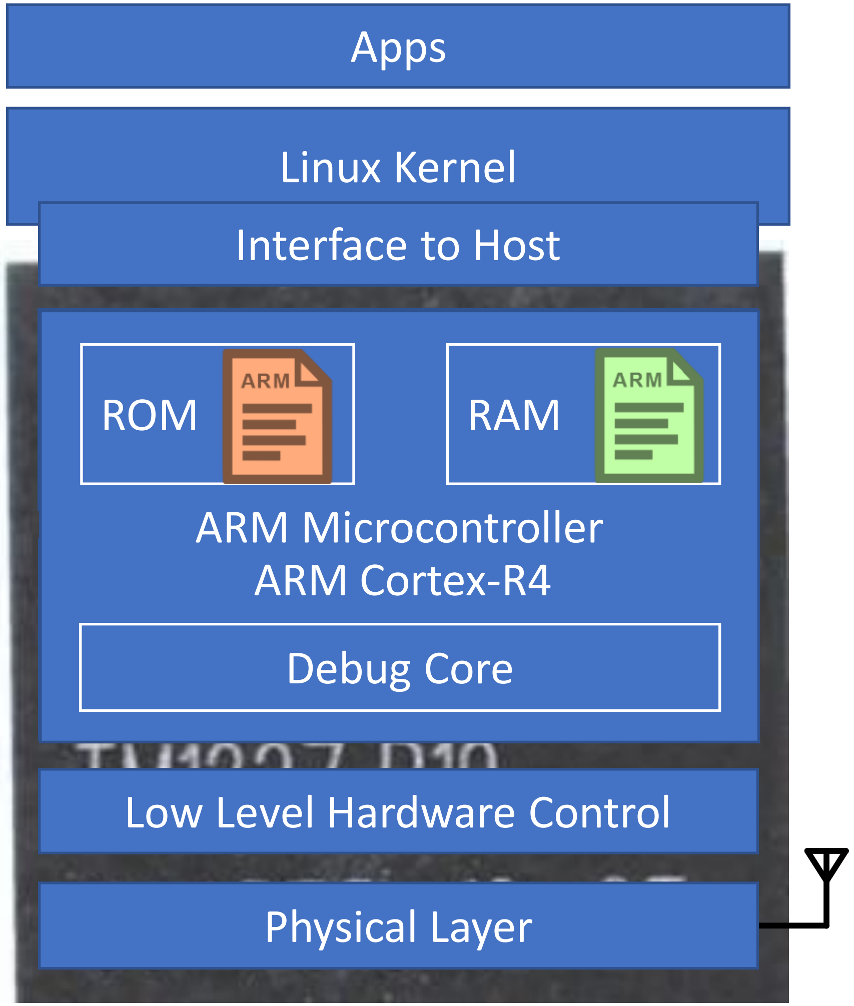
ARM

➔ **Why can‘t we access the debug registers?**

~~**Is the debug core even available?**~~

Reminder: We need to unlock debug registers! ✓

Accessing Debug Core



Firmware Execution

Events

Initialize Hardware

Hook c_main call:

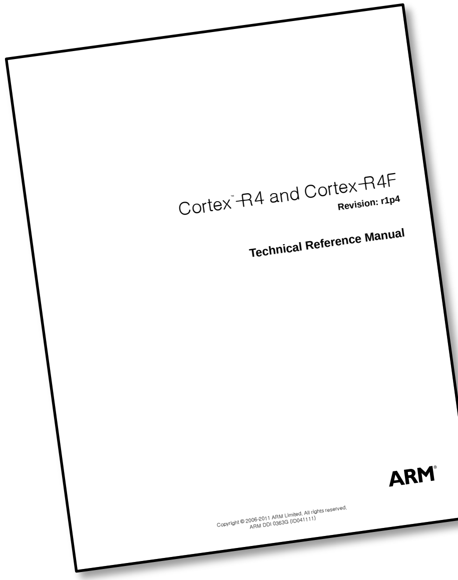
```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware keeps running ☺

Add code before enabling interrupts:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware crashes ☹



Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

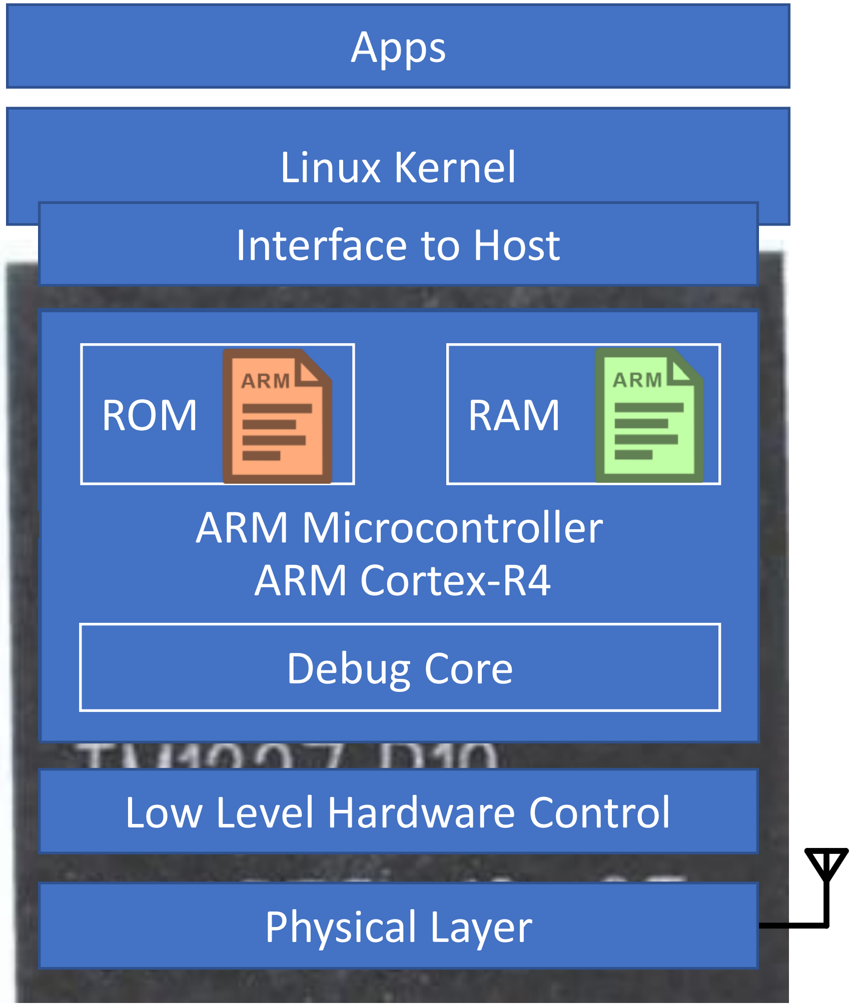
ARM

→ **Why can't we access the debug registers?**

~~Is the debug core even available?~~

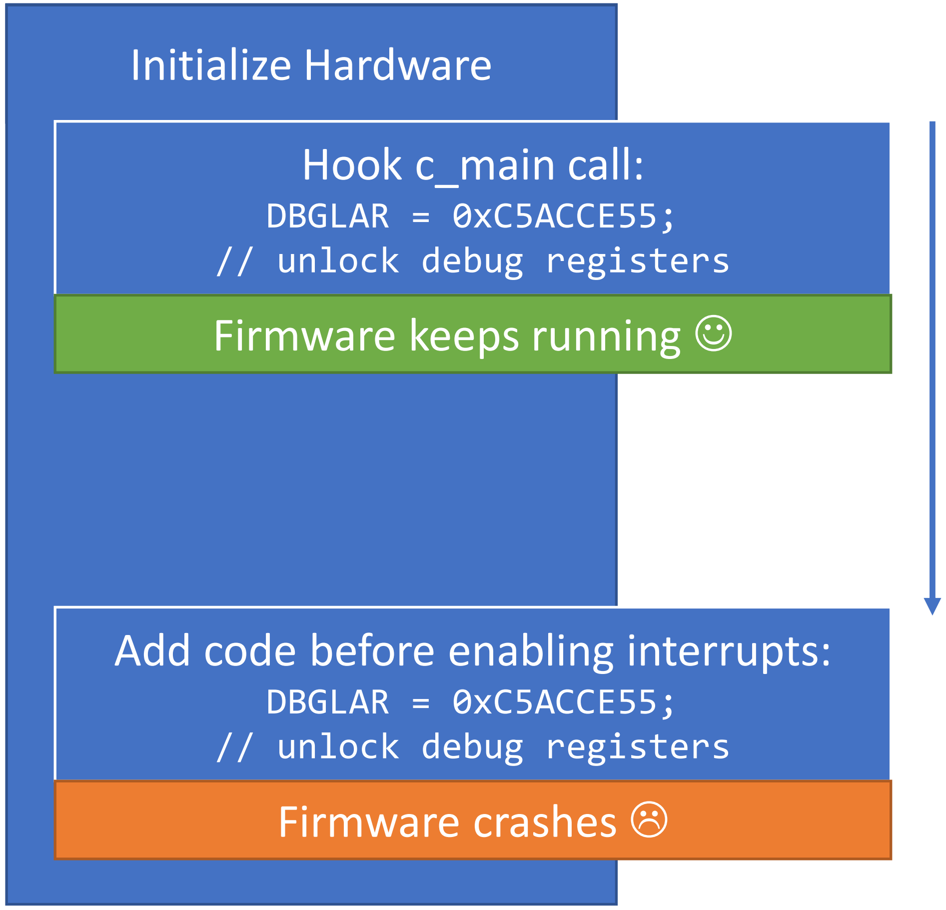
Reminder: We need to unlock debug registers! ✓

Accessing Debug Core



Firmware Execution

Events


Initialize Hardware

Hook c_main call:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

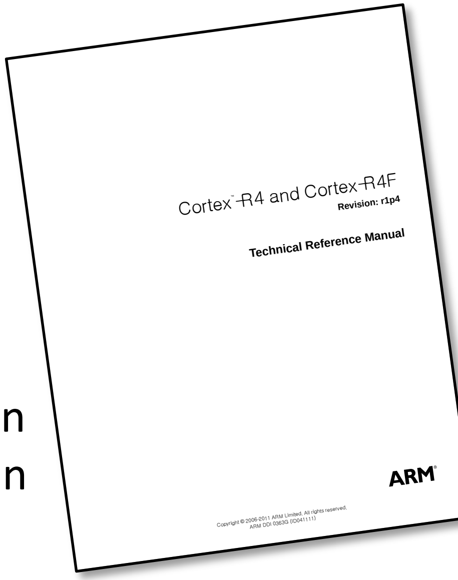
Firmware keeps running ☺

Add code before enabling interrupts:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware crashes ☹

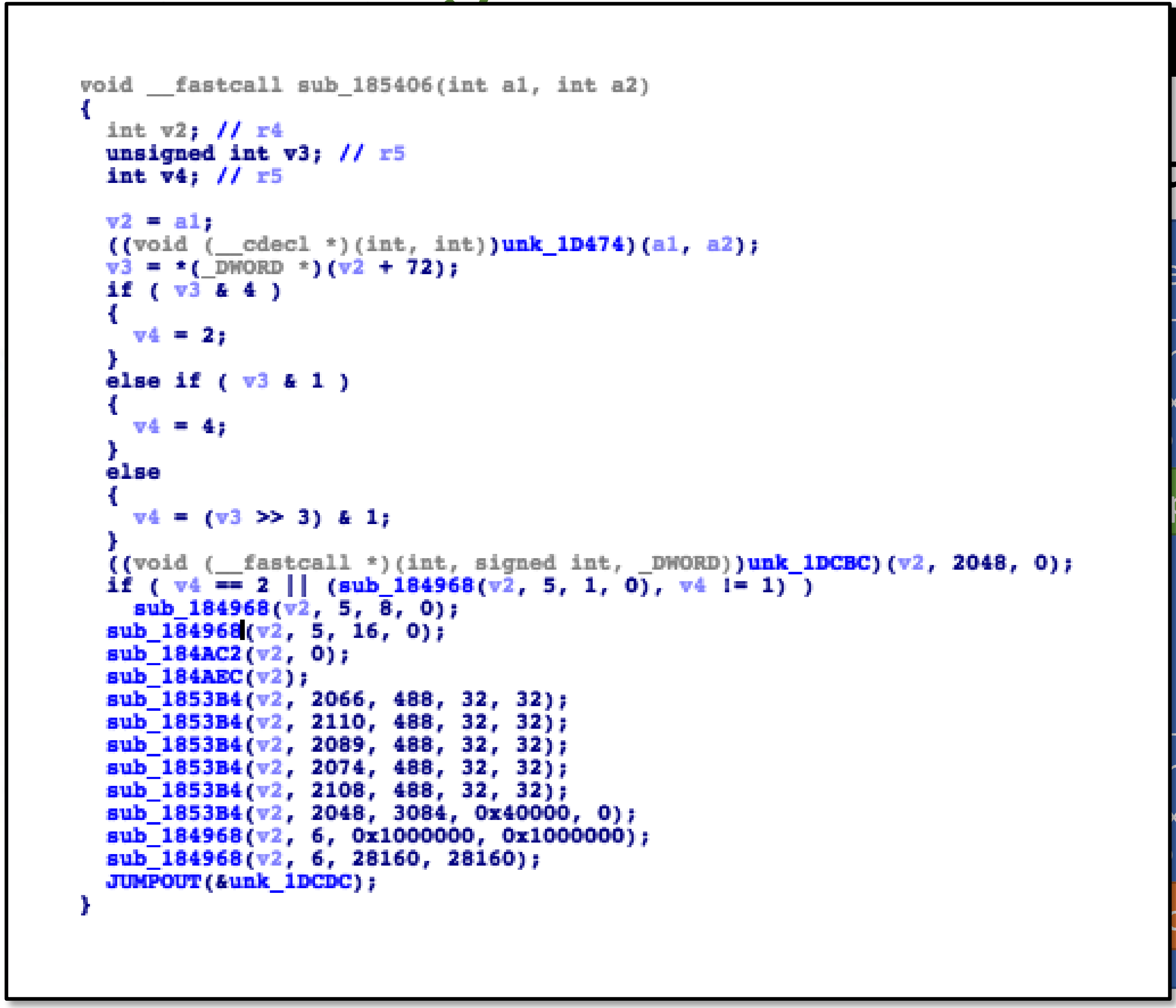
Somewhen in between access to debug core breaks



➔ **Why can‘t we access the debug registers?**

~~Is the debug core even available?~~

bug Core



```
void __fastcall sub_185406(int a1, int a2)
{
  int v2; // r4
  unsigned int v3; // r5
  int v4; // r5

  v2 = a1;
  ((void (__cdecl *)(int, int))unk_1D474)(a1, a2);
  v3 = *(_DWORD *)(v2 + 72);
  if ( v3 & 4 )
  {
    v4 = 2;
  }
  else if ( v3 & 1 )
  {
    v4 = 4;
  }
  else
  {
    v4 = (v3 >> 3) & 1;
  }
  ((void (__fastcall *)(int, signed int, _DWORD))unk_1DCBC)(v2, 2048, 0);
  if ( v4 == 2 || (sub_184968(v2, 5, 1, 0), v4 != 1) )
    sub_184968(v2, 5, 8, 0);
  sub_184968(v2, 5, 16, 0);
  sub_184AC2(v2, 0);
  sub_184AEC(v2);
  sub_1853B4(v2, 2066, 488, 32, 32);
  sub_1853B4(v2, 2110, 488, 32, 32);
  sub_1853B4(v2, 2089, 488, 32, 32);
  sub_1853B4(v2, 2074, 488, 32, 32);
  sub_1853B4(v2, 2108, 488, 32, 32);
  sub_1853B4(v2, 2048, 3084, 0x40000, 0);
  sub_184968(v2, 6, 0x1000000, 0x1000000);
  sub_184968(v2, 6, 28160, 28160);
  JUMPOUT(&unk_1DCDC);
}
```

tion

e

nain call:
xC5ACCE55;
ug registers

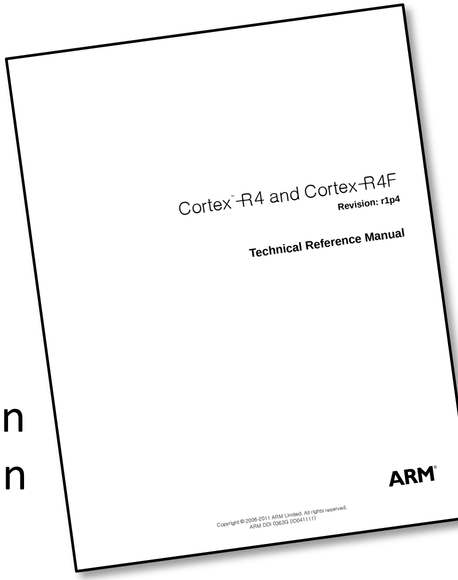
ps running ☺

nabling interrupts:
xC5ACCE55;
ug registers

crashes ☹

Events

Somewhen in between access to debug core breaks

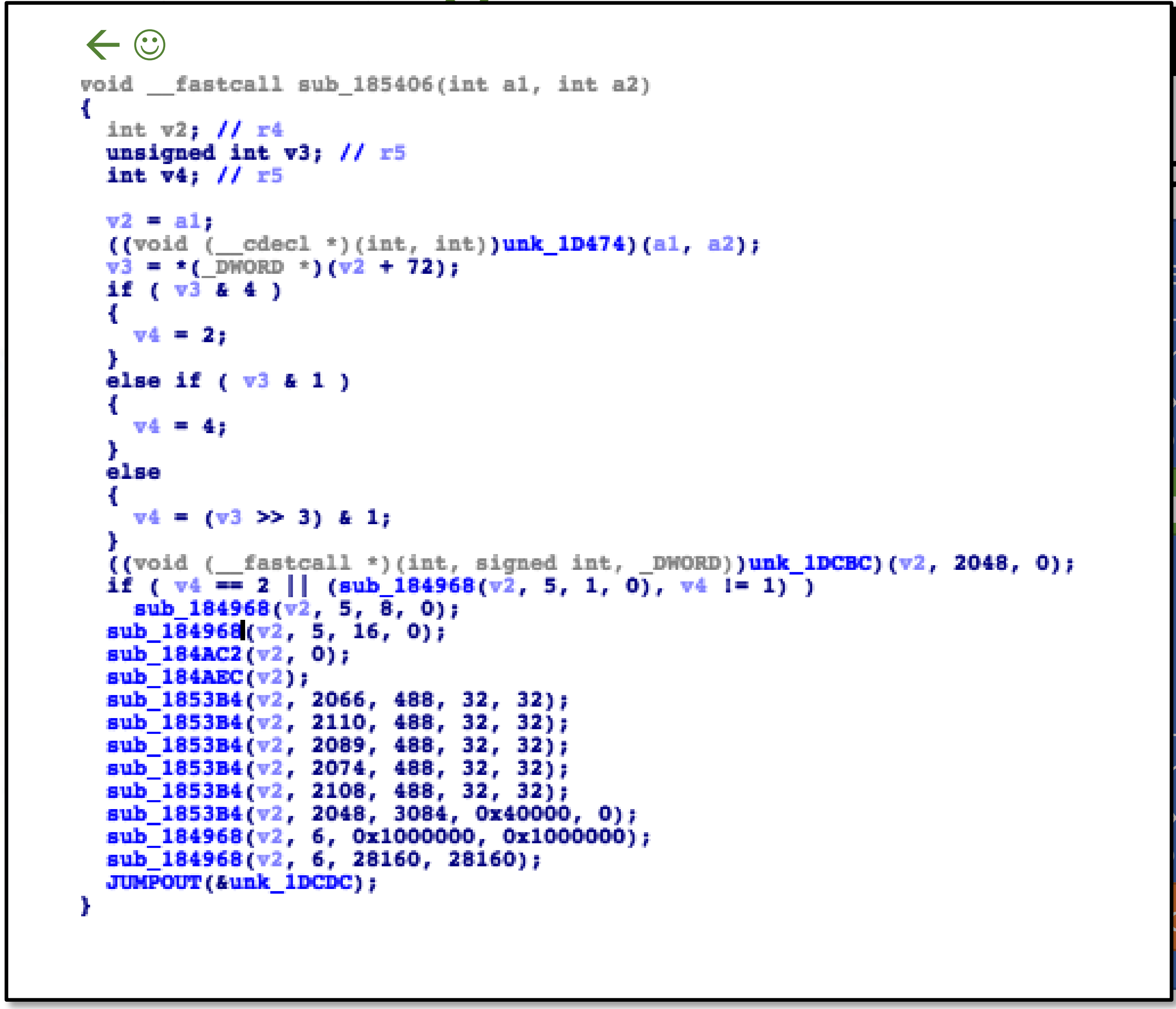


Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➔ **Why can‘t we access the debug registers?**

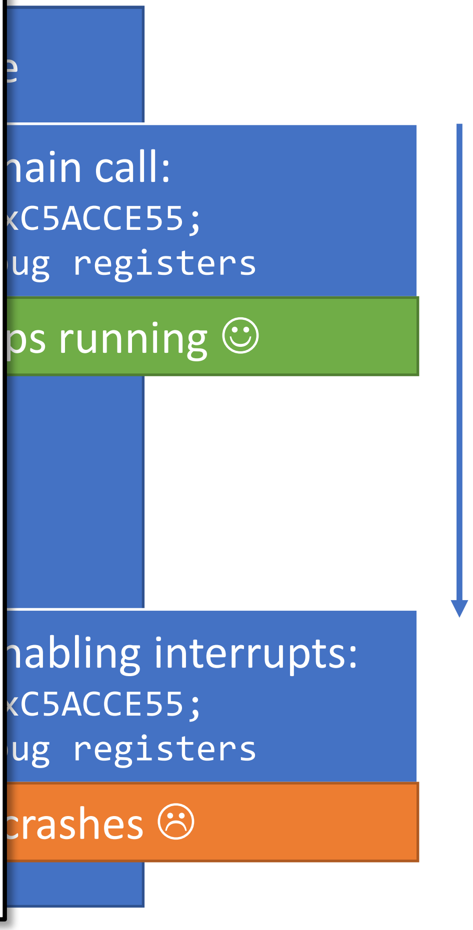
~~**Is the debug core even available?**~~

bug Core

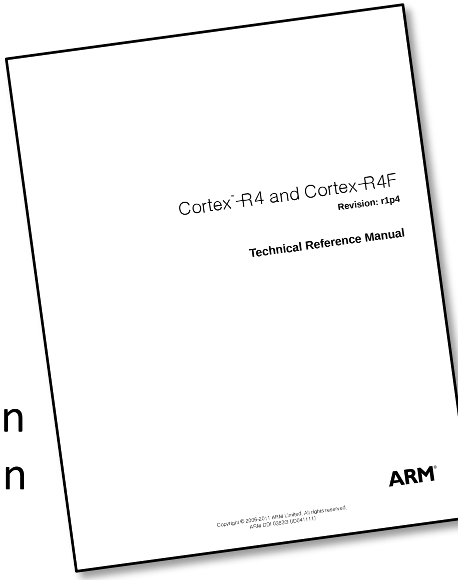


← ☺

```
void __fastcall sub_185406(int a1, int a2)
{
  int v2; // r4
  unsigned int v3; // r5
  int v4; // r5

  v2 = a1;
  ((void (__cdecl *)(int, int))unk_1D474)(a1, a2);
  v3 = *(_DWORD *)(v2 + 72);
  if ( v3 & 4 )
  {
    v4 = 2;
  }
  else if ( v3 & 1 )
  {
    v4 = 4;
  }
  else
  {
    v4 = (v3 >> 3) & 1;
  }
  ((void (__fastcall *)(int, signed int, _DWORD))unk_1DCBC)(v2, 2048, 0);
  if ( v4 == 2 || (sub_184968(v2, 5, 1, 0), v4 != 1) )
    sub_184968(v2, 5, 8, 0);
  sub_184968(v2, 5, 16, 0);
  sub_184AC2(v2, 0);
  sub_184AEC(v2);
  sub_1853B4(v2, 2066, 488, 32, 32);
  sub_1853B4(v2, 2110, 488, 32, 32);
  sub_1853B4(v2, 2089, 488, 32, 32);
  sub_1853B4(v2, 2074, 488, 32, 32);
  sub_1853B4(v2, 2108, 488, 32, 32);
  sub_1853B4(v2, 2048, 3084, 0x40000, 0);
  sub_184968(v2, 6, 0x1000000, 0x1000000);
  sub_184968(v2, 6, 28160, 28160);
  JUMPOUT(&unk_1DCDC);
}
```



Somewhen in between access to debug core breaks

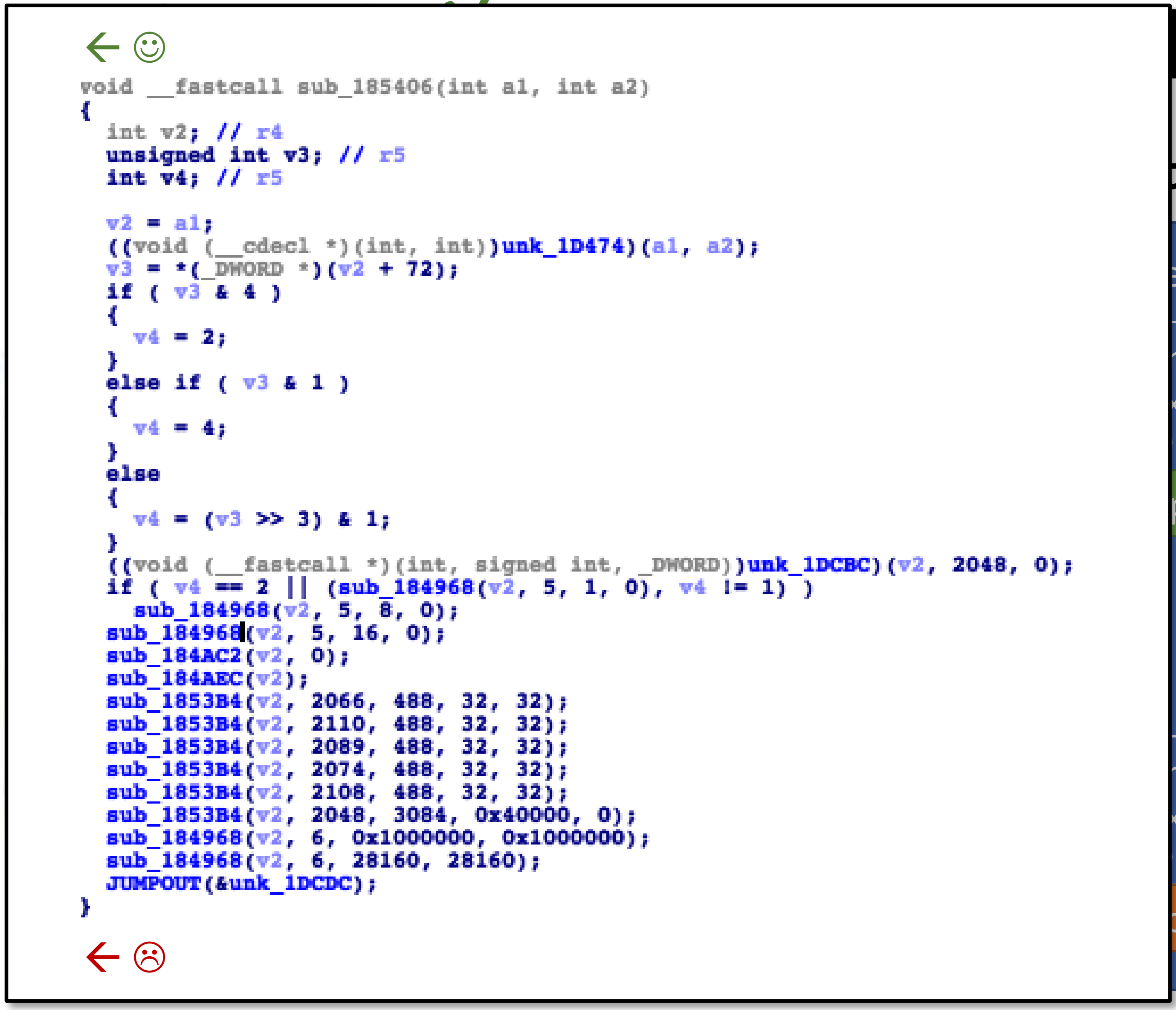


Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual

➔ **Why can‘t we access the debug registers?**

~~**Is the debug core even available?**~~

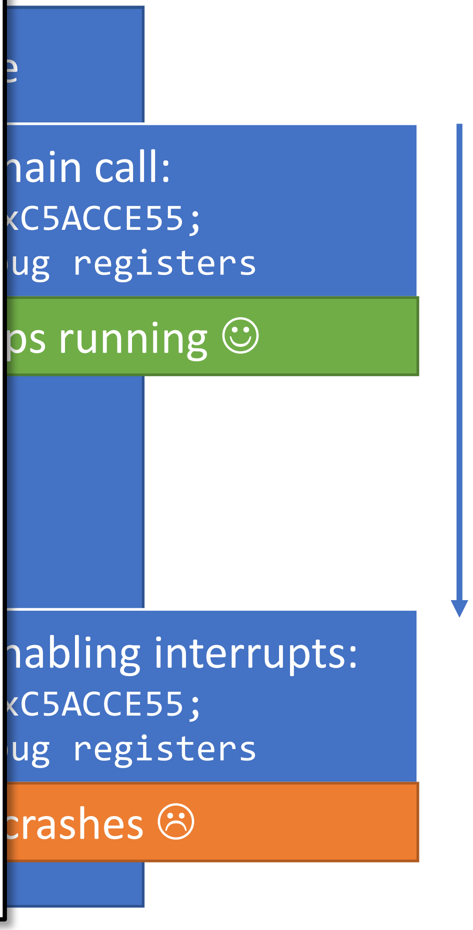
bug Core



← ☺

```
void __fastcall sub_185406(int a1, int a2)
{
  int v2; // r4
  unsigned int v3; // r5
  int v4; // r5

  v2 = a1;
  ((void (__cdecl *)(int, int))unk_1D474)(a1, a2);
  v3 = *(_DWORD *)(v2 + 72);
  if ( v3 & 4 )
  {
    v4 = 2;
  }
  else if ( v3 & 1 )
  {
    v4 = 4;
  }
  else
  {
    v4 = (v3 >> 3) & 1;
  }
  ((void (__fastcall *)(int, signed int, _DWORD))unk_1DCBC)(v2, 2048, 0);
  if ( v4 == 2 || (sub_184968(v2, 5, 1, 0), v4 != 1) )
    sub_184968(v2, 5, 8, 0);
  sub_184968(v2, 5, 16, 0);
  sub_184AC2(v2, 0);
  sub_184AEC(v2);
  sub_1853B4(v2, 2066, 488, 32, 32);
  sub_1853B4(v2, 2110, 488, 32, 32);
  sub_1853B4(v2, 2089, 488, 32, 32);
  sub_1853B4(v2, 2074, 488, 32, 32);
  sub_1853B4(v2, 2108, 488, 32, 32);
  sub_1853B4(v2, 2048, 3084, 0x40000, 0);
  sub_184968(v2, 6, 0x1000000, 0x1000000);
  sub_184968(v2, 6, 28160, 28160);
  JUMPOUT(&unk_1DCDC);
}
```

← ☹



tion

Events

e

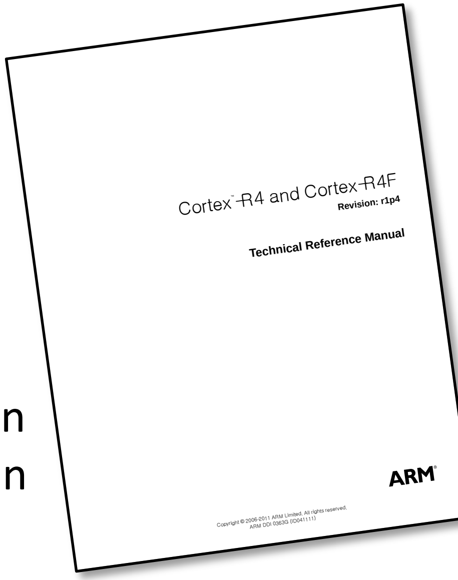
nain call:
xC5ACCE55;
ug registers

ps running ☺

nabling interrupts:
xC5ACCE55;
ug registers

crashes ☹

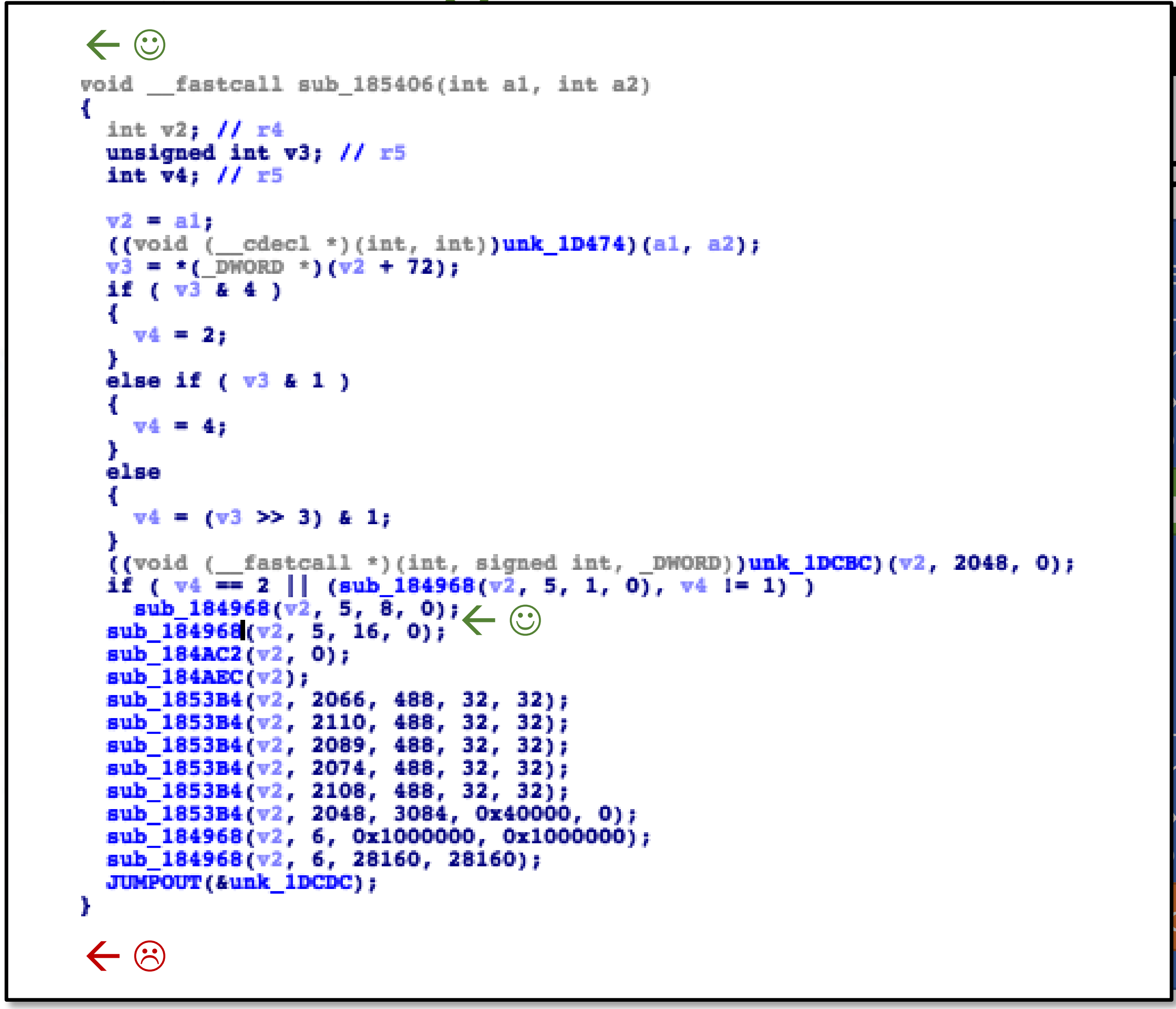
Somewhen in between access to debug core breaks



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

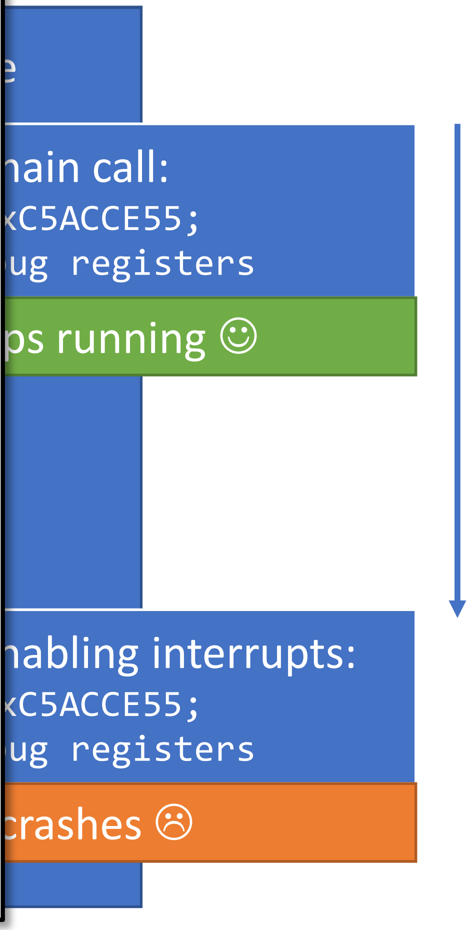
➔ **Why can't we access the debug registers?**
~~Is the debug core even available?~~

bug Core

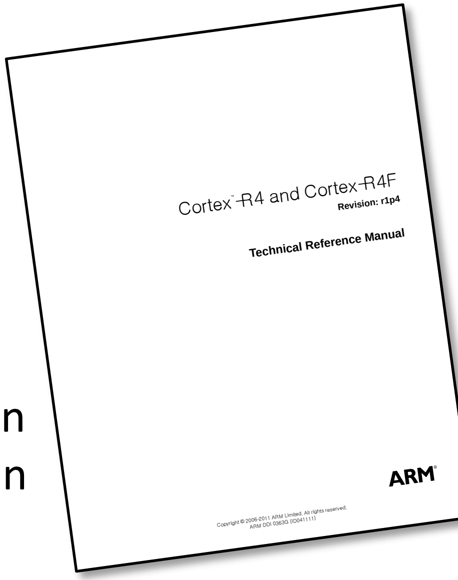


```
← ☺
void __fastcall sub_185406(int a1, int a2)
{
  int v2; // r4
  unsigned int v3; // r5
  int v4; // r5

  v2 = a1;
  ((void (__cdecl *)(int, int))unk_1D474)(a1, a2);
  v3 = *(_DWORD *)(v2 + 72);
  if ( v3 & 4 )
  {
    v4 = 2;
  }
  else if ( v3 & 1 )
  {
    v4 = 4;
  }
  else
  {
    v4 = (v3 >> 3) & 1;
  }
  ((void (__fastcall *)(int, signed int, _DWORD))unk_1DCBC)(v2, 2048, 0);
  if ( v4 == 2 || (sub_184968(v2, 5, 1, 0), v4 != 1) )
    sub_184968(v2, 5, 8, 0);← ☺
  sub_184968(v2, 5, 16, 0);
  sub_184AC2(v2, 0);
  sub_184AEC(v2);
  sub_1853B4(v2, 2066, 488, 32, 32);
  sub_1853B4(v2, 2110, 488, 32, 32);
  sub_1853B4(v2, 2089, 488, 32, 32);
  sub_1853B4(v2, 2074, 488, 32, 32);
  sub_1853B4(v2, 2108, 488, 32, 32);
  sub_1853B4(v2, 2048, 3084, 0x40000, 0);
  sub_184968(v2, 6, 0x1000000, 0x1000000);
  sub_184968(v2, 6, 28160, 28160);
  JUMPOUT(&unk_1DCDC);
}

← ☹
```

tion **Events**



Somewhen in between access to debug core breaks

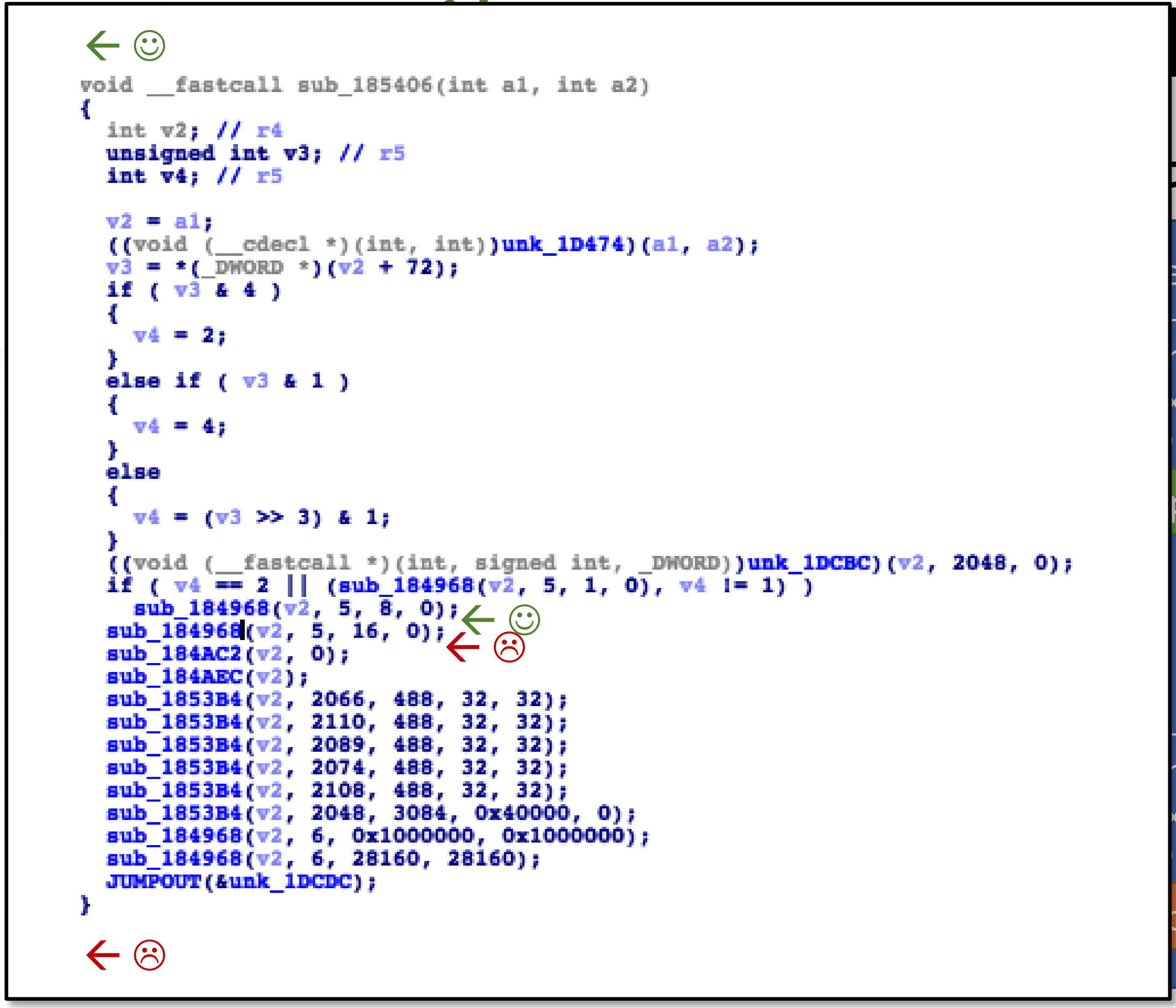


Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➔ **Why can‘t we access the debug registers?**

~~Is the debug core even available?~~

bug Core



```
← ☺
void __fastcall sub_185406(int a1, int a2)
{
  int v2; // r4
  unsigned int v3; // r5
  int v4; // r5

  v2 = a1;
  ((void (__cdecl *)(int, int))unk_1D474)(a1, a2);
  v3 = *(_DWORD *)(v2 + 72);
  if ( v3 & 4 )
  {
    v4 = 2;
  }
  else if ( v3 & 1 )
  {
    v4 = 4;
  }
  else
  {
    v4 = (v3 >> 3) & 1;
  }
  ((void (__fastcall *)(int, signed int, _DWORD))unk_1DCBC)(v2, 2048, 0);
  if ( v4 == 2 || (sub_184968(v2, 5, 1, 0), v4 != 1) )
    sub_184968(v2, 5, 8, 0); ← ☺
  sub_184968(v2, 5, 16, 0); ← ☹
  sub_184AC2(v2, 0);
  sub_184AEC(v2);
  sub_1853B4(v2, 2066, 488, 32, 32);
  sub_1853B4(v2, 2110, 488, 32, 32);
  sub_1853B4(v2, 2089, 488, 32, 32);
  sub_1853B4(v2, 2074, 488, 32, 32);
  sub_1853B4(v2, 2108, 488, 32, 32);
  sub_1853B4(v2, 2048, 3084, 0x40000, 0);
  sub_184968(v2, 6, 0x1000000, 0x1000000);
  sub_184968(v2, 6, 28160, 28160);
  JUMPOUT(&unk_1DCDC);
}
← ☹
```

tion | **Events**

e

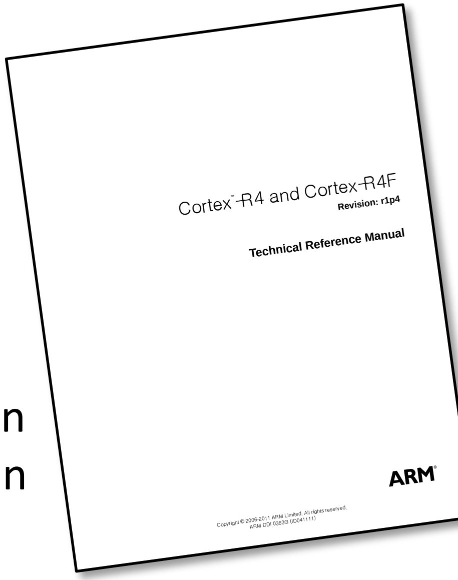
nain call:
xC5ACCE55;
ug registers

ps running ☺

Somewhen in between access to debug core breaks

nabling interrupts:
xC5ACCE55;
ug registers

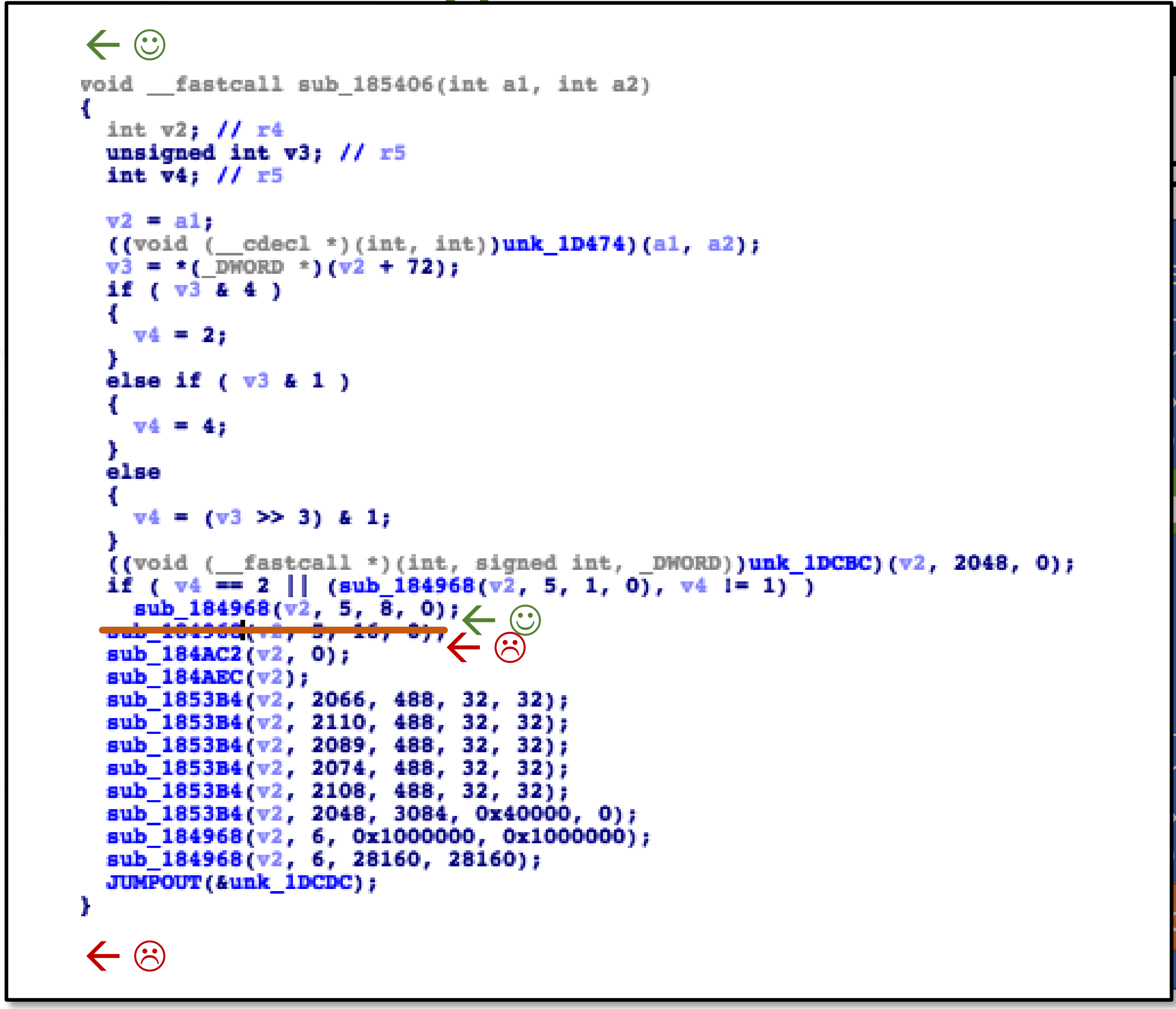
crashes ☹



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➔ **Why can‘t we access the debug registers?**
~~Is the debug core even available?~~

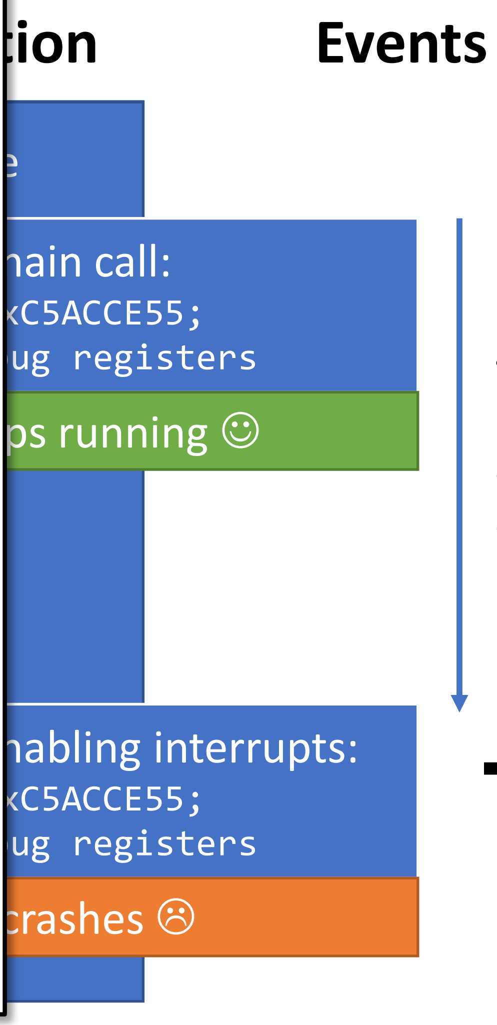
bug Core



← ☺

```
void __fastcall sub_185406(int a1, int a2)
{
  int v2; // r4
  unsigned int v3; // r5
  int v4; // r5

  v2 = a1;
  ((void (__cdecl *)(int, int))unk_1D474)(a1, a2);
  v3 = *(_DWORD *)(v2 + 72);
  if ( v3 & 4 )
  {
    v4 = 2;
  }
  else if ( v3 & 1 )
  {
    v4 = 4;
  }
  else
  {
    v4 = (v3 >> 3) & 1;
  }
  ((void (__fastcall *)(int, signed int, _DWORD))unk_1DCBC)(v2, 2048, 0);
  if ( v4 == 2 || (sub_184968(v2, 5, 1, 0), v4 != 1) )
    sub_184968(v2, 5, 8, 0); ← ☺
  ~~sub_184968(v2, 5, 16, 0);~~ ← ☹
  sub_184AC2(v2, 0);
  sub_184AEC(v2);
  sub_1853B4(v2, 2066, 488, 32, 32);
  sub_1853B4(v2, 2110, 488, 32, 32);
  sub_1853B4(v2, 2089, 488, 32, 32);
  sub_1853B4(v2, 2074, 488, 32, 32);
  sub_1853B4(v2, 2108, 488, 32, 32);
  sub_1853B4(v2, 2048, 3084, 0x40000, 0);
  sub_184968(v2, 6, 0x1000000, 0x1000000);
  sub_184968(v2, 6, 28160, 28160);
  JUMPOUT(&unk_1DCDC);
}
```

← ☹



tion

Events

e

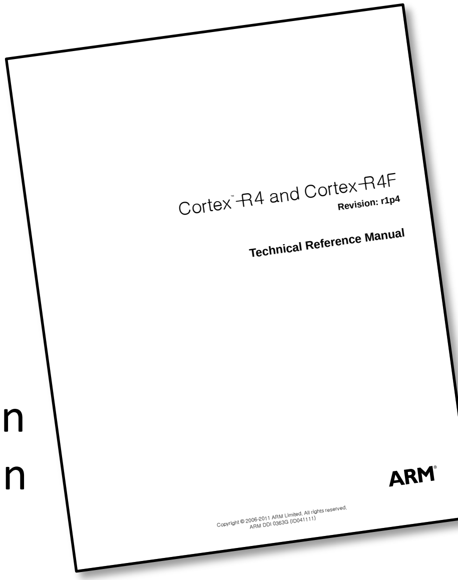
nain call:
xC5ACCE55;
ug registers

ps running ☺

nabling interrupts:
xC5ACCE55;
ug registers

crashes ☹

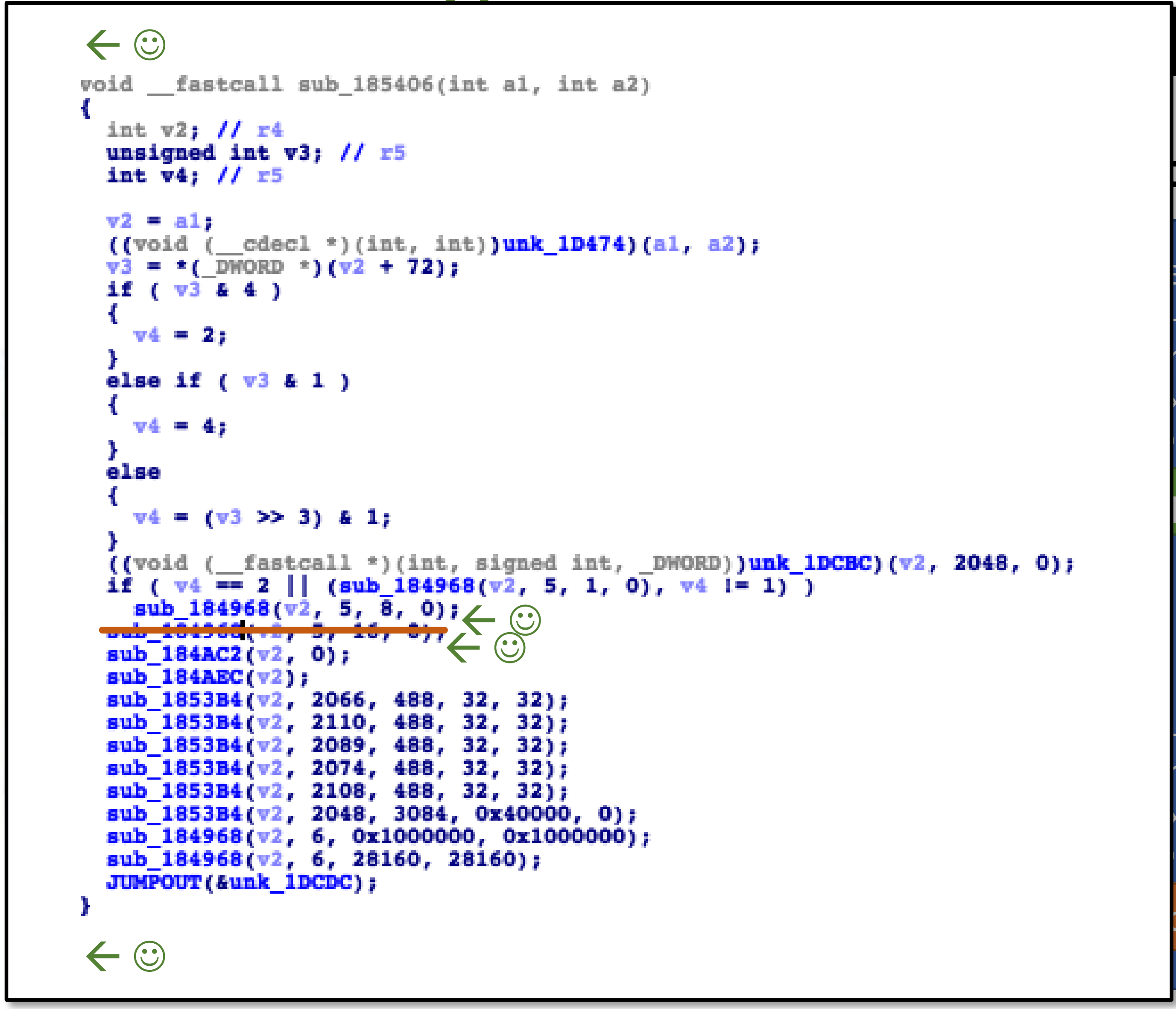
Somewhen in between access to debug core breaks



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➔ **Why can‘t we access the debug registers?**

~~Is the debug core even available?~~



```
← ☺
void __fastcall sub_185406(int a1, int a2)
{
  int v2; // r4
  unsigned int v3; // r5
  int v4; // r5

  v2 = a1;
  ((void (__cdecl *)(int, int))unk_1D474)(a1, a2);
  v3 = *(_DWORD *)(v2 + 72);
  if ( v3 & 4 )
  {
    v4 = 2;
  }
  else if ( v3 & 1 )
  {
    v4 = 4;
  }
  else
  {
    v4 = (v3 >> 3) & 1;
  }
  ((void (__fastcall *)(int, signed int, _DWORD))unk_1DCBC)(v2, 2048, 0);
  if ( v4 == 2 || (sub_184968(v2, 5, 1, 0), v4 != 1) )
    sub_184968(v2, 5, 8, 0); ← ☺
  ~~sub_184968(v2, 5, 16, 0);~~ ← ☺
  sub_184AC2(v2, 0);
  sub_184AEC(v2);
  sub_1853B4(v2, 2066, 488, 32, 32);
  sub_1853B4(v2, 2110, 488, 32, 32);
  sub_1853B4(v2, 2089, 488, 32, 32);
  sub_1853B4(v2, 2074, 488, 32, 32);
  sub_1853B4(v2, 2108, 488, 32, 32);
  sub_1853B4(v2, 2048, 3084, 0x40000, 0);
  sub_184968(v2, 6, 0x1000000, 0x1000000);
  sub_184968(v2, 6, 28160, 28160);
  JUMPOUT(&unk_1DCDC);
}
← ☺
```

bug Core

tion **Events**

e

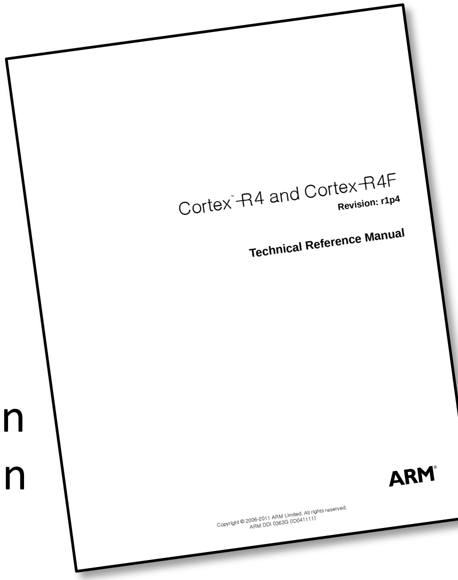
ain call:
xC5ACCE55;
ug registers

ps running ☺

Somewhen in between access to debug core breaks

abling interrupts:
xC5ACCE55;
ug registers

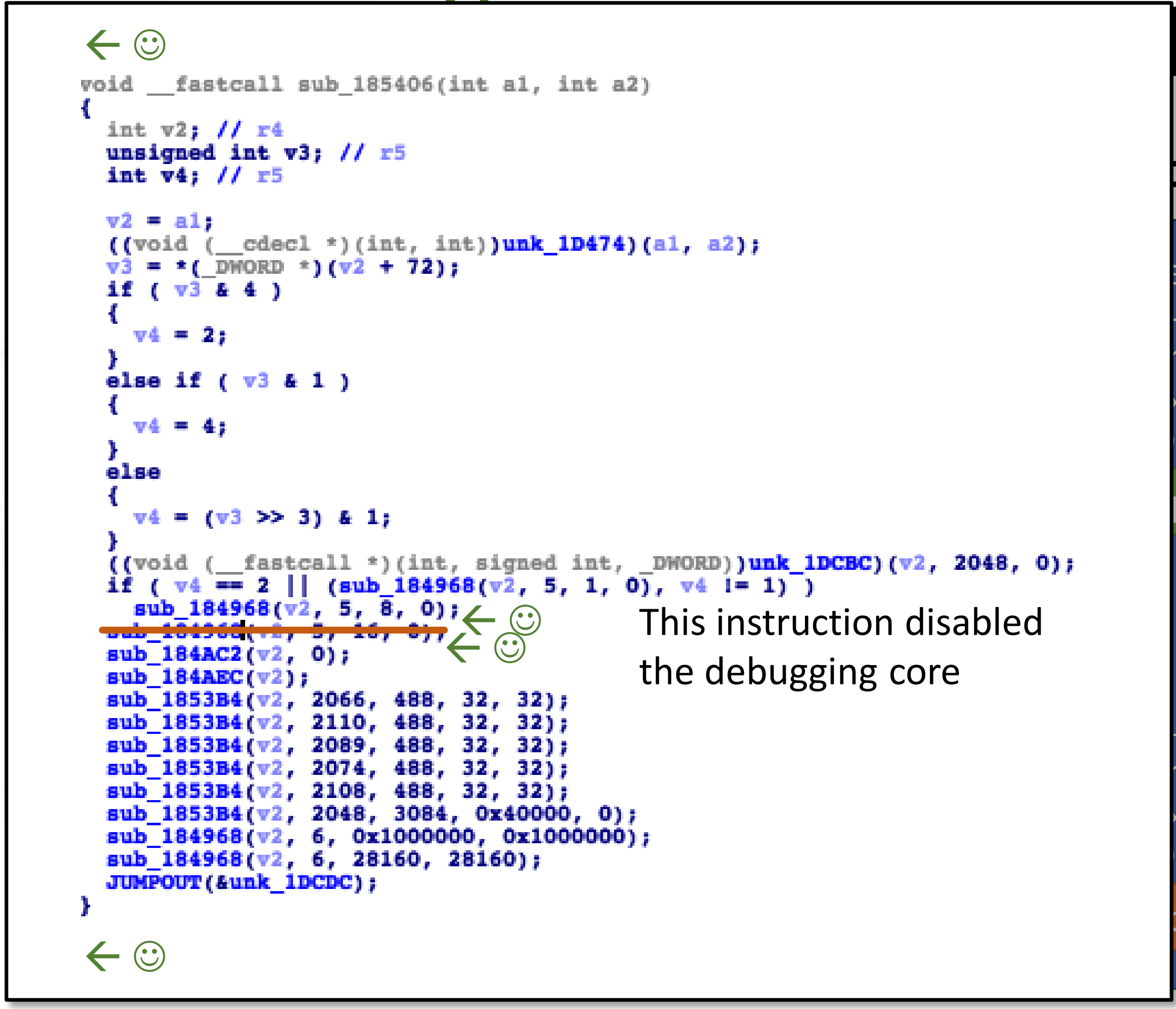
crashes ☹



➔ Why can‘t we access the debug registers?

~~Is the debug core even available?~~

bug Core



← ☺

```
void __fastcall sub_185406(int a1, int a2)
{
  int v2; // r4
  unsigned int v3; // r5
  int v4; // r5

  v2 = a1;
  ((void (__cdecl *)(int, int))unk_1D474)(a1, a2);
  v3 = *(_DWORD *)(v2 + 72);
  if ( v3 & 4 )
  {
    v4 = 2;
  }
  else if ( v3 & 1 )
  {
    v4 = 4;
  }
  else
  {
    v4 = (v3 >> 3) & 1;
  }
  ((void (__fastcall *)(int, signed int, _DWORD))unk_1DCBC)(v2, 2048, 0);
  if ( v4 == 2 || (sub_184968(v2, 5, 1, 0), v4 != 1) )
    sub_184968(v2, 5, 8, 0);  ← ☺
  ~~sub_184968(v2, 5, 16, 0);~~  ← ☺
  sub_184AC2(v2, 0);
  sub_184AEC(v2);
  sub_1853B4(v2, 2066, 488, 32, 32);
  sub_1853B4(v2, 2110, 488, 32, 32);
  sub_1853B4(v2, 2089, 488, 32, 32);
  sub_1853B4(v2, 2074, 488, 32, 32);
  sub_1853B4(v2, 2108, 488, 32, 32);
  sub_1853B4(v2, 2048, 3084, 0x40000, 0);
  sub_184968(v2, 6, 0x1000000, 0x1000000);
  sub_184968(v2, 6, 28160, 28160);
  JUMPOUT(&unk_1DCDC);
}
```

← ☺

This instruction disabled the debugging core

tion

e

ain call:
xC5ACCE55;
ug registers

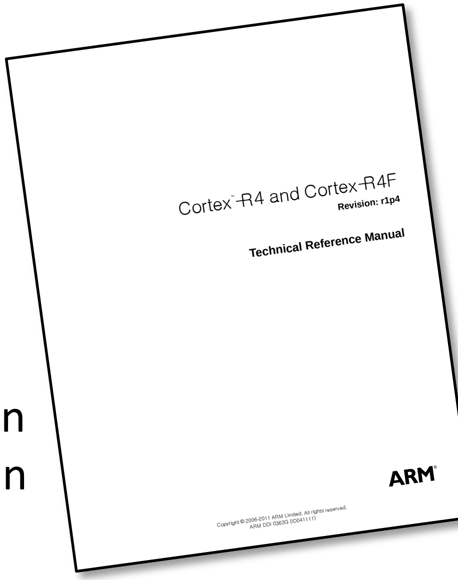
ps running ☺

abling interrupts:
xC5ACCE55;
ug registers

crashes ☹

Events

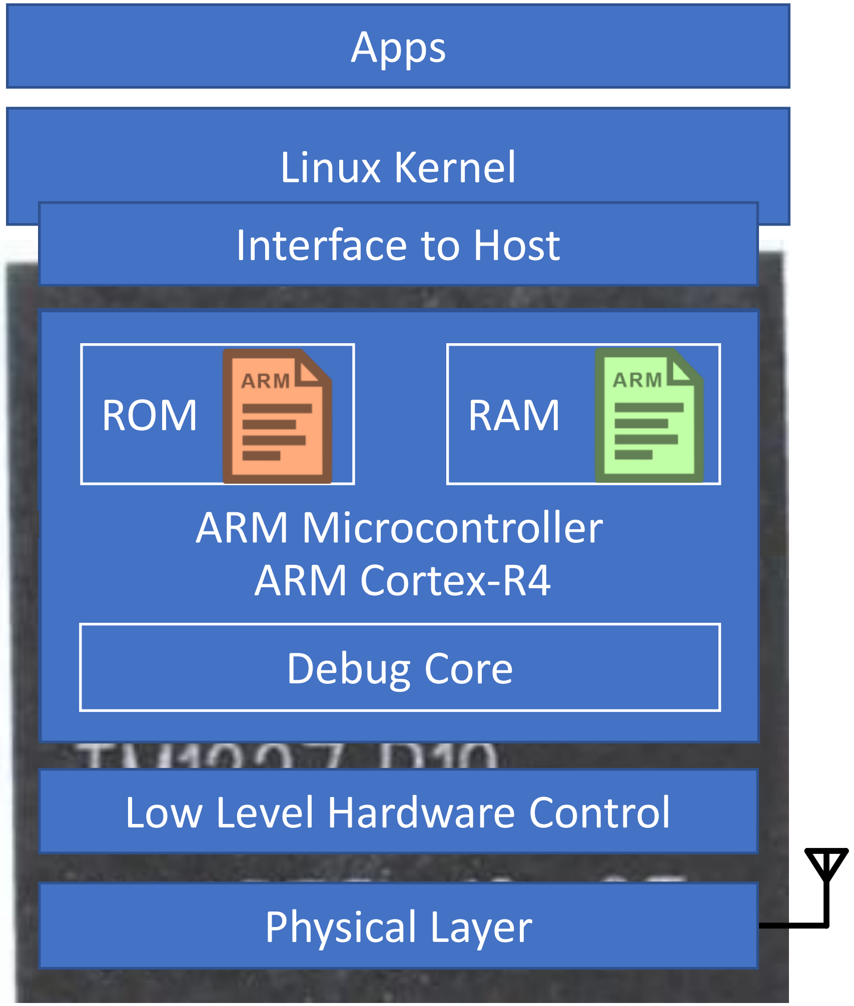
Somewhen in between access to debug core breaks



➔ **Why can‘t we access the debug registers?**

~~Is the debug core even available?~~

Accessing Debug Core



Apps

Linux Kernel

Interface to Host

ROM

RAM

ARM Microcontroller
ARM Cortex-R4

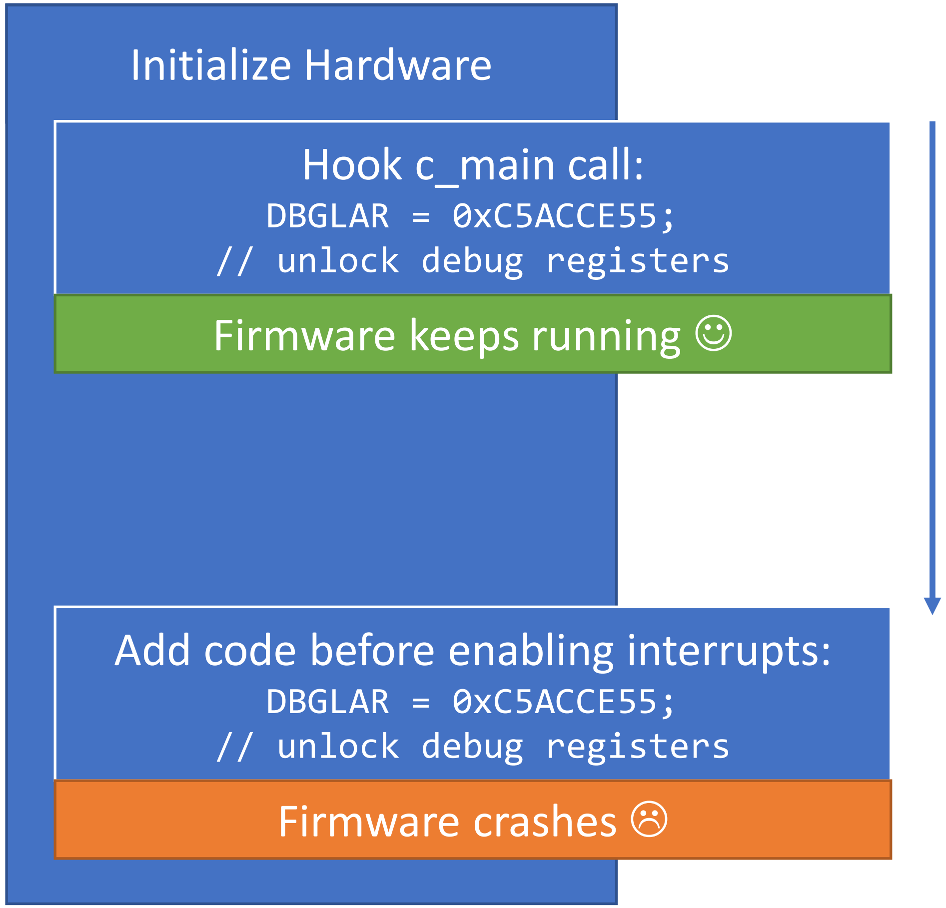
Debug Core

Low Level Hardware Control

Physical Layer

Firmware Execution

Events



Initialize Hardware

Hook c_main call:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

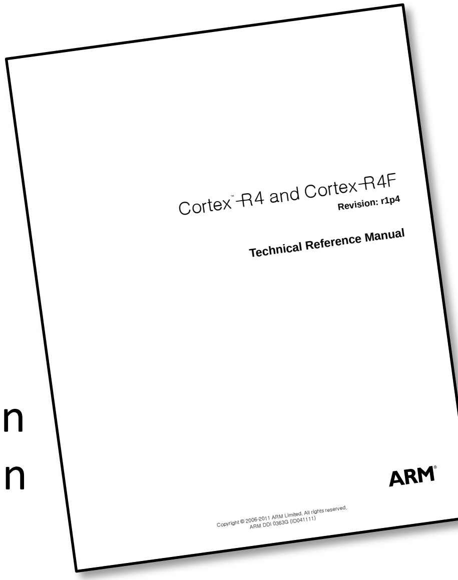
Firmware keeps running ☺

Add code before enabling interrupts:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware crashes ☹

Somewhen in between access to debug core breaks



Cortex-R4 and Cortex-R4F
Revision: r1p4

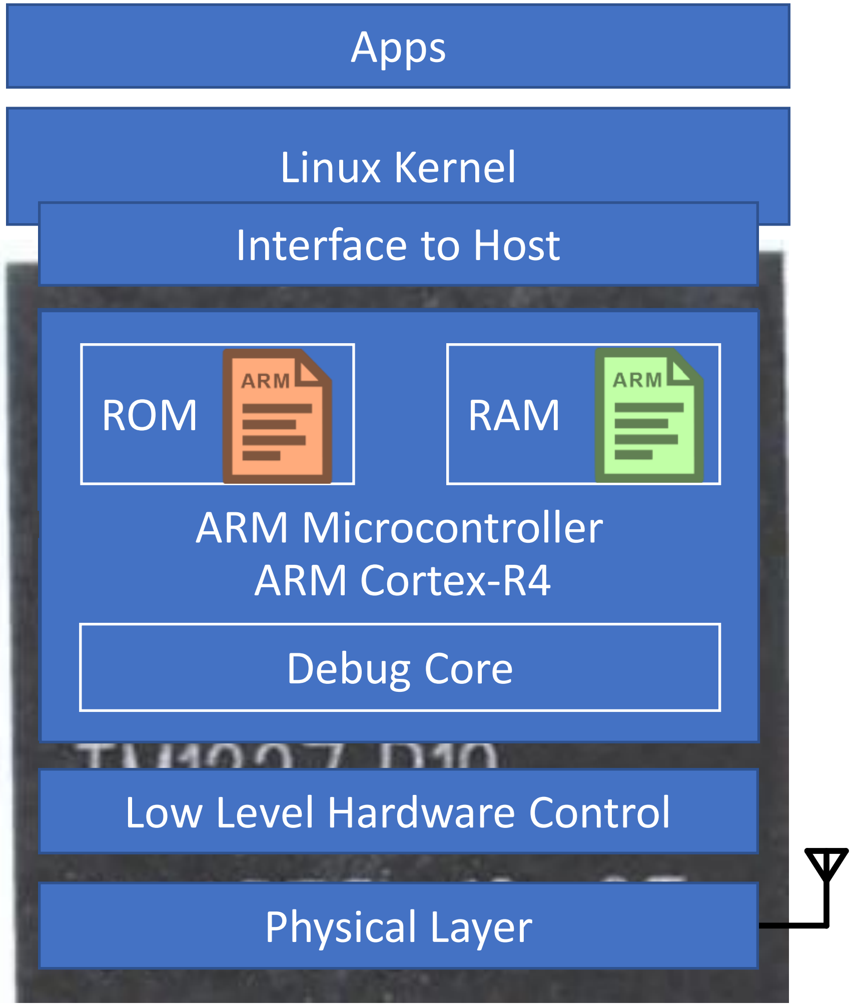
Technical Reference Manual

ARM

➔ **Why can‘t we access the debug registers?**

~~Is the debug core even available?~~

Accessing Debug Core



Firmware Execution

Events

Initialize Hardware

Hook c_main call:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware keeps running ☺

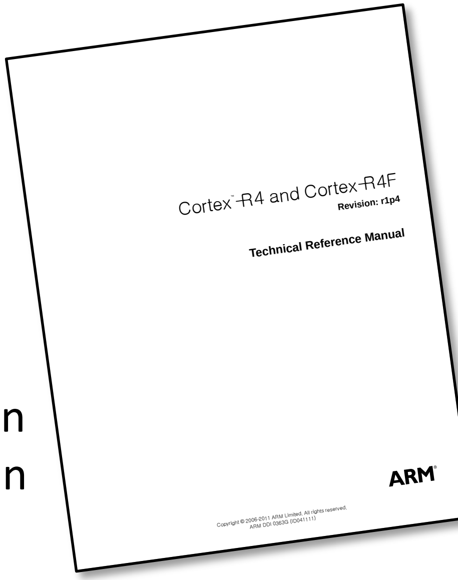
```
sub_184968(v2, 5, 16, 0);
```

Add code before enabling interrupts:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware crashes ☹

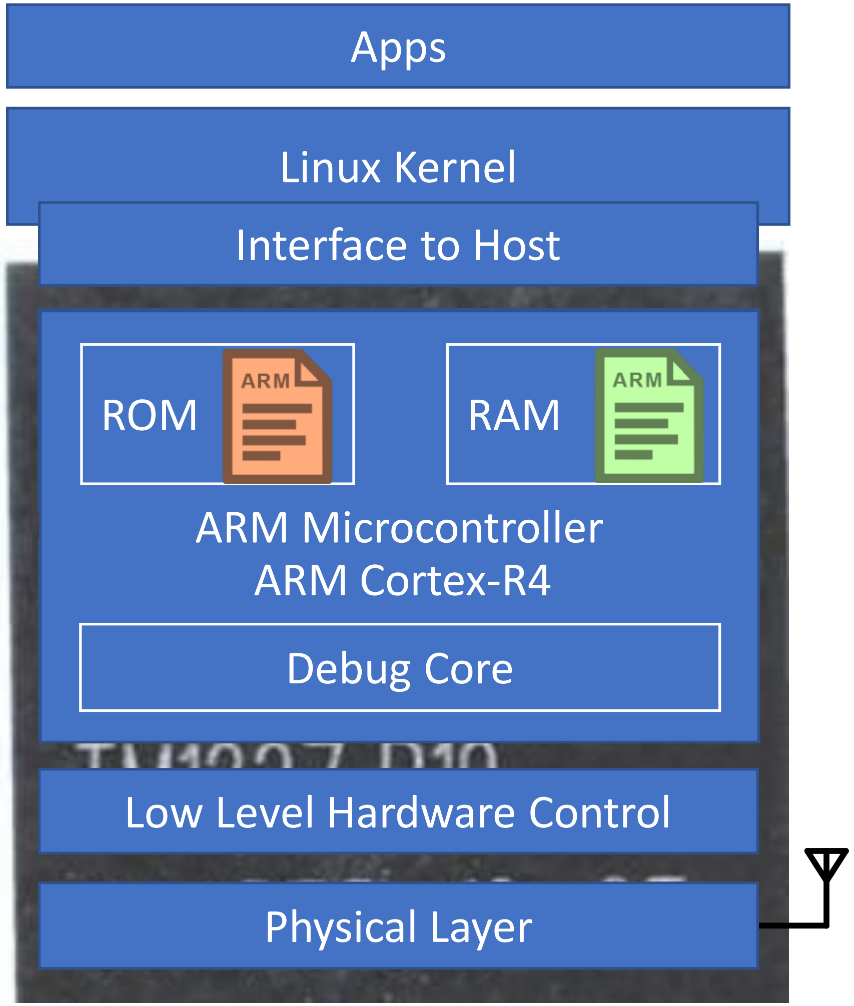
Somewhen in between access to debug core breaks


Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

➔ **Why can‘t we access the debug registers?**

~~Is the debug core even available?~~

Accessing Debug Core



Firmware Execution

Initialize Hardware

Hook c_main call:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware keeps running ☺

~~`sub_184968(v2, 5, 16, 0);`~~

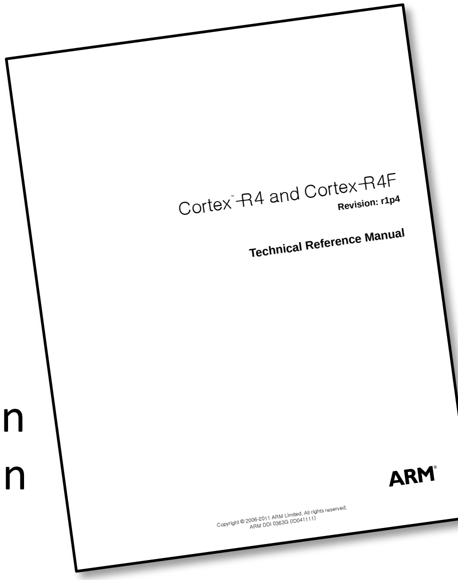
Add code before enabling interrupts:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware crashes ☹

Events

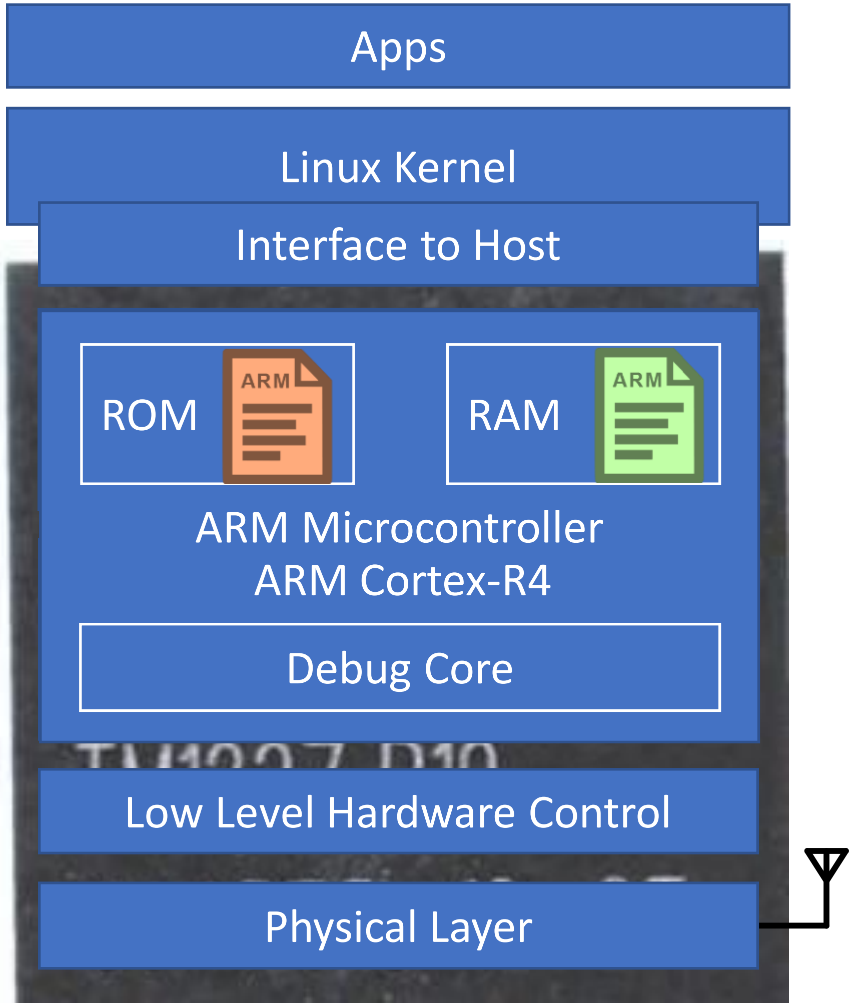
Somewhen in between access to debug core breaks


Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

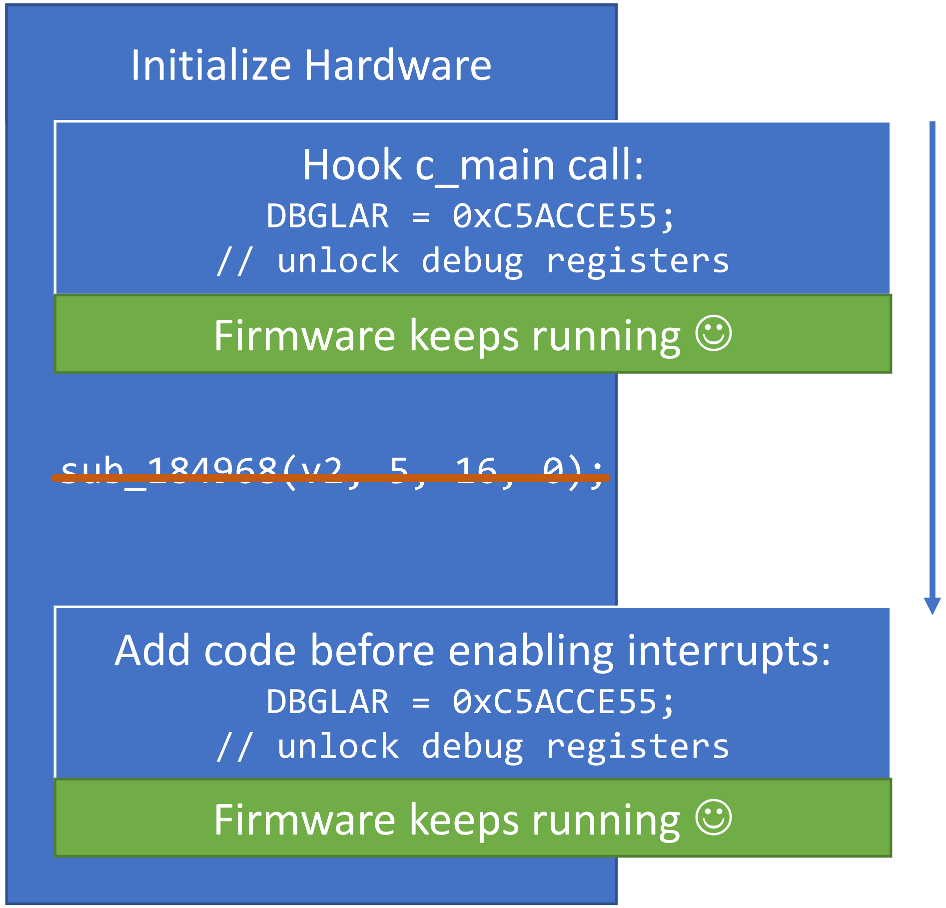
➔ **Why can‘t we access the debug registers?**

~~**Is the debug core even available?**~~

Accessing Debug Core



Firmware Execution **Events**



Initialize Hardware

Hook c_main call:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware keeps running ☺

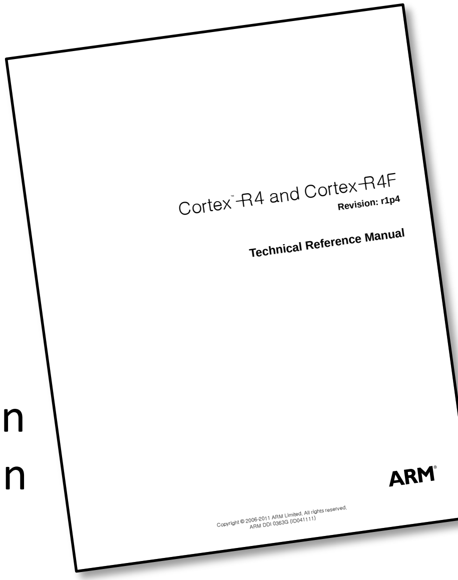
~~`sub_184968(v2, 5, 16, 0);`~~

Add code before enabling interrupts:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware keeps running ☺

Somewhen in between access to debug core breaks



Cortex™-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

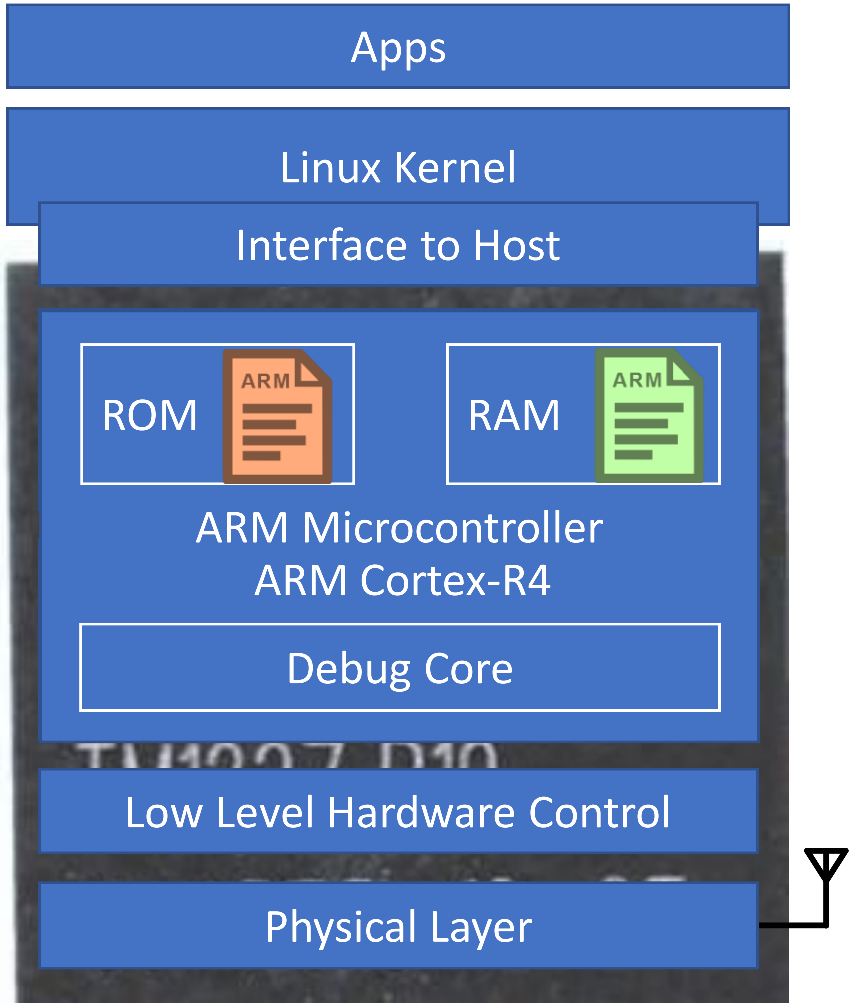
ARM®

Copyright © 2006-2011 ARM Limited. All rights reserved.
ARM DDI 0363G (ID041111)

➔ **Why can‘t we access the debug registers?**

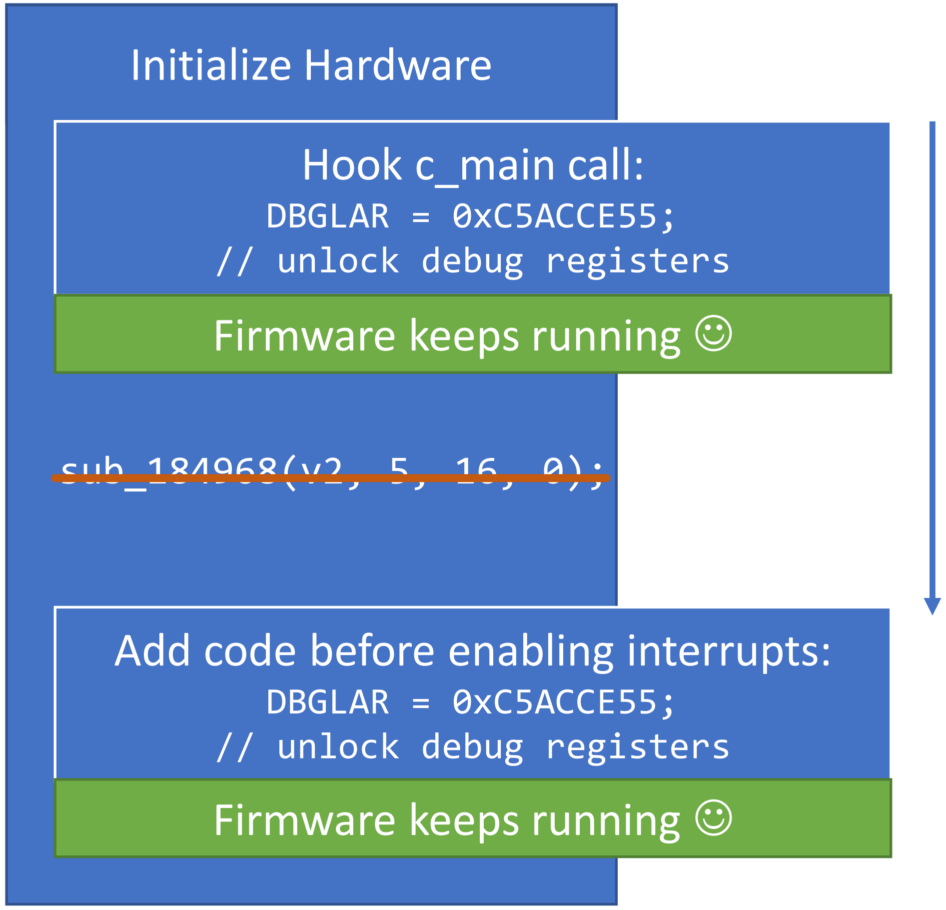
~~**Is the debug core even available?**~~

Accessing Debug Core



Firmware Execution

Events


Initialize Hardware

Hook c_main call:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware keeps running ☺

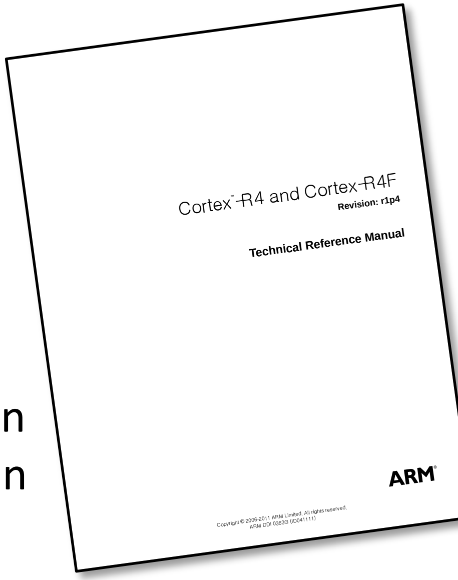
~~`sub_184968(v2, 5, 16, 0);`~~

Add code before enabling interrupts:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware keeps running ☺

Somewhen in between access to debug core breaks

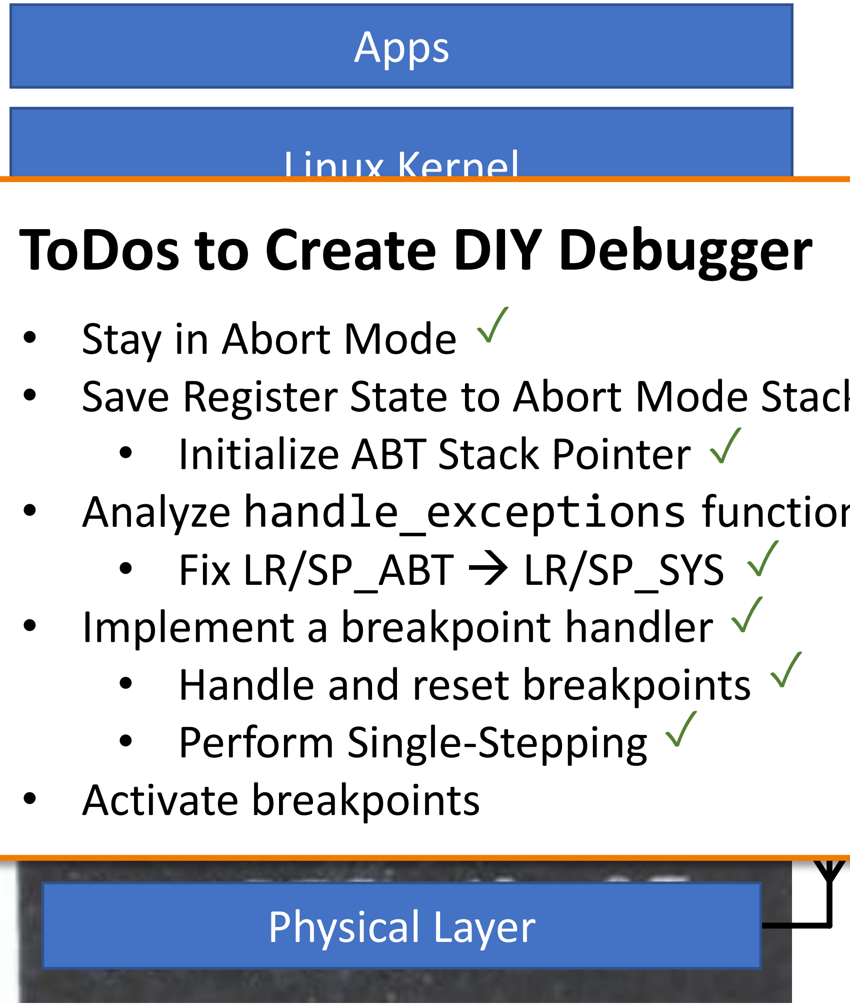

Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

➔ **We managed to reactivate access to the debugging core ☺.**

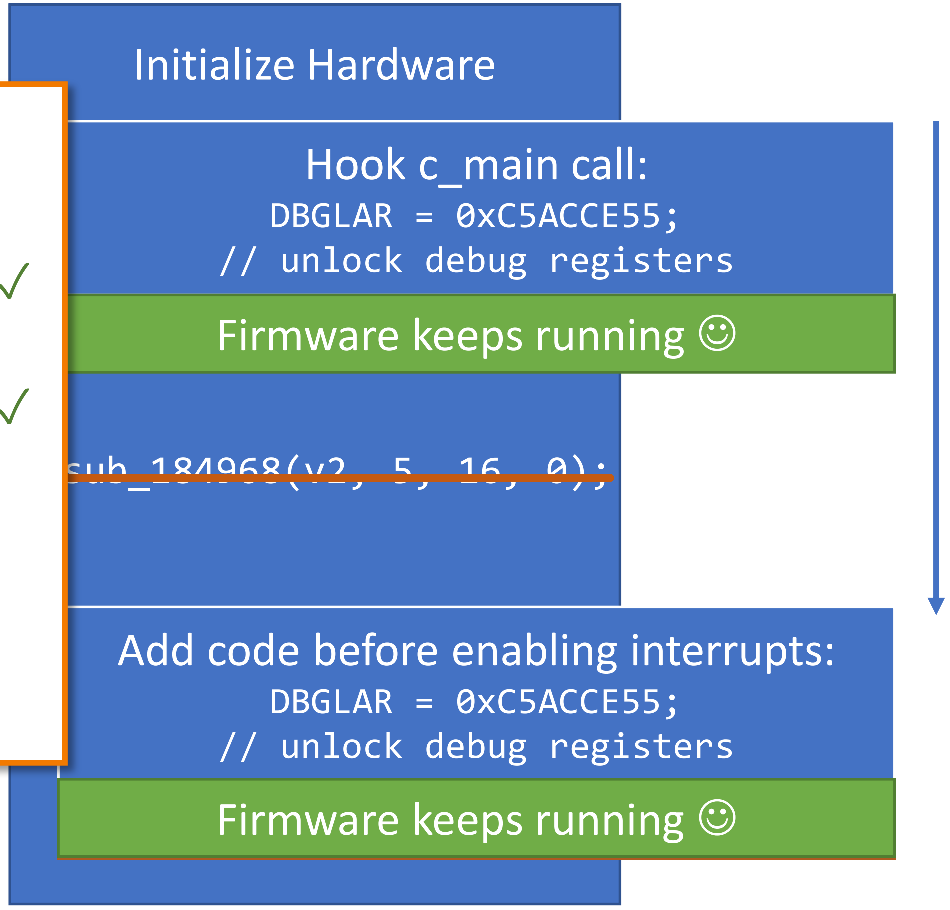
Accessing Debug Core



ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping ✓
- Activate breakpoints

Firmware Execution



Initialize Hardware

Hook c_main call:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware keeps running ☺

~~`sub_184968(v2, 5, 16, 0);`~~

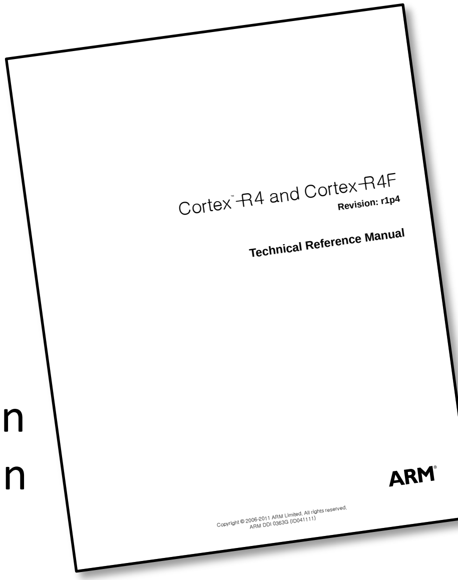
Add code before enabling interrupts:

```
DBGLAR = 0xC5ACCE55;
// unlock debug registers
```

Firmware keeps running ☺

Events

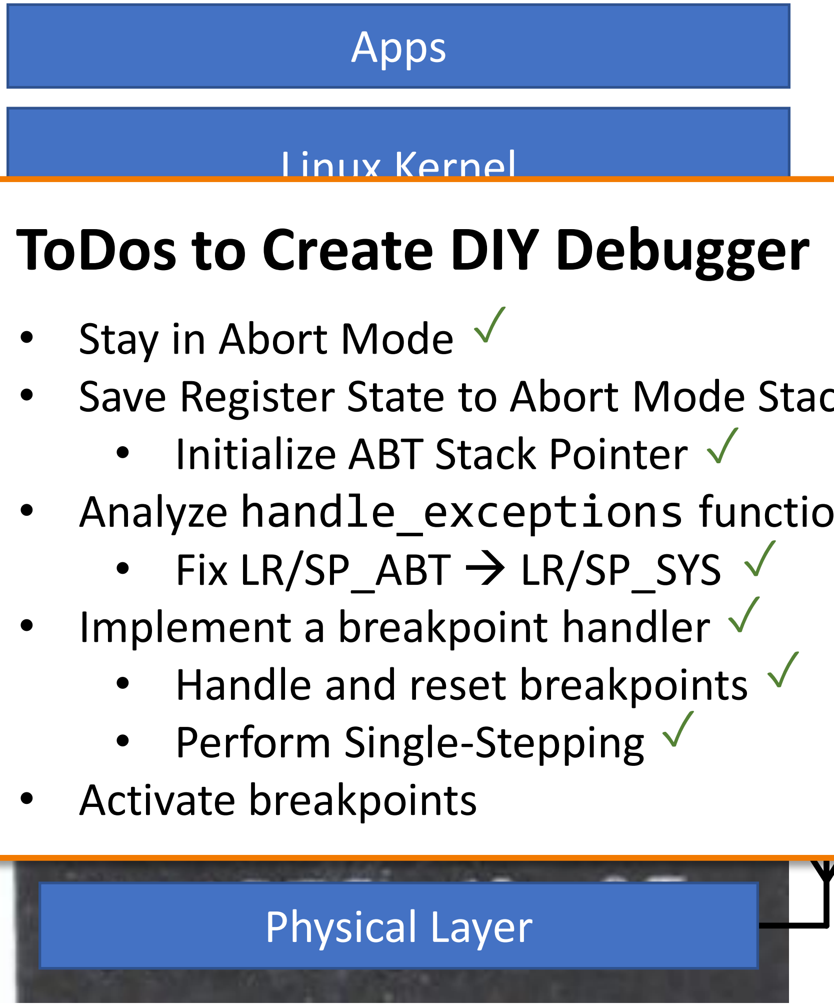
Somewhen in between access to debug core breaks



Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

→ We managed to reactivate access to the debugging core ☺.

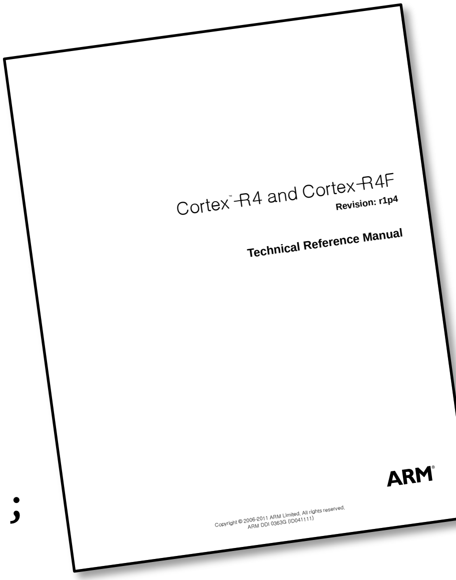
Activating Breakpoints



ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping ✓
- Activate breakpoints

```
// Called by c_main_hook
void set_debug_registers(void) {
    dbg_unlock_debug_registers();
    dbg_disable_breakpoint(0);
    dbg_disable_breakpoint(1);
    dbg_disable_breakpoint(2);
    dbg_disable_breakpoint(3);
    dbg_enable_monitor_mode_debugging();

    dbg_set_breakpoint_for_addr_match(0, 0x126f0);

    dbg_set_watchpoint_for_addr_match(0, 0x1FC2A4);
}
```

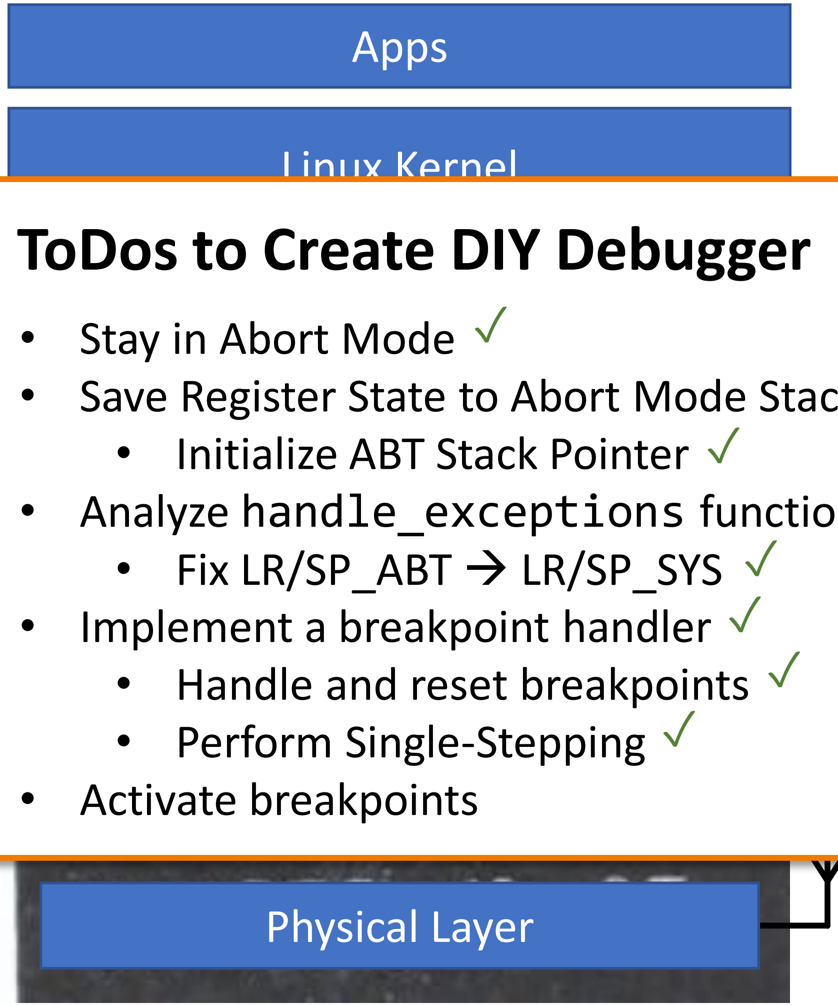


Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

Activating Breakpoints

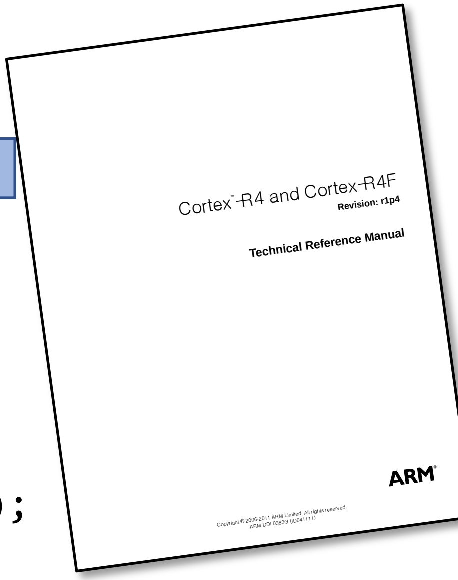


ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping ✓
- Activate breakpoints

```
// Called by c_main_hook
void set_debug_registers(void) {
    dbg_unlock_debug_registers();
    dbg_disable_breakpoint(0);
    dbg_disable_breakpoint(1);
    dbg_disable_breakpoint(2);
    dbg_disable_breakpoint(3);
    dbg_enable_monitor_mode_debugging();

    dbg_set_breakpoint_for_addr_match(0, 0x126f0);

    dbg_set_watchpoint_for_addr_match(0, 0x1FC2A4);
}

/* DBGLAR - Lock Access Register */
#define DBGLAR (*(volatile int *) (DBGBASE + 0xFB0))
#define DBGLAR_UNLOCK_CODE (0xC5ACCE55)

#define dbg_unlock_debug_registers() do { \
                   DBGLAR = DBGLAR_UNLOCK_CODE; \
} while (0)
```

Unlock access to debugging registers

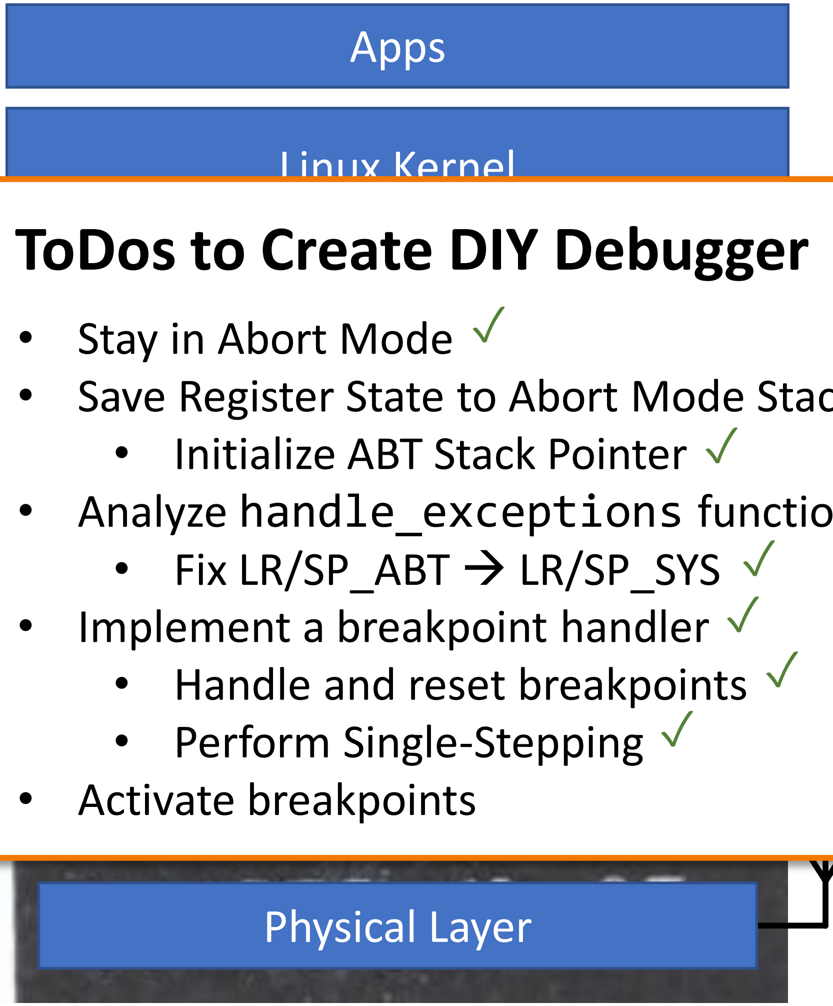


Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

Activating Breakpoints



ToDos to Create DIY Debugger

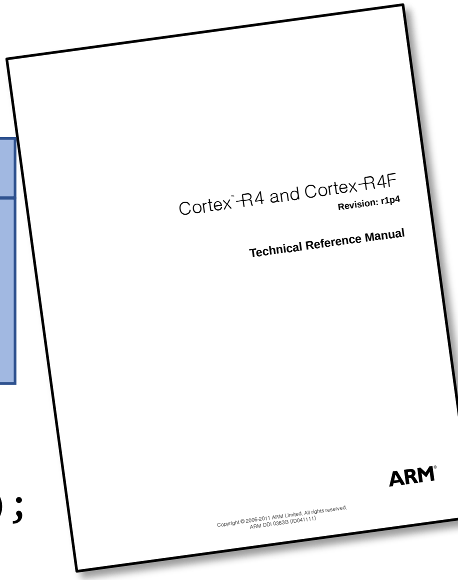
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping ✓
- Activate breakpoints

```
// Called by c_main_hook
void set_debug_registers(void) {
    dbg_unlock_debug_registers();
    dbg_disable_breakpoint(0);
    dbg_disable_breakpoint(1);
    dbg_disable_breakpoint(2);
    dbg_disable_breakpoint(3);
    dbg_enable_monitor_mode_debugging();

    dbg_set_breakpoint_for_addr_match(0, 0x126f0);

    dbg_set_watchpoint_for_addr_match(0, 0x1FC2A4);
}
```

Unlock access to debugging registers

Disable all four hardware breakpoints

```
#define dbg_disable_breakpoint(number) do { \
    DBGBCR ## number = UPDATE_DBG_REG(DBGBCR ## number, \
    GET_DBG_MASK(DBGBCR_E), SET_DBG_VALUE(DBGBCR_E, \
    DBGBCR_E_DISABLED)); \
} while (0)
```



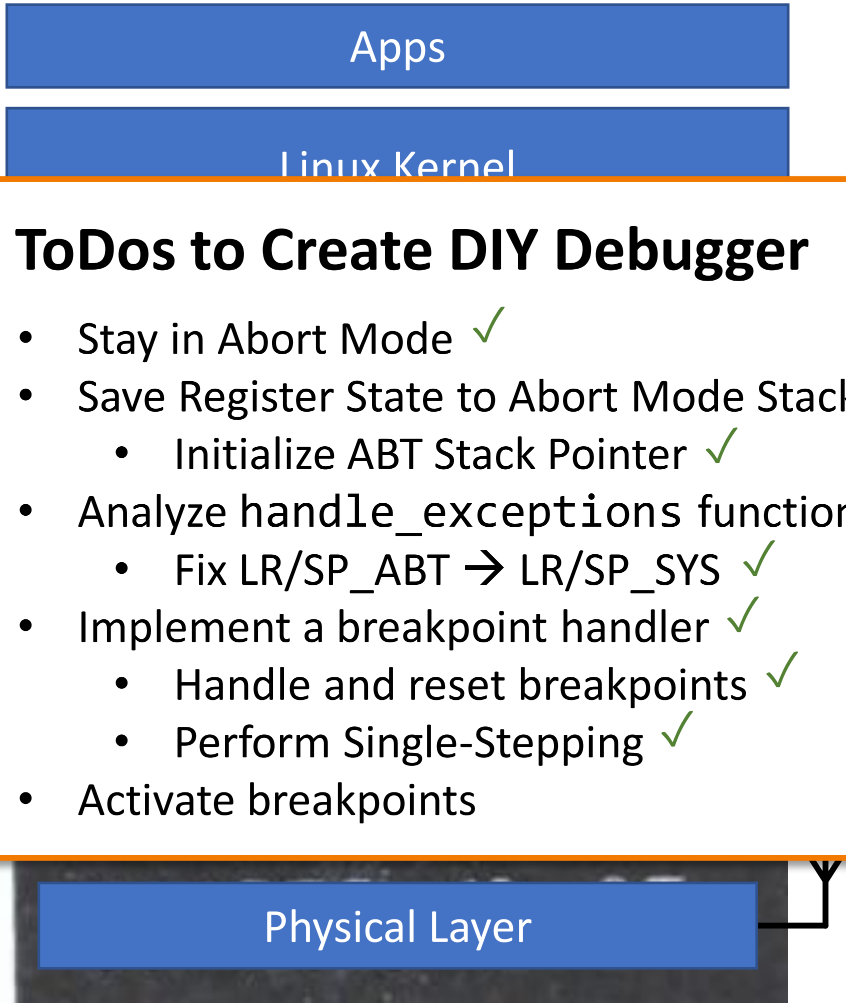
Cortex-R4 and Cortex-R4F

Revision: r1p4

Technical Reference Manual

ARM

Activating Breakpoints



ToDos to Create DIY Debugger

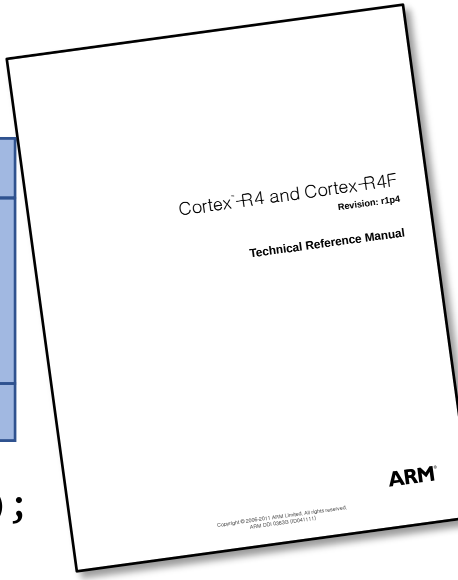
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
    - Handle and reset breakpoints ✓
    - Perform Single-Stepping ✓
- Activate breakpoints

```
// Called by c_main_hook
void set_debug_registers(void) {
    dbg_unlock_debug_registers();
    dbg_disable_breakpoint(0);
    dbg_disable_breakpoint(1);
    dbg_disable_breakpoint(2);
    dbg_disable_breakpoint(3);
    dbg_enable_monitor_mode_debugging();

    dbg_set_breakpoint_for_addr_match(0, 0x126f0);

    dbg_set_watchpoint_for_addr_match(0, 0x1FC2A4);
}

#define dbg_enable_monitor_mode_debugging() do { \
    DBGDSCR = UPDATE_DBG_REG(DBGDSCR, \
    GET_DBG_MASK(DBGDSCR_MDBGen), \
    SET_DBG_VALUE(DBGDSCR_MDBGen, \
    DBGDSCR_MDBGen_ENABLED)); \
} while (0)
```

Unlock access to debugging registers

Disable all four hardware breakpoints

Enable monitor-debug mode


Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

Activating Breakpoints

Apps

Linux Kernel

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
    - Handle and reset breakpoints ✓
    - Perform Single-Stepping ✓
- Activate breakpoints

Physical Layer

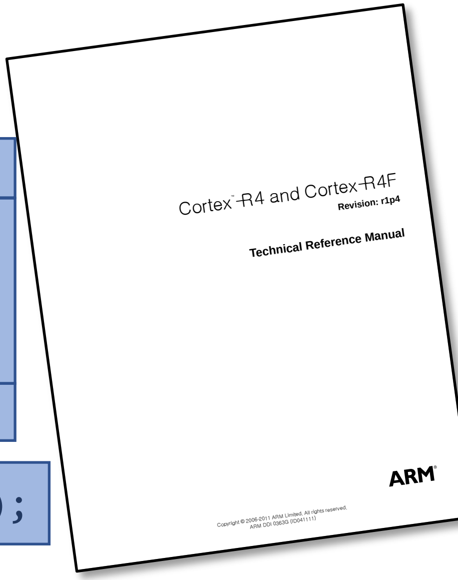
```
// Called by c_main_hook
void set_debug_registers(void) {
    dbg_unlock_debug_registers();
    dbg_disable_breakpoint(0);
    dbg_disable_breakpoint(1);
    dbg_disable_breakpoint(2);
    dbg_disable_breakpoint(3);
    dbg_enable_monitor_mode_debugging();

    dbg_set_breakpoint_for_addr_match(0, 0x1[illegible]0);

    dbg_set_watchpoint_for_addr_match(0, 0x1FC2A4);
}
```

Unlock access to debugging registers

Disable all four hardware breakpoints

Enable monitor-debug mode

Set breakpoint at beginning of printf function



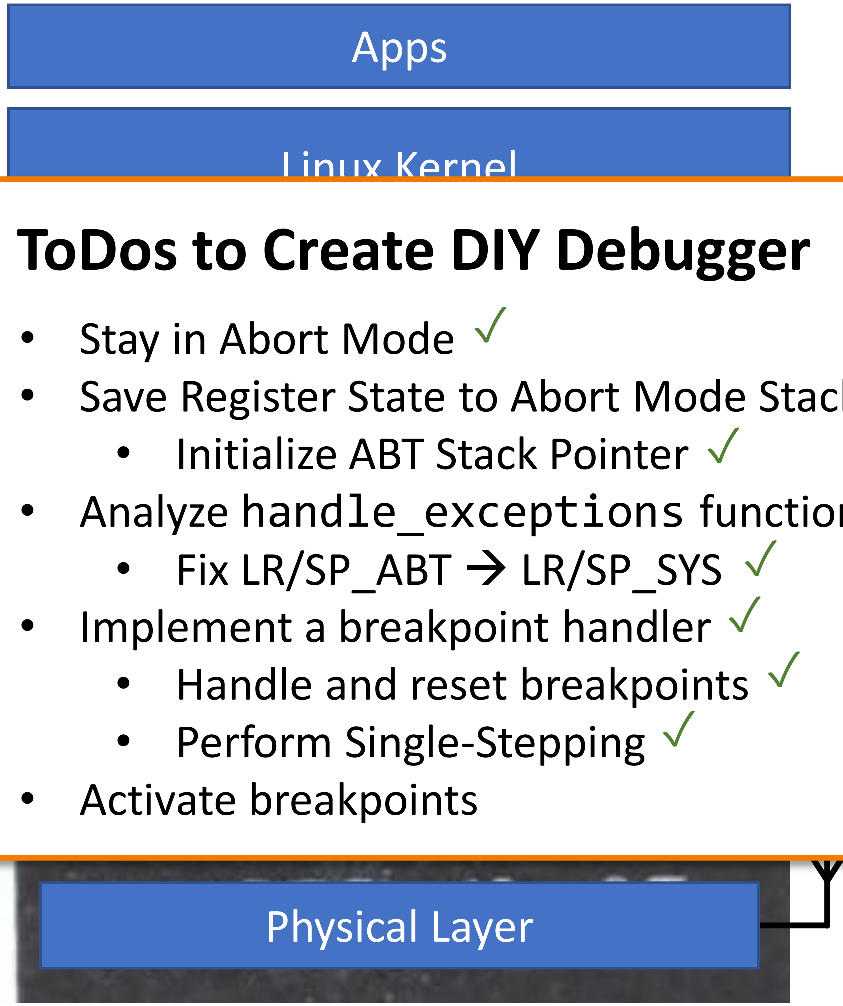
Cortex-R4 and Cortex-R4F
Revision: r1p4

Technical Reference Manual

ARM

```
#define dbg_set_breakpoint_for_addr_match(number, address) do { \
    DBGBCR ## number = 0x0; \
    DBGBVR ## number = (address) & DBGBVR_ADDRMASK; \
    DBGBCR ## number = \
        SET_DBG_VALUE(DBGBCR_BT, DBGBCR_BT_UNLINKED_INSTR_ADDR_MATCH) |  \
        SET_DBG_VALUE(DBGBCR_MASK, DBGBCR_MASK_NO_MASK) |  \
        SET_DBG_VALUE(DBGBCR_E, DBGBCR_E_ENABLED) | \
        SET_DBG_VALUE(DBGBCR_SSC_HMC_PMC, DBGBCR_SSC_HMC_PMC__PL0_SUP_SYS) |  \
        SET_DBG_VALUE(DBGBCR_BAS, GET_BAS_FOR_THUMB_ADDR(address)); \
} while (0)
```

Activating Breakpoints



ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
    - Handle and reset breakpoints ✓
    - Perform Single-Stepping ✓
- Activate breakpoints

```
// Called by c_main_hook
void set_debug_registers(void) {
    dbg_unlock_debug_registers();
    dbg_disable_breakpoint(0);
    dbg_disable_breakpoint(1);
    dbg_disable_breakpoint(2);
    dbg_disable_breakpoint(3);
    dbg_enable_monitor_mode_debugging();

    dbg_set_breakpoint_for_addr_match(0, 0x126f0);

    dbg_set_watchpoint_for_addr_match(0, 0x1FC2A4);
}
```

Unlock access to debugging registers

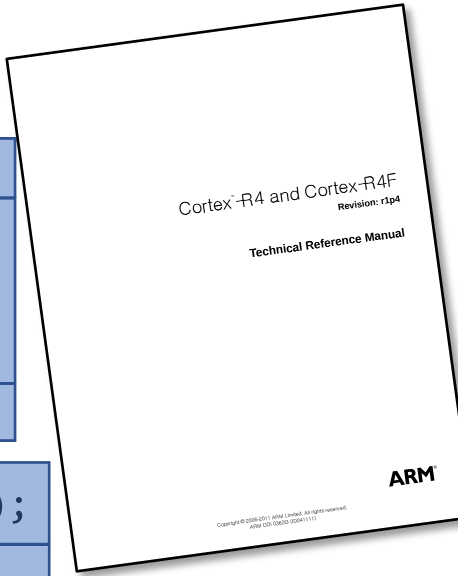
Disable all four hardware breakpoints

Enable monitor-debug mode

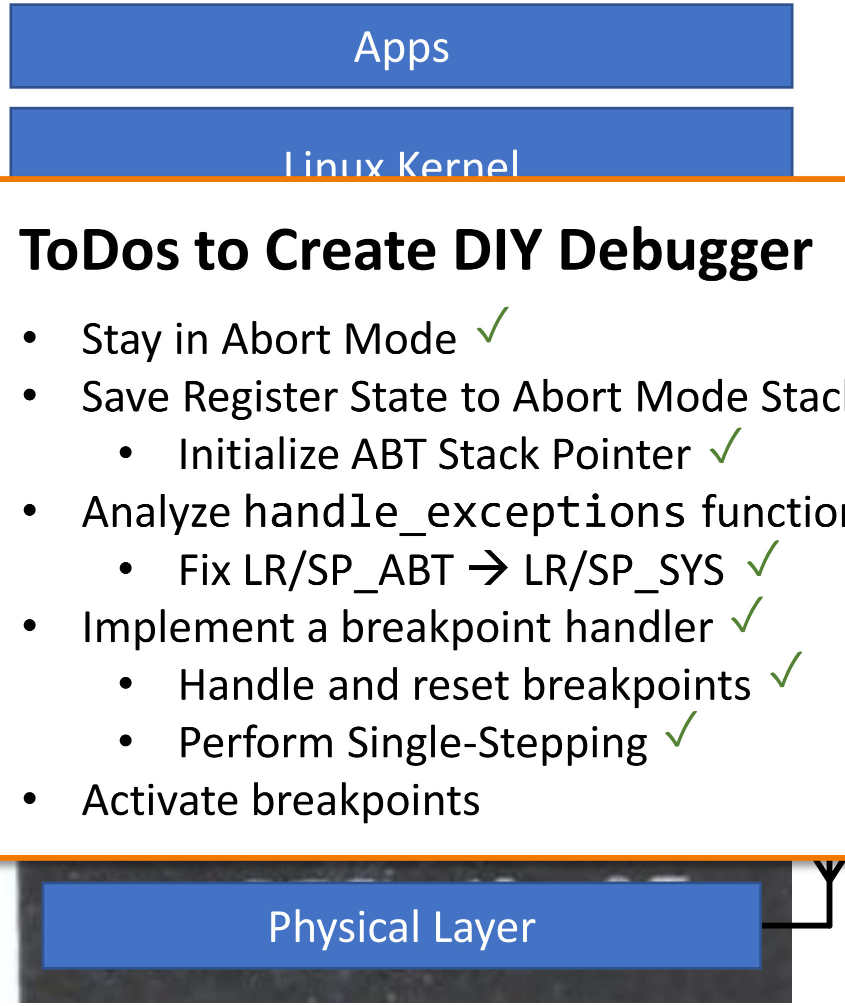
Set breakpoint at beginning of printf function

Set memory watchpoint on address of
“%s: Broadcom SDPCMD CDC driver“

```
#define dbg_set_watchpoint_for_addr_match(number, address) do { \
    DBGWCR ## number = 0x0; \
    DBGWVR ## number = (address) & DBGWVR_ADDRMASK; \
    DBGWCR ## number = \
        SET_DBG_VALUE(DBGWCR_WT, DBGWCR_WT_UNLINKED_DATA_ADDR_MATCH) | \
        SET_DBG_VALUE(DBGWCR_MASK, DBGWCR_MASK_NO_MASK) | \
        SET_DBG_VALUE(DBGWCR_E, DBGWCR_E_ENABLED) | \
        SET_DBG_VALUE(DBGWCR_SSC_HMC_PAC, DBGWCR_SSC_HMC_PAC__ALL) | \
        SET_DBG_VALUE(DBGWCR_LSC, DBGWCR_LSC_MATCH_ALL) | \
        SET_DBG_VALUE(DBGWCR_BAS_4BIT, 0xF); \
} while (0)
```


Cortex-R4 and Cortex-R4F
Revision: r1p4
Technical Reference Manual
ARM

Activating Breakpoints



ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping ✓
- Activate breakpoints

Set breakpoint at beginning of printf function

Set memory watchpoint on address of
“%s: Broadcom SDPCMD CDC driver“

```
RTE (USB-SDIO-CDC) 6.37.32.RC23.34.43 (r639704) on BCM4339 r1 @ 37.4/161.3/161.3MHz
000000.010    WP hit pc=00012b3a
000000.013    WP hit pc=00012b3a
000000.010 sdpcmdcdc0: Broadcom SDPCMD CDC driver
000000.141 reclaim section 0: Returned 31688 bytes to the heap
000000.189 nexmon_ver: 63fb-dirty-14
000000.192 wl_nd_ra_filter_init: Enter..
000000.196 TCAM: 256 used: 198 exceed:0
000000.200    WP hit pc=000126c2
000000.203 reclaim section 1: Returned 71844 bytes to the heap
000000.208    BP0 step 0: pc=000126f0 *r1=sdpcmd_dpc
000000.213    BP0 step 1: pc=000126f2
000000.216    BP0 step 2: pc=000126f4
000000.219    BP0 step 3: pc=000126f6
000000.223    BP0 step 4: pc=000126fa
000000.226    BP0 single-stepping done
000000.229 sdpcmd_dpc: Enable
000000.234 wl0: wlc_bmac_ucodembss_hwcap: Insuff mem for MBSS: templ memblks 192 fifo ...
000000.249 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
000000.295 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
000000.303 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
```

Activating Breakpoints

Apps

Linux Kernel

ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping ✓
- Activate breakpoints

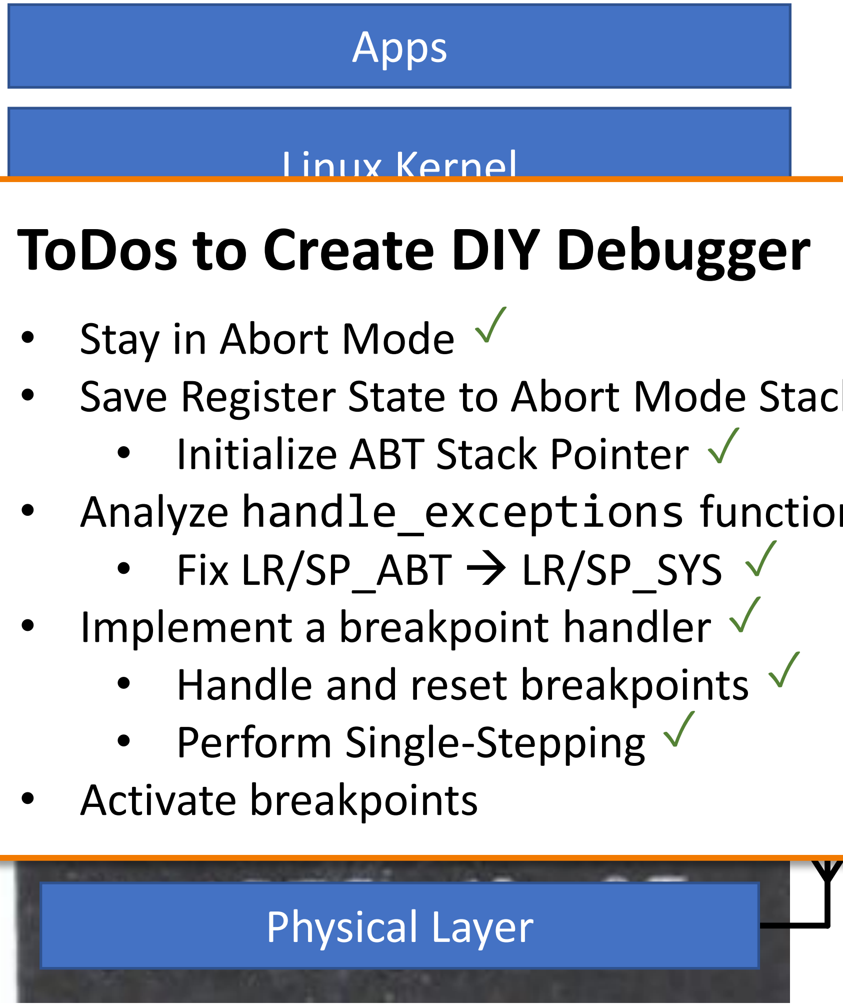
Physical Layer

Set breakpoint at beginning of printf function

Set memory watchpoint on address of
“%s: Broadcom SDPCMD CDC driver“

```
RTE (USB-SDIO-CDC) 6.37.32.RC23.34.43 (r639704) on BCM4339 r1 @ 37.4/161.3/161.3MHz
000000.010    WP hit pc=00012b3a
000000.013    WP hit pc=00012b3a
000000.010 sdpcmdcdc0: Broadcom SDPCMD CDC driver
000000.141 reclaim section 0: Returned 31688 bytes to the heap
000000.189 nexmon_ver: 63fb-dirty-14
000000.192 wl_nd_ra_filter_init: Enter..
000000.196 TCAM: 256 used: 198 exceed:0
000000.200    WP hit pc=000126c2
000000.203 reclaim section 1: Returned 71844 bytes to the heap
000000.208    BP0 step 0: pc=000126f0 *r1=sdpcmd_dpc
000000.213    BP0 step 1: pc=000126f2
000000.216    BP0 step 2: pc=000126f4
000000.219    BP0 step 3: pc=000126f6
000000.223    BP0 step 4: pc=000126fa
000000.226    BP0 single-stepping done
000000.229 sdpcmd_dpc: Enable
000000.234 wl0: wlc_bmac_ucodembss_hwcap: Insuff mem for MBSS: templ memblks 192 fifo ...
000000.249 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
000000.295 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
000000.303 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
```

Activating Breakpoints



ToDos to Create DIY Debugger

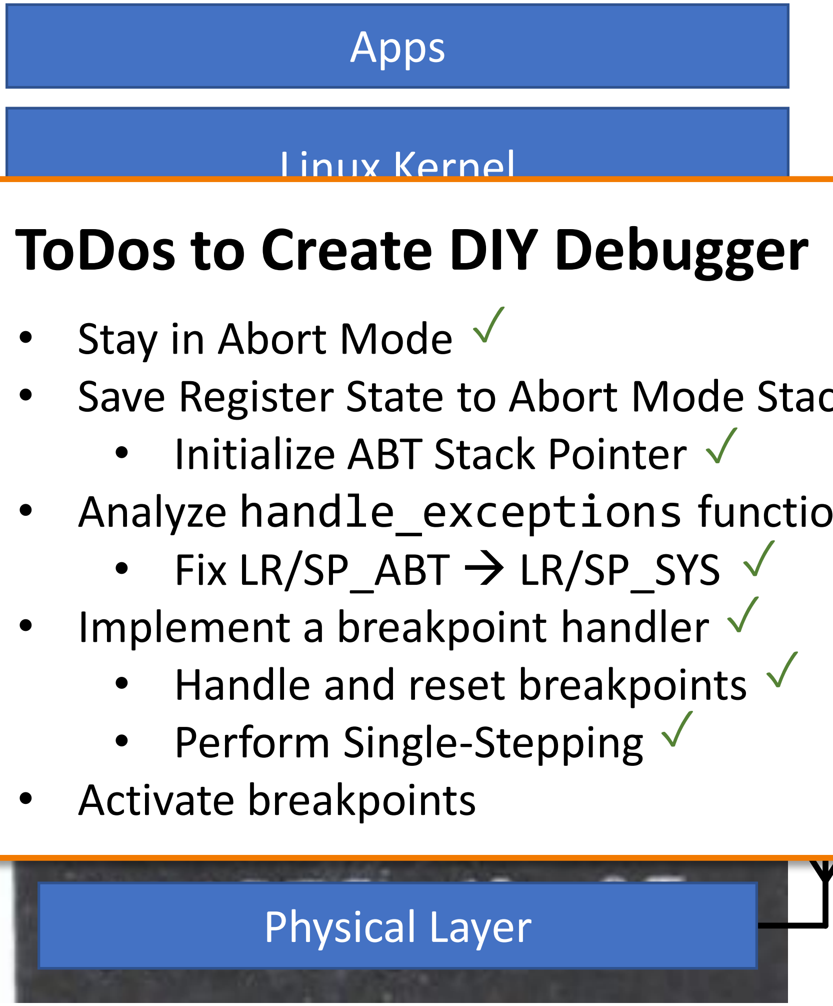
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping ✓
- Activate breakpoints

Set breakpoint at beginning of printf function

Set memory watchpoint on address of “%s: Broadcom SDPCMD CDC driver“

```
RTE (USB-SDIO-CDC) 6.37.32.RC23.34.43 (r639704) on BCM4339 r1 @ 37.4/161.3/161.3MHz
000000.010    WP hit pc=00012b3a
000000.013    WP hit pc=00012b3a
000000.010 sdpcmdcdc0: Broadcom SDPCMD CDC driver
000000.141 reclaim section 0: Returned 31688 bytes to the heap
000000.189 nexmon_ver: 63fb-dirty-14
000000.192 wl_nd_ra_filter_init: Enter..
000000.196 TCAM: 256 used: 198 exceed:0
000000.200    WP hit pc=000126c2
000000.203 reclaim section 1: Returned 71844 bytes to the heap
000000.208    BP0 step 0: pc=000126f0 *r1=sdpcmd_dpc
000000.213    BP0 step 1: pc=000126f2
000000.216    BP0 step 2: pc=000126f4
000000.219    BP0 step 3: pc=000126f6
000000.223    BP0 step 4: pc=000126fa
000000.226    BP0 single-stepping done
000000.229 sdpcmd_dpc: Enable
000000.234 wl0: wlc_bmac_ucodembss_hwcap: Insuff mem for MBSS: templ memblks 192 fifo ...
000000.249 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
000000.295 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
000000.303 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
```

Activating Breakpoints



ToDos to Create DIY Debugger

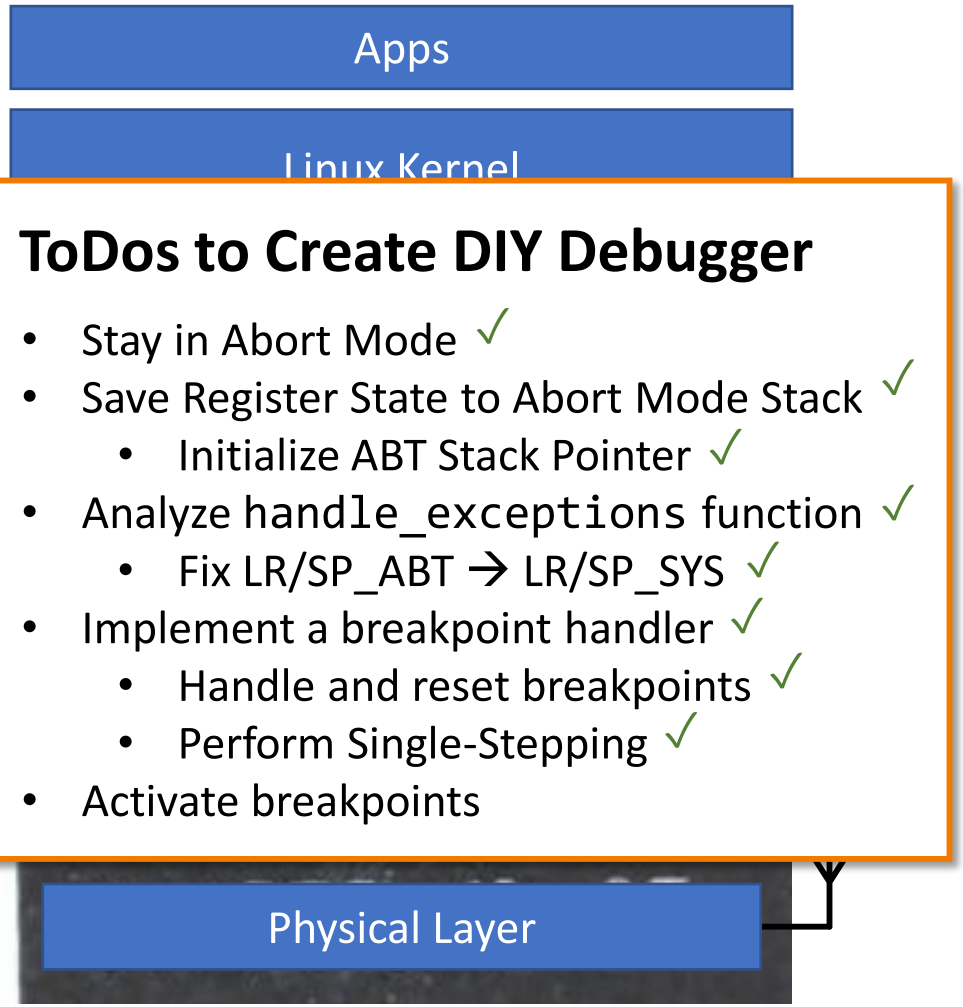
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
    - Handle and reset breakpoints ✓
    - Perform Single-Stepping ✓
- Activate breakpoints

Set breakpoint at beginning of printf function

Set memory watchpoint on address of
“%s: Broadcom SDPCMD CDC driver“

```
RTE (USB-SDIO-CDC) 6.37.32.RC23.34.43 (r639704) on BCM4339 r1 @ 37.4/161.3/161.3MHz
000000.010    WP hit pc=00012b3a
000000.013    WP hit pc=00012b3a
000000.010 sdpcmdcdc0: Broadcom SDPCMD CDC driver
000000.141 reclaim section 0: Returned 31688 bytes to the heap
000000.189 nexmon_ver: 63fb-dirty-14
000000.192 wl_nd_ra_filter_init: Enter..
000000.196 TCAM: 256 used: 198 exceed:0
000000.200    WP hit pc=000126c2
000000.203 reclaim section 1: Returned 71844 bytes to the heap
000000.208    BP0 step 0: pc=000126f0 *r1=sdpcmd_dpc
000000.213    BP0 step 1: pc=000126f2
000000.216    BP0 step 2: pc=000126f4
000000.219    BP0 step 3: pc=000126f6
000000.223    BP0 step 4: pc=000126fa
000000.226    BP0 single-stepping done
000000.229 sdpcmd_dpc: Enable
000000.234 wl0: wlc_bmac_ucodembss_hwcap: Insuff mem for MBSS: templ memblks 192 fifo ...
000000.249 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
000000.295 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
000000.303 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
```

Activating Breakpoints



ToDos to Create DIY Debugger

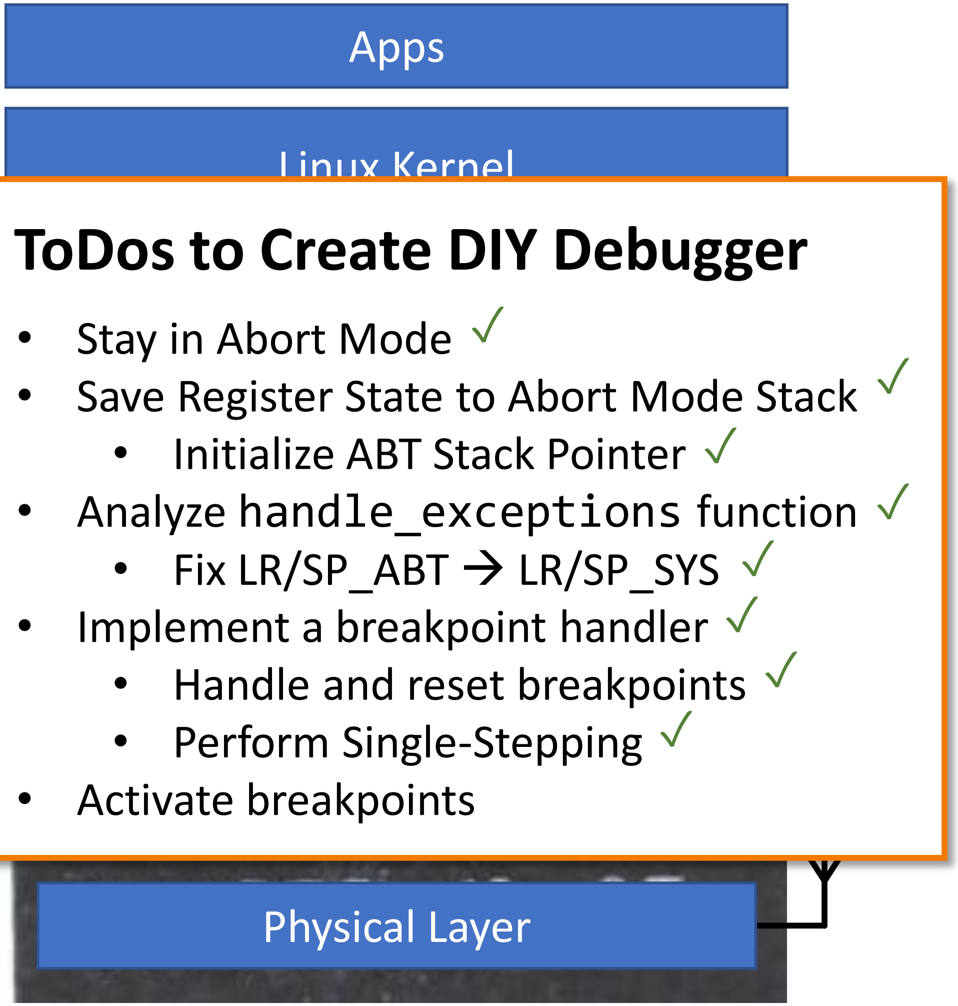
- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
    - Handle and reset breakpoints ✓
    - Perform Single-Stepping ✓
- Activate breakpoints

Set breakpoint at beginning of printf function

Set memory watchpoint on address of “%s: Broadcom SDPCMD CDC driver“

```
RTE (USB-SDIO-CDC) 6.37.32.RC23.34.43 (r639704) on BCM4339 r1 @ 37.4/161.3/161.3MHz
000000.010    WP hit pc=00012b3a
000000.013    WP hit pc=00012b3a
000000.010 sdpcmdcdc0: Broadcom SDPCMD CDC driver
000000.141 reclaim section 0: Returned 31688 bytes to the heap
000000.189 nexmon_ver: 63fb-dirty-14
000000.192 wl_nd_ra_filter_init: Enter..
000000.196 TCAM: 256 used: 198 exceed:0
000000.200    WP hit pc=000126c2
000000.203 reclaim section 1: Returned 71844 bytes to the heap
000000.208    BP0 step 0: pc=000126f0 *r1=sdpcmd_dpc
000000.213    BP0 step 1: pc=000126f2
000000.216    BP0 step 2: pc=000126f4
000000.219    BP0 step 3: pc=000126f6
000000.223    BP0 step 4: pc=000126fa
000000.226    BP0 single-stepping done
000000.229 sdpcmd_dpc: Enable
000000.234 wl0: wlc_bmac_ucodembss_hwcap: Insuff mem for MBSS: templ memblks 192 fifo ...
000000.249 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
000000.295 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
000000.303 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
```

Activating Breakpoints



ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
    - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
    - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
    - Handle and reset breakpoints ✓
    - Perform Single-Stepping ✓
- Activate breakpoints

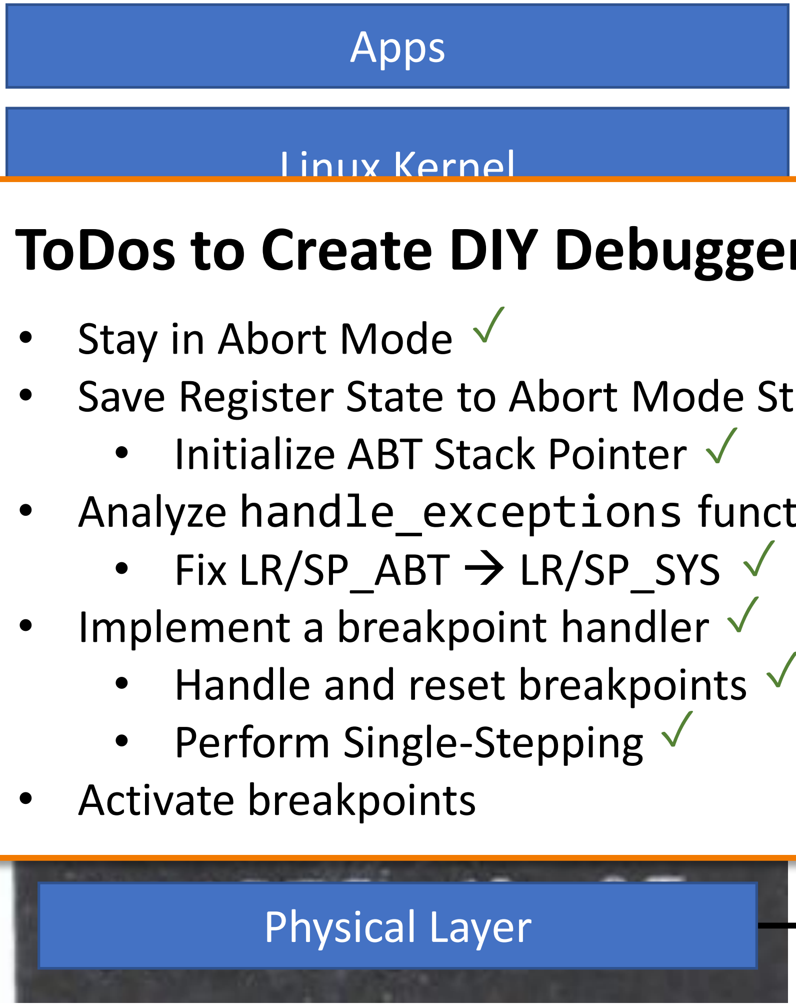
Set breakpoint at beginning of printf function

Set memory watchpoint on address of
“%s: Broadcom SDPCMD CDC driver“

```
RTE (USB-SDIO-CDC) 6.37.32.RC23.34.43 (r639704) on BCM4339 r1 @ 37.4/161.3/161.3MHz
000000.010    WP hit pc=00012b3a
000000.013    WP hit pc=00012b3a
000000.010 sdpcmdcdc0: Broadcom SDPCMD CDC driver
000000.141 reclaim section 0: Returned 31688 bytes to the heap
000000.189 nexmon_ver: 63fb-dirty-14
000000.192 wl_nd_ra_filter_init: Enter..
000000.196 TCAM: 256 used: 198 exceed:0
000000.200    WP hit pc=000126c2
000000.203 reclaim section 1: Returned 71844 bytes to the heap
000000.208    BP0 step 0: pc=000126f0 *r1=sdpcmd_dpc
000000.213    BP0 step 1: pc=000126f2
000000.216    BP0 step 2: pc=000126f4
000000.219    BP0 step 3: pc=000126f6
000000.223    BP0 step 4: pc=000126fa
000000.226    BP0 single-stepping done
000000.229 sdpcmd_dpc: Enable
000000.234 wl0: wlc_bmac_ucodembss_hwcap: Insuff mem for MBSS: templ memblks 192 fifo ...
000000.249 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
000000.295 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
000000.303 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
```

Activating Breakpoints

Why did it not trigger on other printf calls?



ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze `handle_exceptions` function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping ✓
- Activate breakpoints

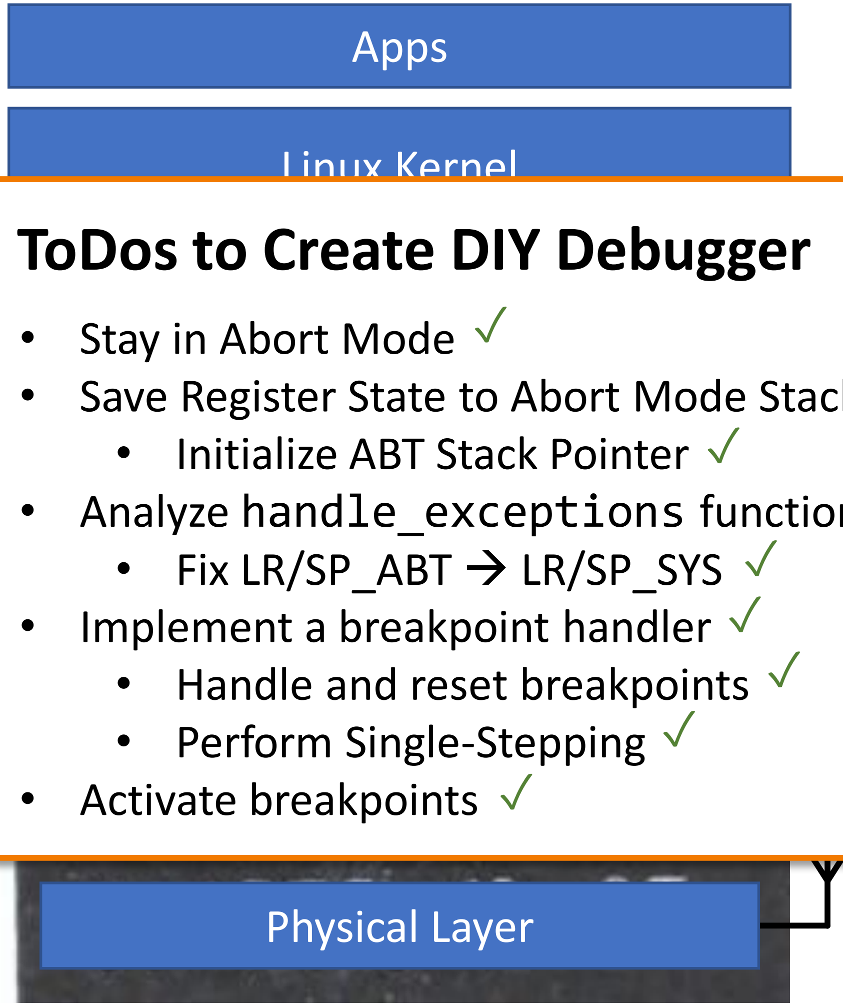
Set breakpoint at beginning of printf function

Set memory watchpoint on address of
“%s: Broadcom SDPCMD CDC driver“

```
RTE (USB-SDIO-CDC) 6.37.32.RC23.34.43 (r639704) on BCM4339 r1 @ 37.4/161.3/161.3MHz
000000.010    WP hit pc=00012b3a
000000.013    WP hit pc=00012b3a
000000.010 sdpcmdcdc0: Broadcom SDPCMD CDC driver
000000.141 reclaim section 0: Returned 31688 bytes to the heap
000000.189 nexmon_ver: 63fb-dirty-14
000000.192 wl_nd_ra_filter_init: Enter..
000000.196 TCAM: 256 used: 198 exceed:0
000000.200    WP hit pc=000126c2
000000.203 reclaim section 1: Returned 71844 bytes to the heap
000000.208    BP0 step 0: pc=000126f0 *r1=sdpcmd_dpc
000000.213    BP0 step 1: pc=000126f2
000000.216    BP0 step 2: pc=000126f4
000000.219    BP0 step 3: pc=000126f6
000000.223    BP0 step 4: pc=000126fa
000000.226    BP0 single-stepping done
000000.229 sdpcmd_dpc: Enable
000000.234 wl0: wlc_bmac_ucodembss_hwcap: Insuff mem for MBSS: templ memblks 192 fifo ...
000000.249 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
000000.295 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
000000.303 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
```

Activating Breakpoints

Why did it not trigger on other printf calls?



ToDos to Create DIY Debugger

- Stay in Abort Mode ✓
- Save Register State to Abort Mode Stack ✓
  - Initialize ABT Stack Pointer ✓
- Analyze handle_exceptions function ✓
  - Fix LR/SP_ABT → LR/SP_SYS ✓
- Implement a breakpoint handler ✓
  - Handle and reset breakpoints ✓
  - Perform Single-Stepping ✓
- Activate breakpoints ✓

Set breakpoint at beginning of printf function

Set memory watchpoint on address of “%s: Broadcom SDPCMD CDC driver“

```
RTE (USB-SDIO-CDC) 6.37.32.RC23.34.43 (r639704) on BCM4339 r1 @ 37.4/161.3/161.3MHz
000000.010    WP hit pc=00012b3a
000000.013    WP hit pc=00012b3a
000000.010 sdpcmdcdc0: Broadcom SDPCMD CDC driver
000000.141 reclaim section 0: Returned 31688 bytes to the heap
000000.189 nexmon_ver: 63fb-dirty-14
000000.192 wl_nd_ra_filter_init: Enter..
000000.196 TCAM: 256 used: 198 exceed:0
000000.200    WP hit pc=000126c2
000000.203 reclaim section 1: Returned 71844 bytes to the heap
000000.208    BP0 step 0: pc=000126f0 *r1=sdpcmd_dpc
000000.213    BP0 step 1: pc=000126f2
000000.216    BP0 step 2: pc=000126f4
000000.219    BP0 step 3: pc=000126f6
000000.223    BP0 step 4: pc=000126fa
000000.226    BP0 single-stepping done
000000.229 sdpcmd_dpc: Enable
000000.234 wl0: wlc_bmac_ucodembss_hwcap: Insuff mem for MBSS: templ memblks 192 fifo ...
000000.249 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
000000.295 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
000000.303 wl0: wlc_enable_probe_req: state down, deferring setting of host flags
```

Run the Debugger on Your Own!

nexmon.org

Run the Debugger on Your Own!

nexmon.org

Clone repository

- buildtools (e.g., compiler)
- firmwares (e.g., for BCM4339)
- patches
  - <chip>
  - bcm4339
    - <firmware version>
    - 6_37_34_43
      - <patch name>
      - nexmon (monitormode + frame injection)
      - …
- Makefile
- setup_env.sh
- …

Run the Debugger on Your Own!

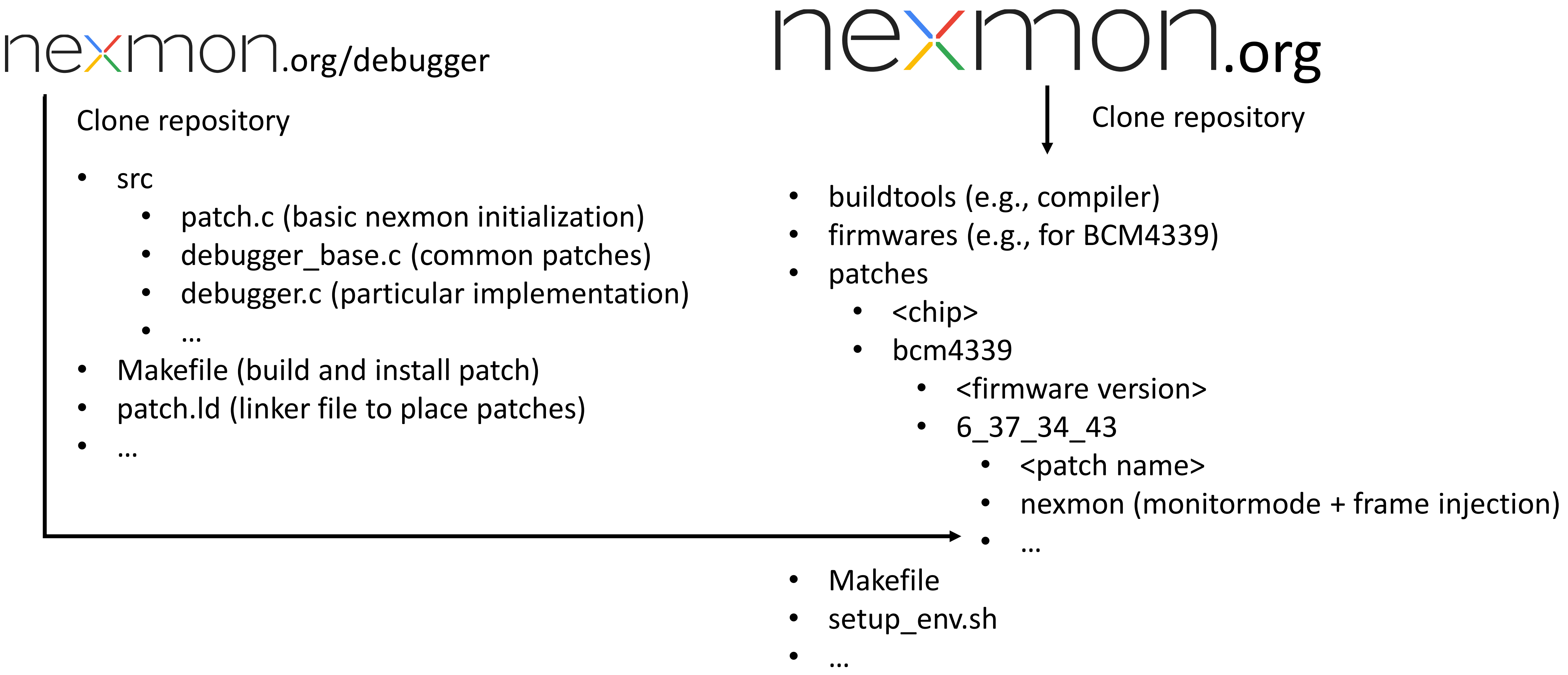
nexmon.org/debugger

nexmon.org

Clone repository

- buildtools (e.g., compiler)
- firmwares (e.g., for BCM4339)
- patches
  - <chip>
  - bcm4339
    - <firmware version>
    - 6_37_34_43
      - <patch name>
      - nexmon (monitormode + frame injection)
      - …
- Makefile
- setup_env.sh
- …

Run the Debugger on Your Own!



nexmon.org/debugger

Clone repository

- src
  - patch.c (basic nexmon initialization)
  - debugger_base.c (common patches)
  - debugger.c (particular implementation)
  - …
- Makefile (build and install patch)
- patch.ld (linker file to place patches)
- …

nexmon.org

Clone repository

- buildtools (e.g., compiler)
- firmwares (e.g., for BCM4339)
- patches
  - <chip>
  - bcm4339
    - <firmware version>
    - 6_37_34_43
      - <patch name>
      - nexmon (monitormode + frame injection)
      - …
- Makefile
- setup_env.sh
- …

Run the Debugger on Your Own!

nexmon.org/debugger

Clone repository

- src
    - patch.c (basic nexmon initialization)
    - debugger_base.c (common patches)
    - debugger.c (particular implementation)
    - …
- Makefile (build and install patch)
- patch.ld (linker file to place patches)
- …

```
nexmon> make && source setup_env.sh
nexmon> cd patches/bcm4339/ 6_37_34_43/debugger
nexmon> make install-firmware
```

nexmon.org

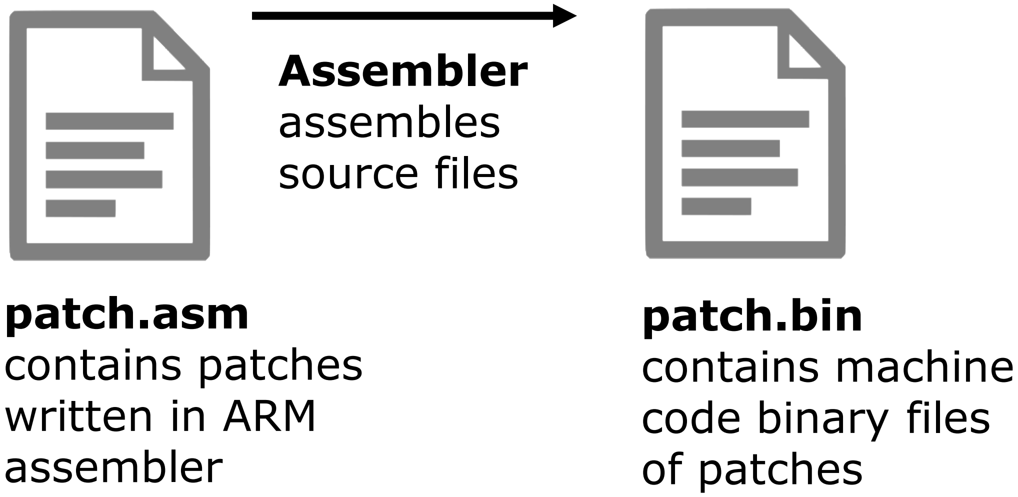
Clone repository

- buildtools (e.g., compiler)
- firmwares (e.g., for BCM4339)
- patches
    - <chip>
    - bcm4339
        - <firmware version>
        - 6_37_34_43
            - <patch name>
            - nexmon (monitormode + frame injection)
            - …
- Makefile
- setup_env.sh
- …

Firmware Patching

patch.asm
contains patches
written in ARM
assembler

Firmware Patching

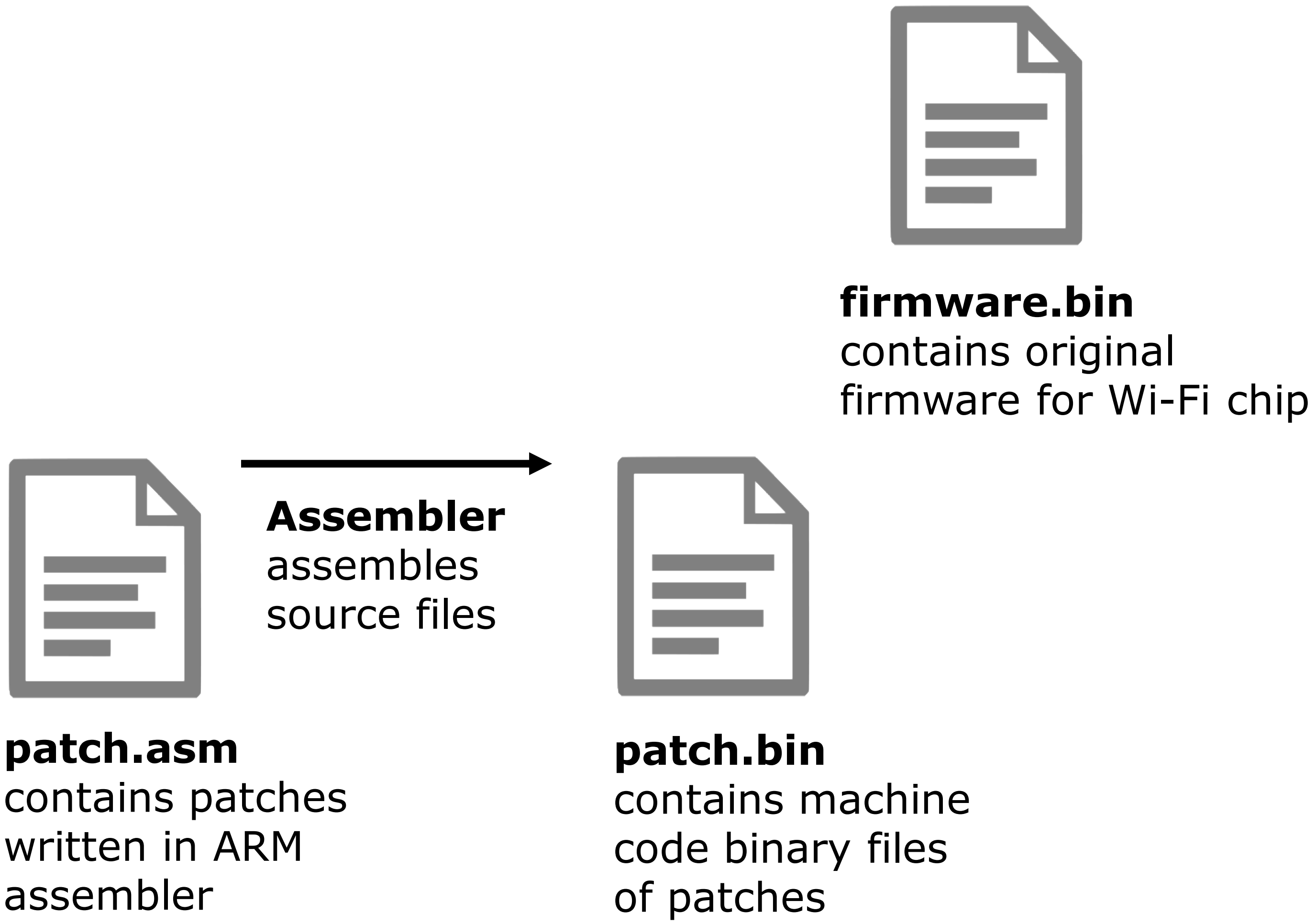


Assembler
assembles
source files

patch.asm
contains patches
written in ARM
assembler

patch.bin
contains machine
code binary files
of patches

Firmware Patching



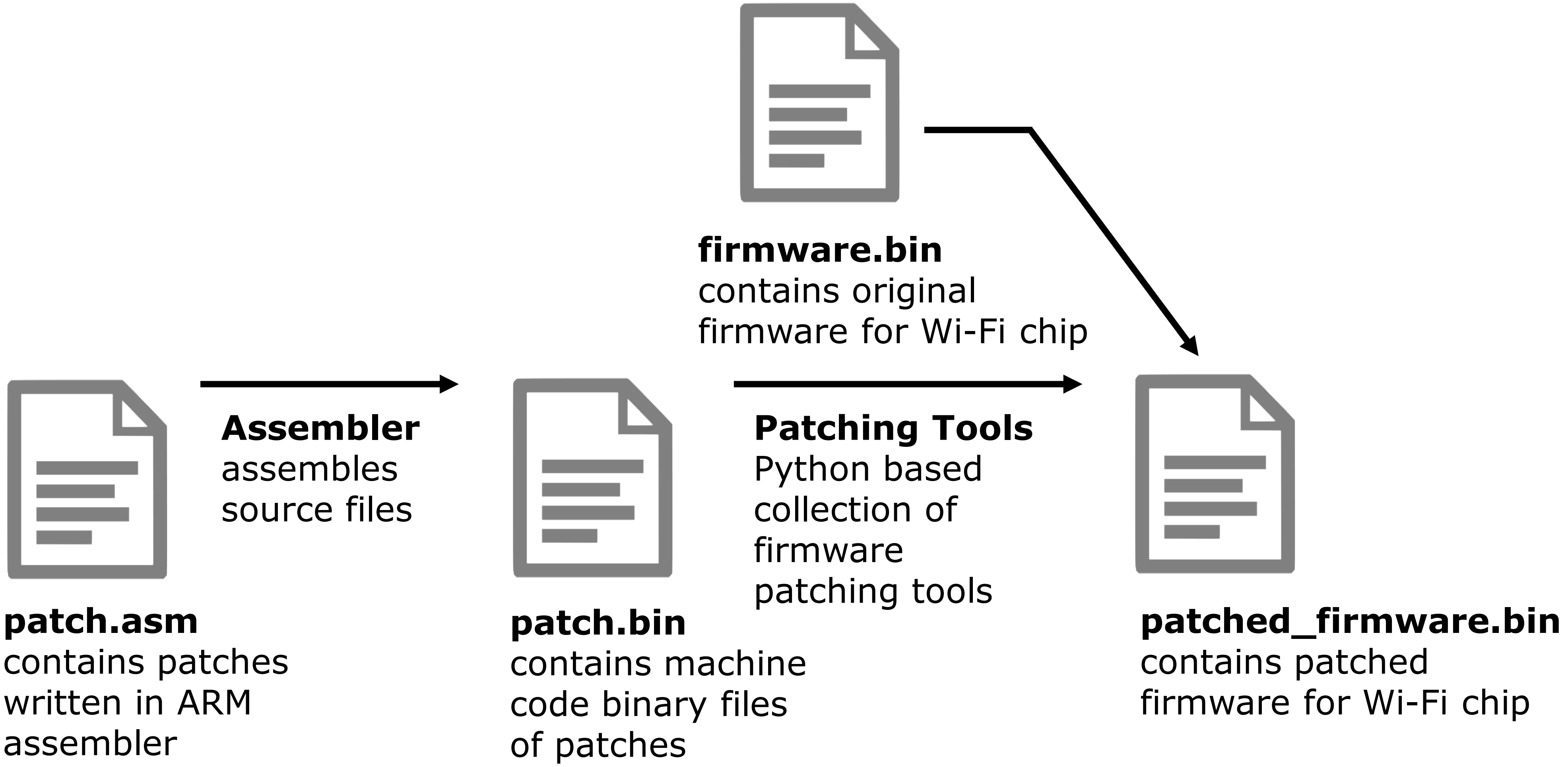
firmware.bin
contains original
firmware for Wi-Fi chip

Assembler
assembles
source files

patch.asm
contains patches
written in ARM
assembler

patch.bin
contains machine
code binary files
of patches

Firmware Patching



firmware.bin
contains original
firmware for Wi-Fi chip

patch.asm
contains patches
written in ARM
assembler

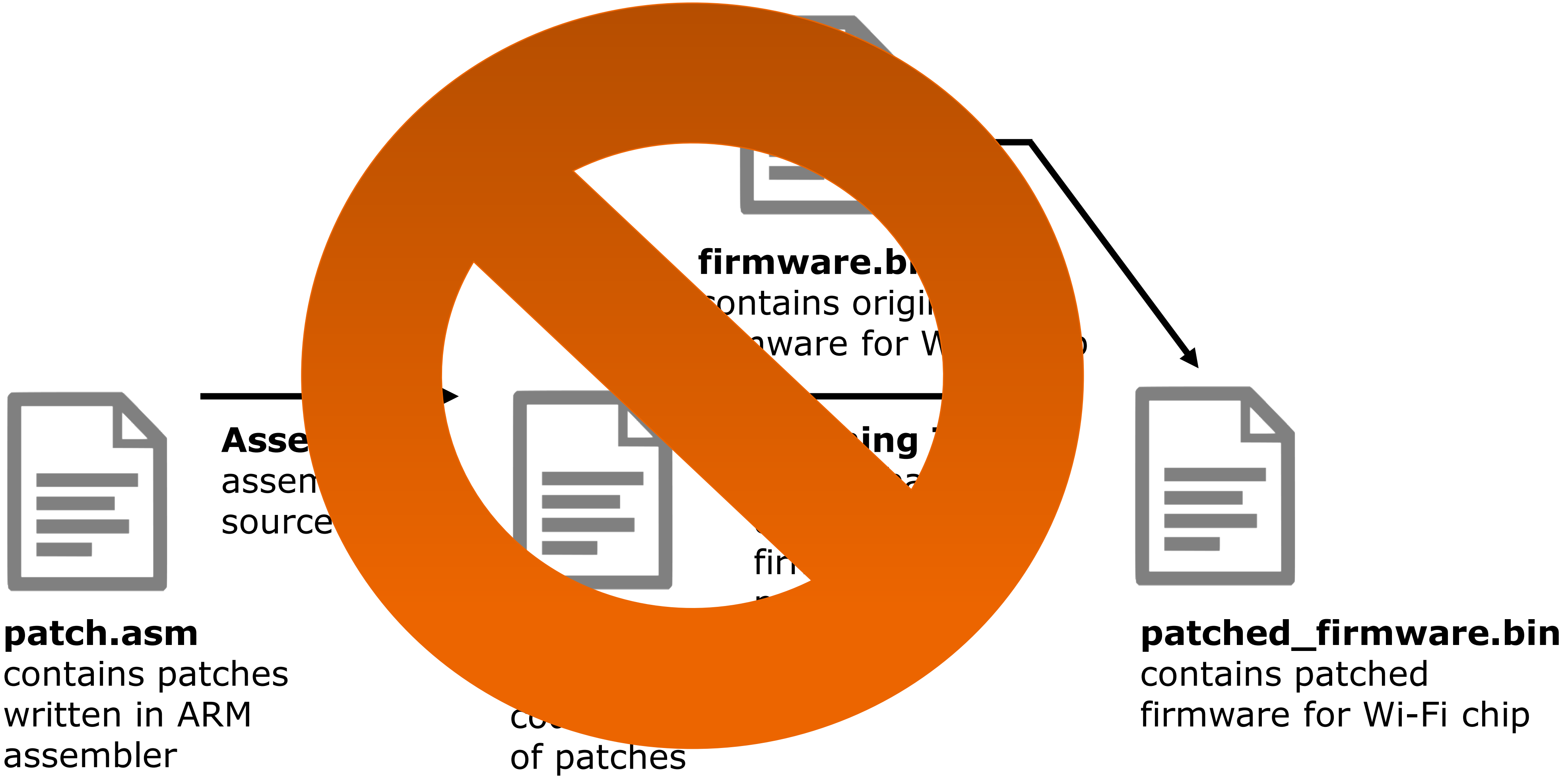
Assembler
assembles
source files

patch.bin
contains machine
code binary files
of patches

Patching Tools
Python based
collection of
firmware
patching tools

patched_firmware.bin
contains patched
firmware for Wi-Fi chip

Firmware Patching



patch.asm
contains patches
written in ARM
assembler

Asse
assen
source

co
of patches

firmware.bi
ontains origi
nware for W

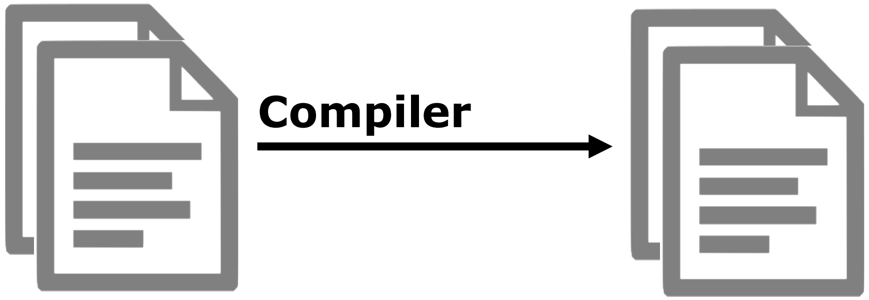
ing
fir

patched_firmware.bin
contains patched
firmware for Wi-Fi chip

C Based Firmware Patching

C files
contain
source code
for patches

C Based Firmware Patching

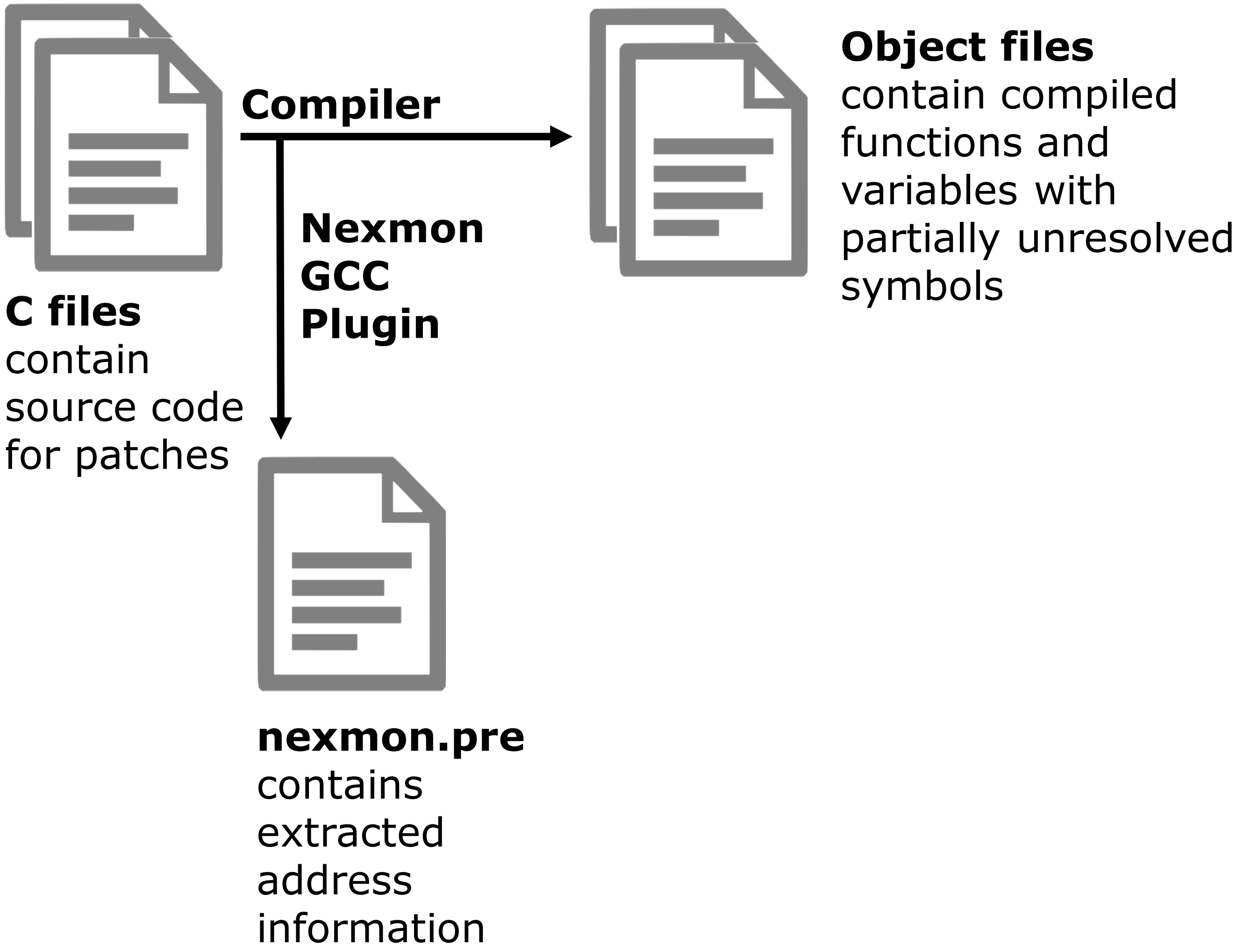


Compiler

C files
contain
source code
for patches

Object files
contain compiled
functions and
variables with
partially unresolved
symbols

C Based Firmware Patching



C files
contain
source code
for patches

Compiler

Nexmon
GCC
Plugin

Object files
contain compiled
functions and
variables with
partially unresolved
symbols

nexmon.pre
contains
extracted
address
information

C Based Firmware Patching

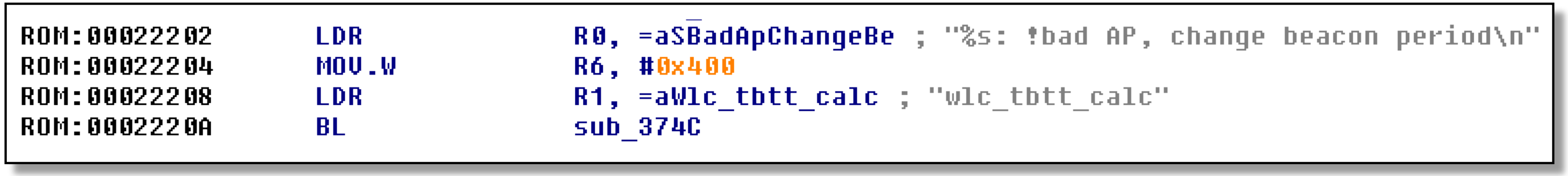
C files
contain source code for patches

Compiler

Nexmon GCC Plugin

Object files
contain compiled functions and variables with partially unresolved symbols

nexmon.pre
contains extracted address information



```
ROM:00022202        LDR              R0, =aSBadApChangeBe ; "%s: !bad AP, change beacon period\n"
ROM:00022204        MOV.W            R6, #0x400
ROM:00022208        LDR              R1, =aWlc_tbtt_calc ; "wlc_tbtt_calc"
ROM:0002220A        BL               sub_374C
```

C Based Firmware Patching

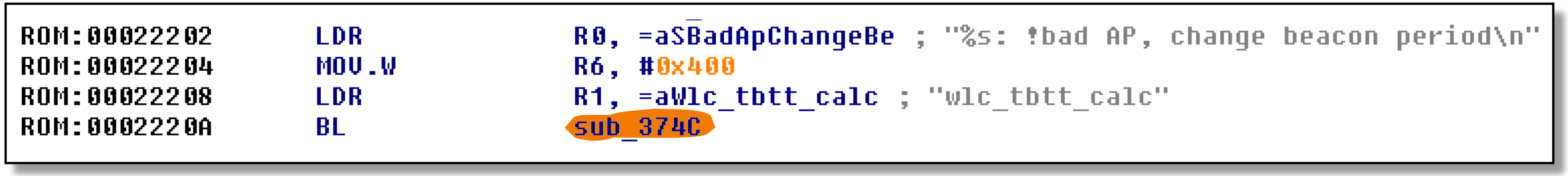
C files
contain source code for patches

Compiler

Object files
contain compiled functions and variables with partially unresolved symbols

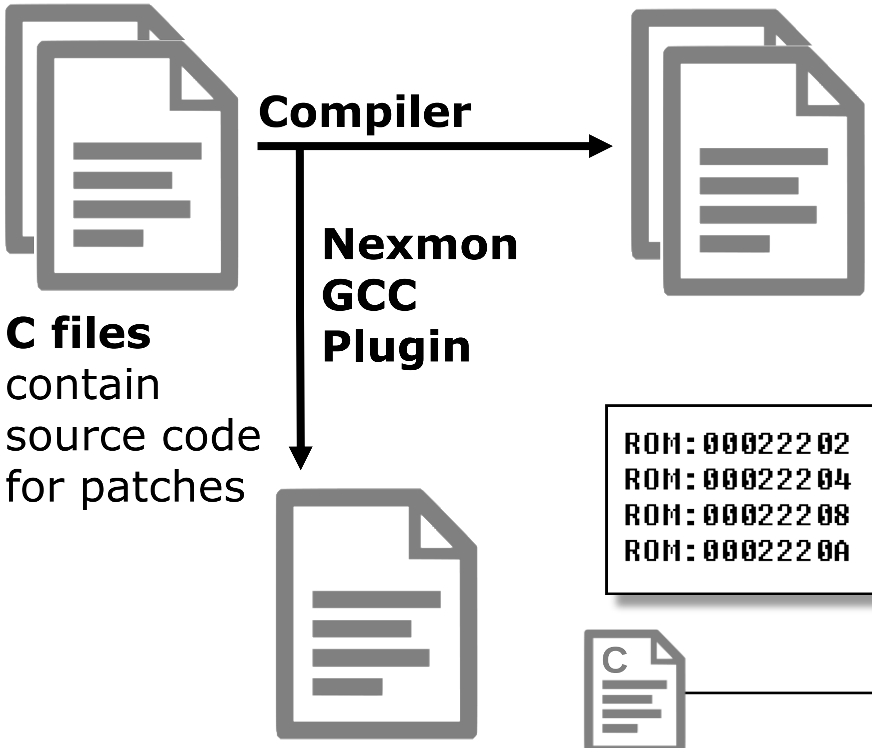
Nexmon GCC Plugin

nexmon.pre
contains extracted address information



```
ROM:00022202        LDR         R0, =aSBadApChangeBe ; "%s: !bad AP, change beacon period\n"
ROM:00022204        MOV.W       R6, #0x400
ROM:00022208        LDR         R1, =aWlc_tbtt_calc ; "wlc_tbtt_calc"
ROM:0002220A        BL          sub_374C
```

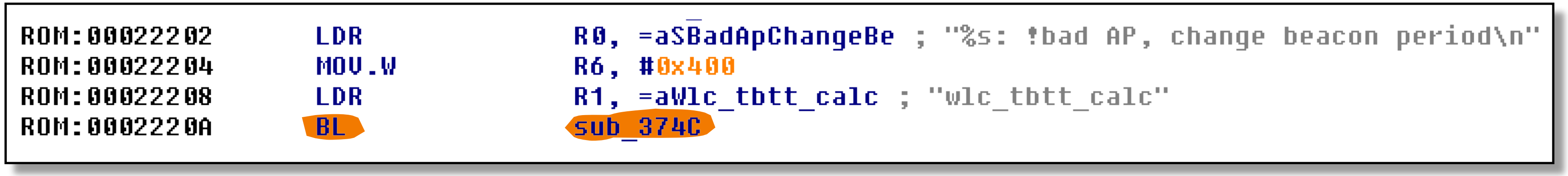
C Based Firmware Patching



C files contain source code for patches

Object files contain compiled functions and variables with partially unresolved symbols

nexmon.pre contains extracted address information



```
ROM:00022202        LDR          R0, =aSBadApChangeBe ; "%s: !bad AP, change beacon period\n"
ROM:00022204        MOV.W        R6, #0x400
ROM:00022208        LDR          R1, =aWlc_tbtt_calc ; "wlc_tbtt_calc"
ROM:0002220A        BL           sub_374C
```

C Based Firmware Patching

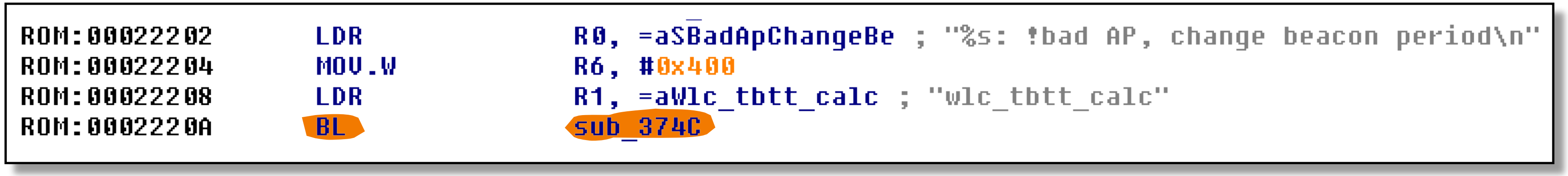
C files contain source code for patches

Compiler

Nexmon GCC Plugin

Object files contain compiled functions and variables with partially unresolved symbols

nexmon.pre contains extracted address information



```
ROM:00022202        LDR             R0, =aSBadApChangeBe ; "%s: !bad AP, change beacon period\n"
ROM:00022204        MOV.W           R6, #0x400
ROM:00022208        LDR             R1, =aWlc_tbtt_calc ; "wlc_tbtt_calc"
ROM:0002220A        BL              sub_374C
```

```
__attribute__((at(0x2220A, ”flashpatch”, CHIP_..., FW_...)))
BLPatch(hook_name, target_function_name);
```

C Based Firmware Patching

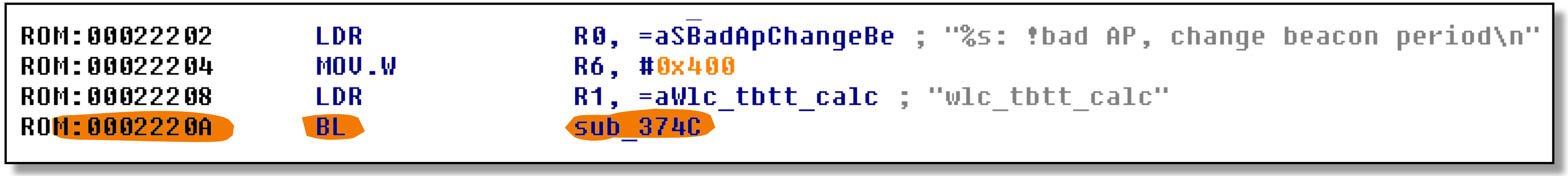
Compiler

Nexmon GCC Plugin

C files contain source code for patches

Object files contain compiled functions and variables with partially unresolved symbols

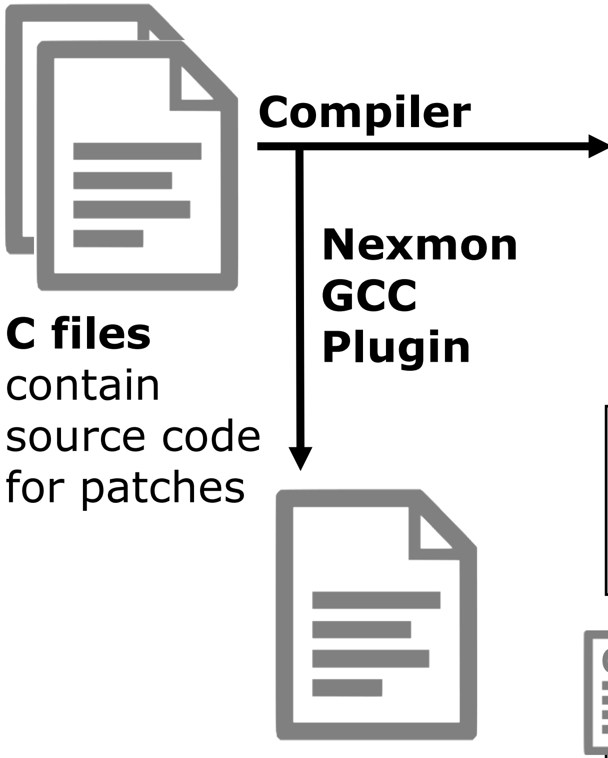
nexmon.pre contains extracted address information



```
ROM:00022202        LDR         R0, =aSBadApChangeBe ; "%s: !bad AP, change beacon period\n"
ROM:00022204        MOV.W       R6, #0x400
ROM:00022208        LDR         R1, =aWlc_tbtt_calc ; "wlc_tbtt_calc"
ROM:0002220A        BL          sub_374C
```

```
__attribute__((at(0x2220A, ”flashpatch”, CHIP_..., FW_...)))
BLPatch(hook_name, target_function_name);
```

C Based Firmware Patching



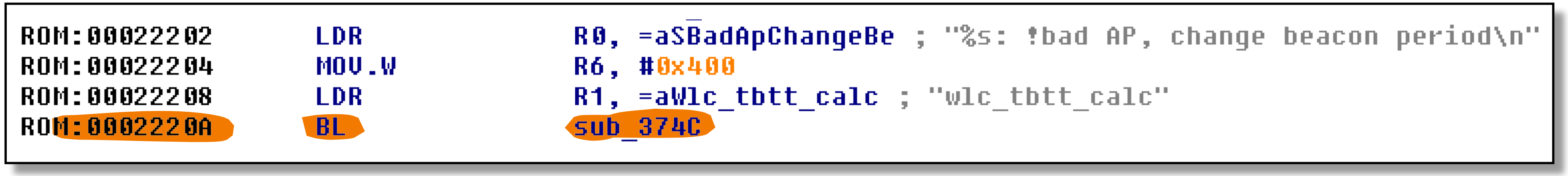
C files
contain source code for patches

Compiler

Nexmon GCC Plugin

nexmon.pre
contains extracted address information

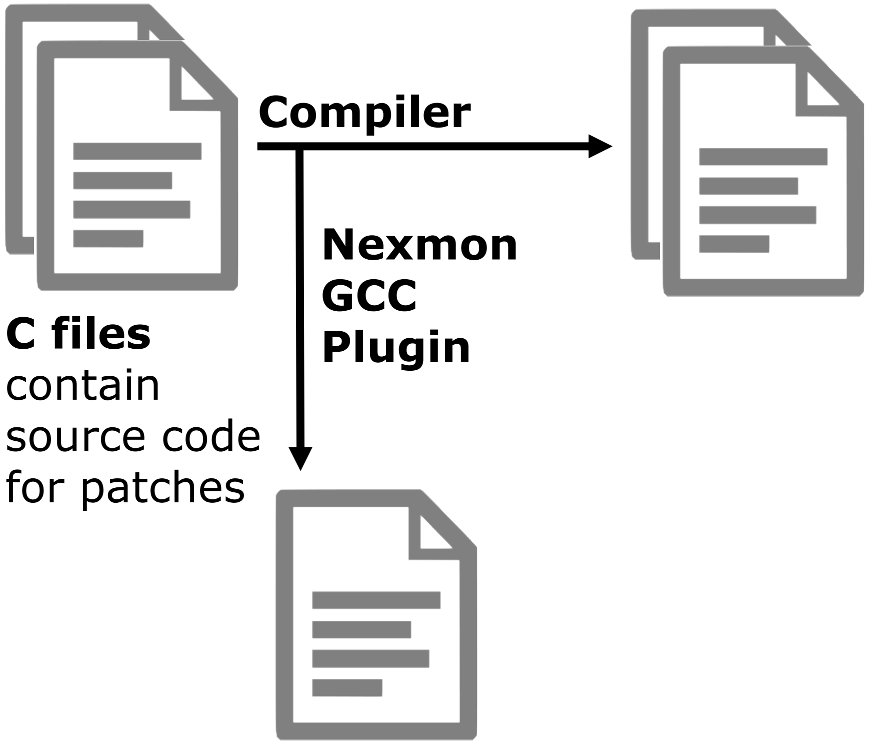
Object files
contain compiled functions and variables with partially unresolved symbols



```
ROM:00022202      LDR          R0, =aSBadApChangeBe ; "%s: !bad AP, change beacon period\n"
ROM:00022204      MOV.W        R6, #0x400
ROM:00022208      LDR          R1, =aWlc_tbtt_calc ; "wlc_tbtt_calc"
ROM:0002220A      BL           sub_374C
```

```
__attribute__((at(0x2220A, "flashpatch", CHIP_..., FW_...)))
BLPatch(hook_name, target_function_name);
```

C Based Firmware Patching



C files
contain
source code
for patches

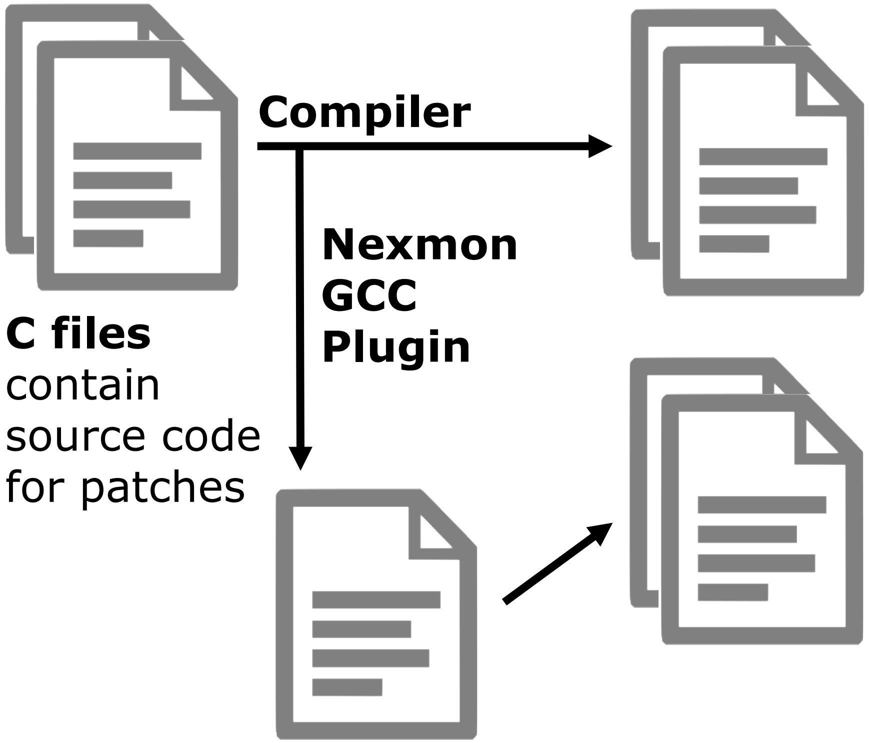
Compiler

Object files
contain compiled
functions and
variables with
partially unresolved
symbols

**Nexmon
GCC
Plugin**

nexmon.pre
contains
extracted
address
information

C Based Firmware Patching



C files
contain source code for patches

Compiler

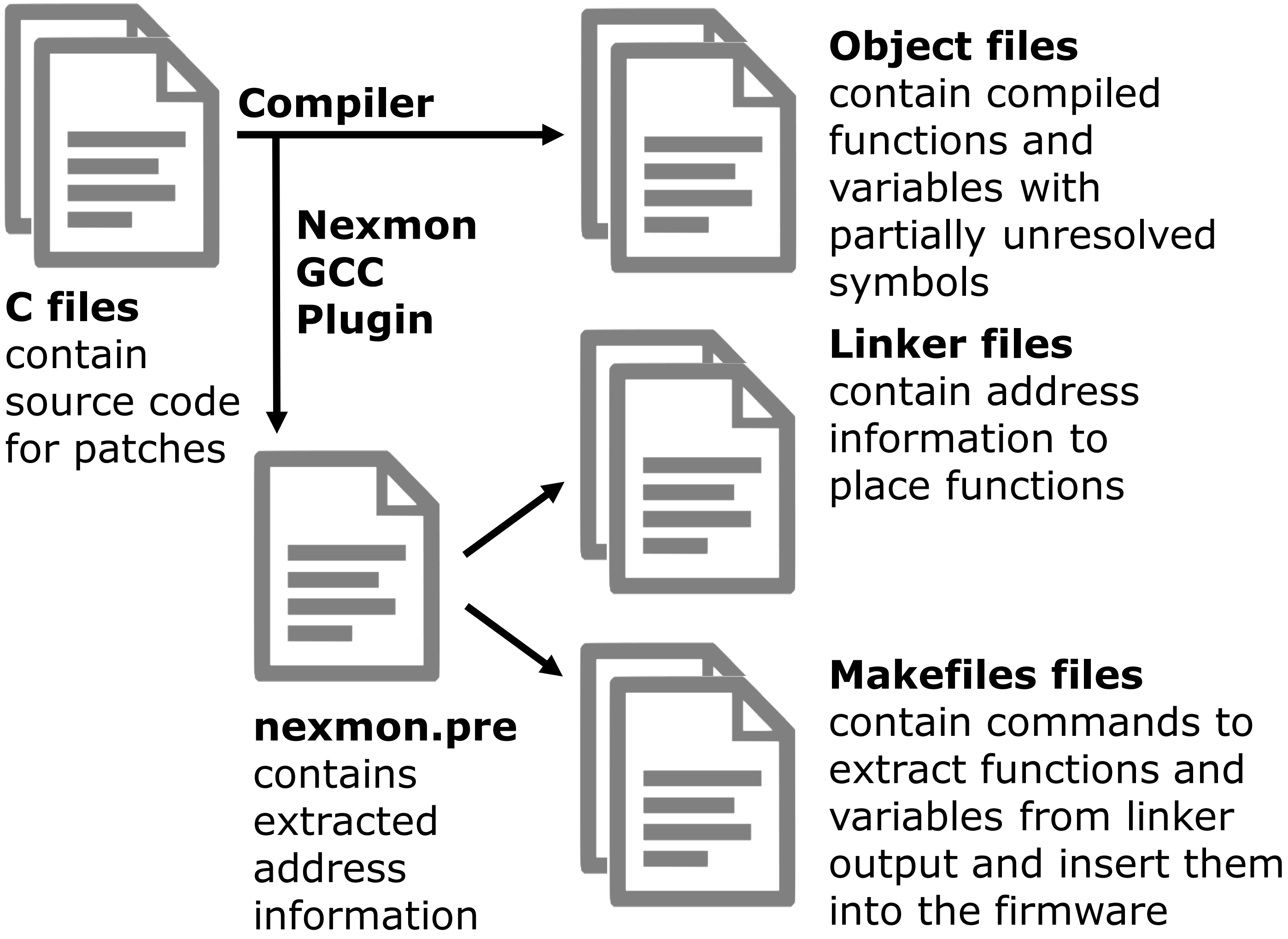
Nexmon GCC Plugin

Object files
contain compiled functions and variables with partially unresolved symbols

nexmon.pre
contains extracted address information

Linker files
contain address information to place functions

C Based Firmware Patching



C files
contain source code for patches

Compiler

Nexmon GCC Plugin

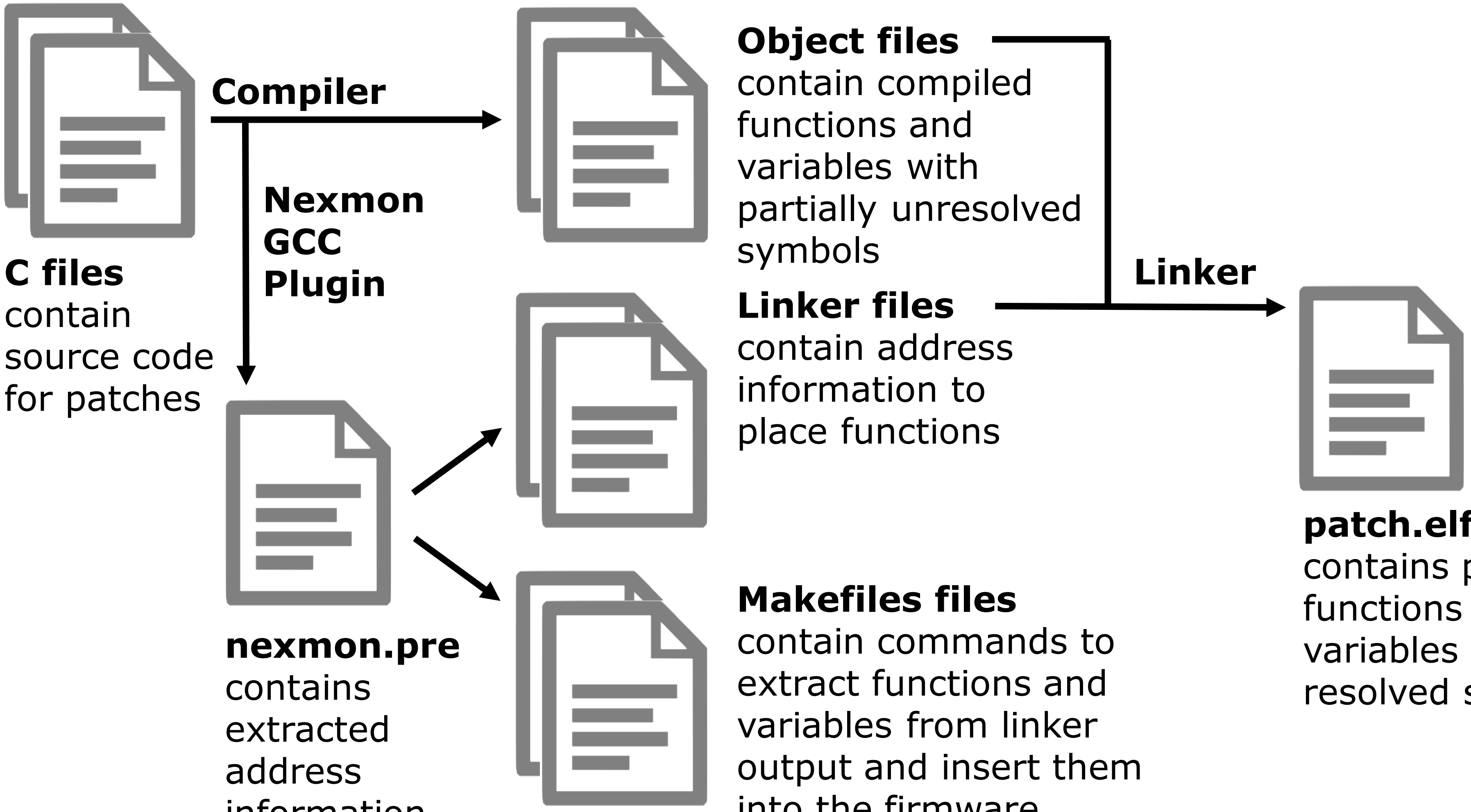
nexmon.pre
contains extracted address information

Object files
contain compiled functions and variables with partially unresolved symbols

Linker files
contain address information to place functions

Makefiles files
contain commands to extract functions and variables from linker output and insert them into the firmware

C Based Firmware Patching



C files
contain source code for patches

Compiler

Nexmon GCC Plugin

nexmon.pre
contains extracted address information

Object files
contain compiled functions and variables with partially unresolved symbols

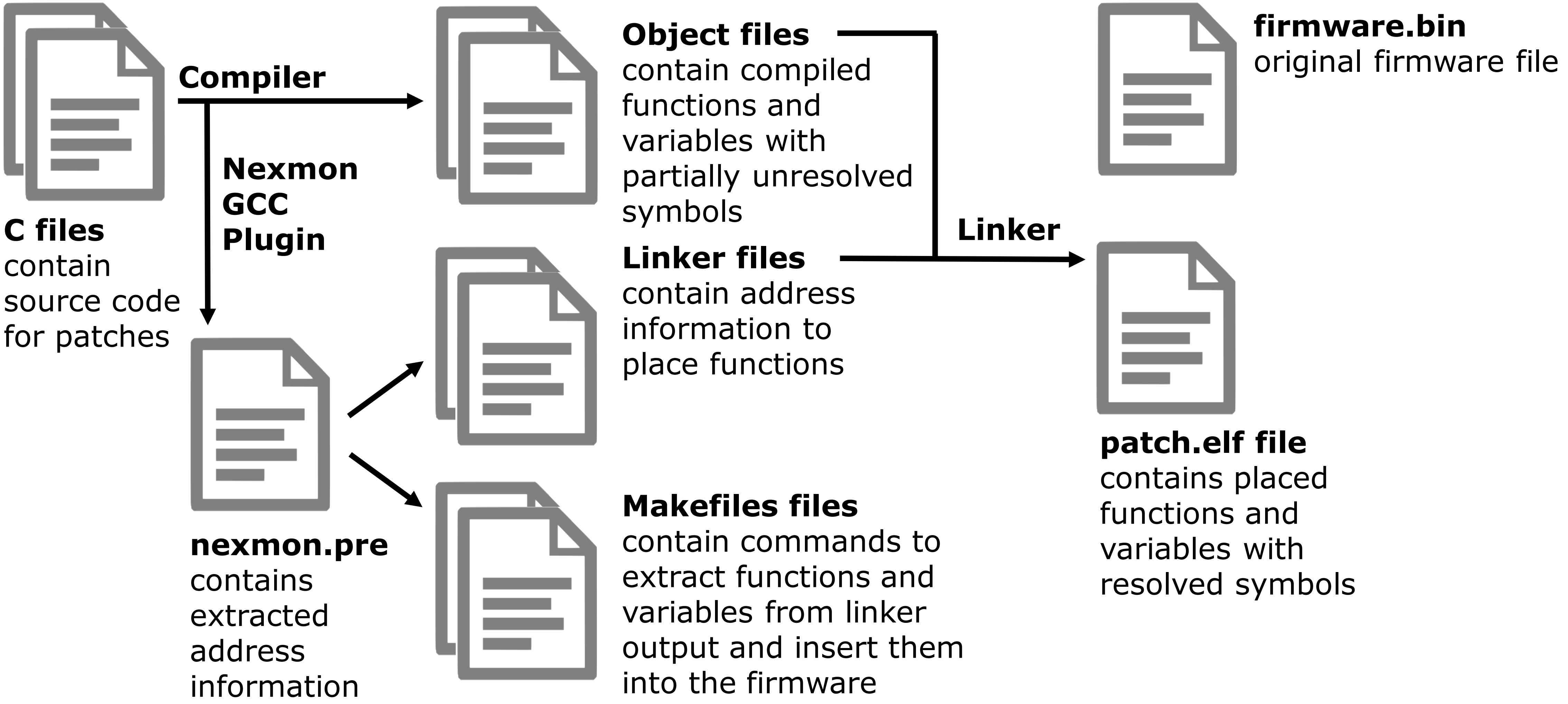
Linker files
contain address information to place functions

Makefiles files
contain commands to extract functions and variables from linker output and insert them into the firmware

Linker

patch.elf file
contains placed functions and variables with resolved symbols

C Based Firmware Patching



C files
contain source code for patches

Compiler

Nexmon GCC Plugin

nexmon.pre
contains extracted address information

Object files
contain compiled functions and variables with partially unresolved symbols

Linker files
contain address information to place functions

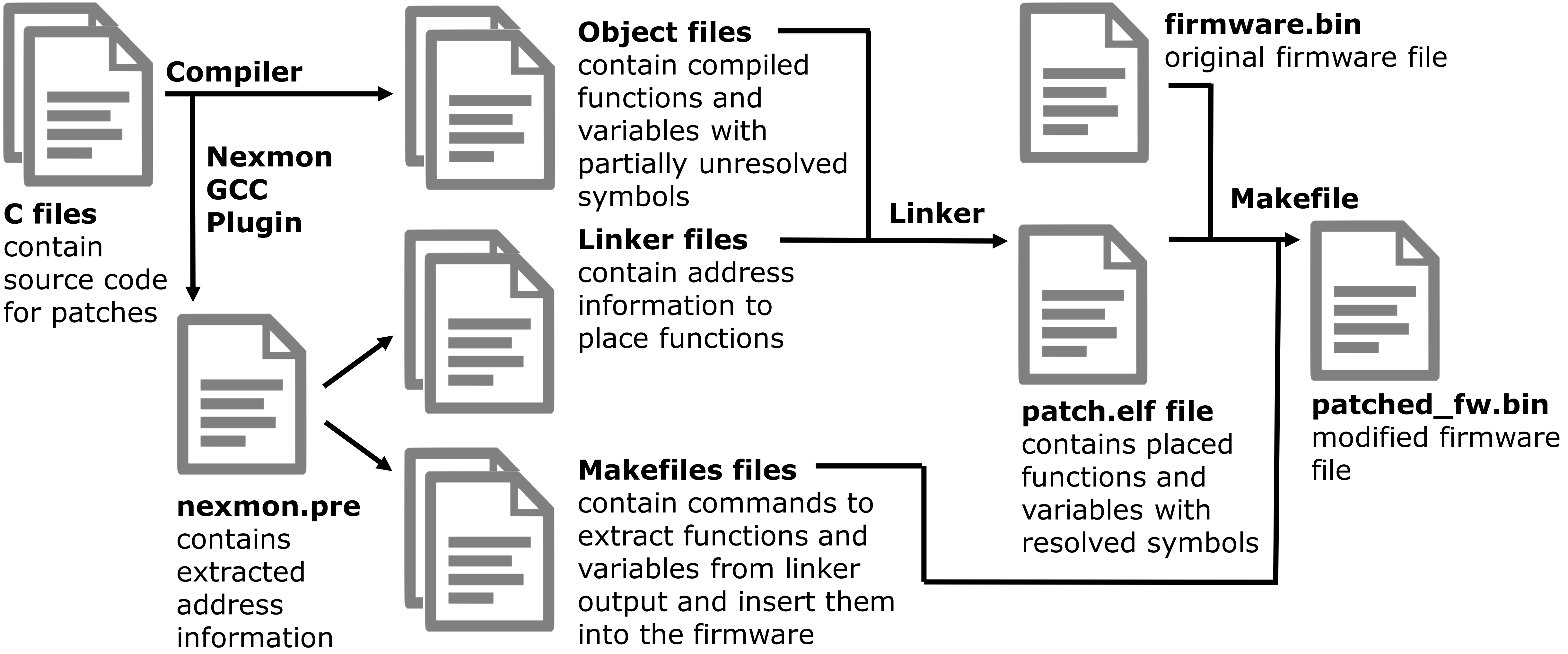
Makefiles files
contain commands to extract functions and variables from linker output and insert them into the firmware

Linker

firmware.bin
original firmware file

patch.elf file
contains placed functions and variables with resolved symbols

C Based Firmware Patching



C files
contain source code for patches

Compiler

Nexmon GCC Plugin

nexmon.pre
contains extracted address information

Object files
contain compiled functions and variables with partially unresolved symbols

Linker files
contain address information to place functions

Makefiles files
contain commands to extract functions and variables from linker output and insert them into the firmware

Linker

firmware.bin
original firmware file

patch.elf file
contains placed functions and variables with resolved symbols

Makefile

patched_fw.bin
modified firmware file

Patch Placement

```c
#pragma NEXMON targetregion "patch"

void target_function_name(int a) {
    so_something(a);
}

__attribute__((at(0x2220A, ”flashpatch", CHIP_..., FW_...)))
BLPatch(hook_name, target_function_name);
```

Patch Placement

```
#pragma NEXMON targetregion "patch"

void target_function_name(int a) {
    so_something(a);
}

__attribute__((at(0x2220A, ”flashpatch", CHIP_..., FW_...)))
BLPatch(hook_name, target_function_name);
```

RAM of a BCM4339 in Nexus 5:

Patch Placement

```
#pragma NEXMON targetregion "patch"

void target_function_name(int a) {
    so_something(a);
}

__attribute__((at(0x2220A, ”flashpatch", CHIP_..., FW_...)))
BLPatch(hook_name, target_function_name);
```

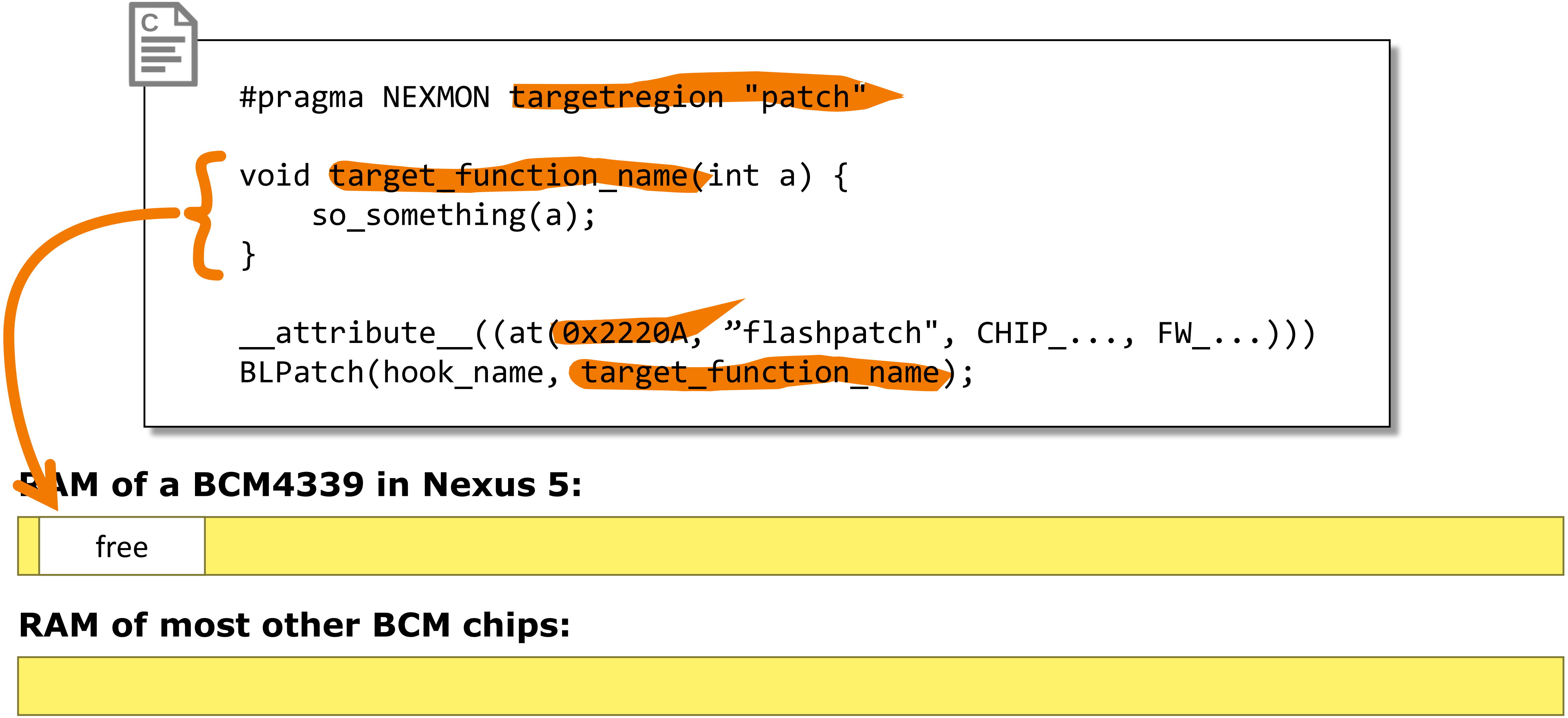
RAM of a BCM4339 in Nexus 5:

Patch Placement

```
#pragma NEXMON targetregion "patch"

void target_function_name(int a) {
    so_something(a);
}

__attribute__((at(0x2220A, ”flashpatch", CHIP_..., FW_...)))
BLPatch(hook_name, target_function_name);
```

RAM of a BCM4339 in Nexus 5:

Patch Placement



```
#pragma NEXMON targetregion "patch"

void target_function_name(int a) {
    so_something(a);
}

__attribute__((at(0x2220A, ”flashpatch", CHIP_..., FW_...)))
BLPatch(hook_name, target_function_name);
```

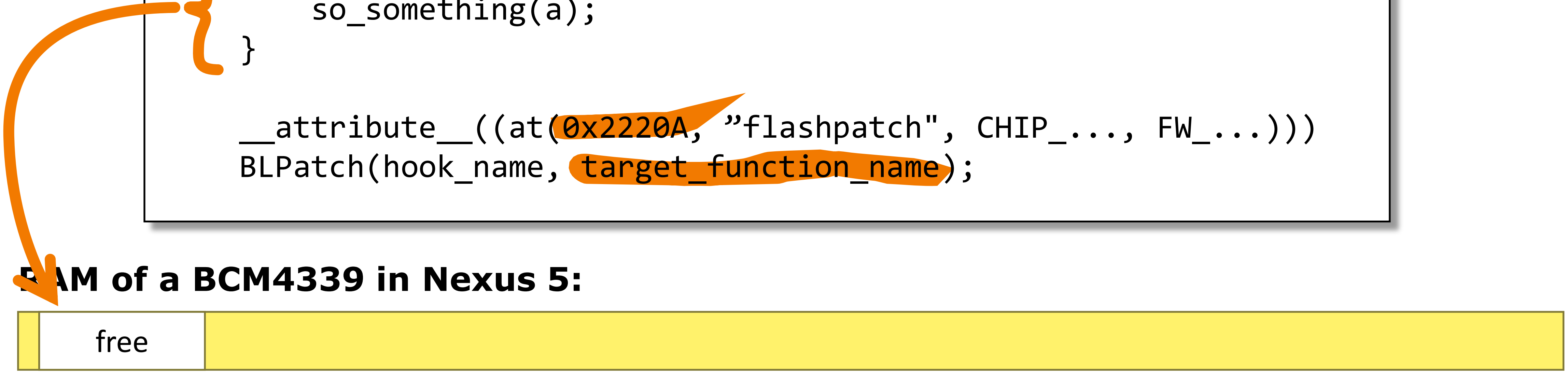
RAM of a BCM4339 in Nexus 5:

RAM of most other BCM chips:

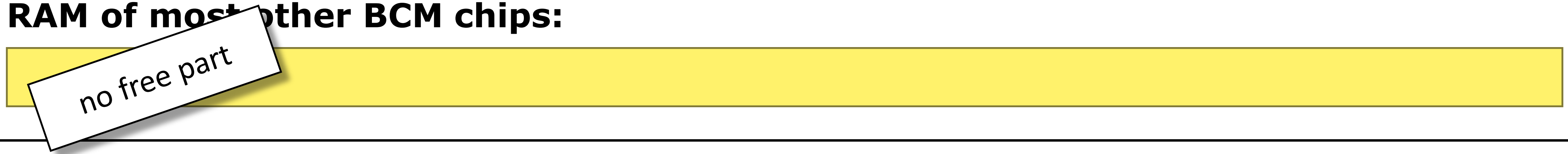
Patch Placement

```
#pragma NEXMON targetregion "patch"

void target_function_name(int a) {
    so_something(a);
}

__attribute__((at(0x2220A, ”flashpatch", CHIP_..., FW_...)))
BLPatch(hook_name, target_function_name);
```

RAM of a BCM4339 in Nexus 5:



RAM of most other BCM chips:

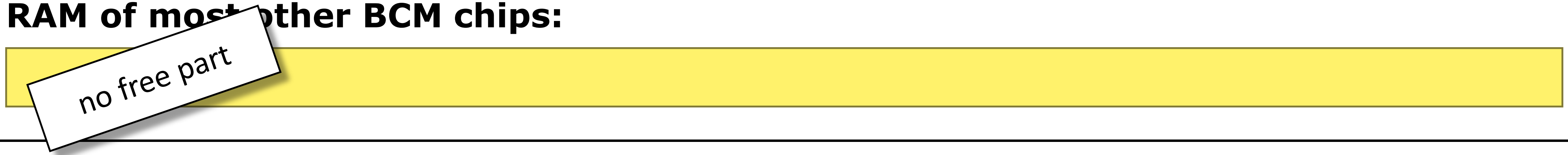


Patch Placement

```
#pragma NEXMON targetregion "patch"

void target_function_name(int a) {
    so_something(a);
}

__attribute__((at(0x2220A, ”flashpatch", CHIP_..., FW_...)))
BLPatch(hook_name, target_function_name);
```

RAM of a BCM4339 in Nexus 5:

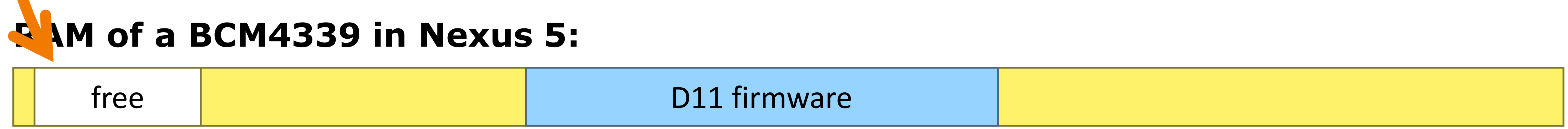
RAM of most other BCM chips:



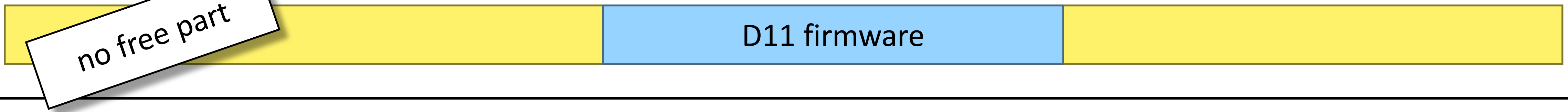
Patch Placement

```
#pragma NEXMON targetregion "patch"

void target_function_name(int a) {
    so_something(a);
}

__attribute__((at(0x2220A, ”flashpatch", CHIP_..., FW_...)))
BLPatch(hook_name, target_function_name);
```

RAM of a BCM4339 in Nexus 5:

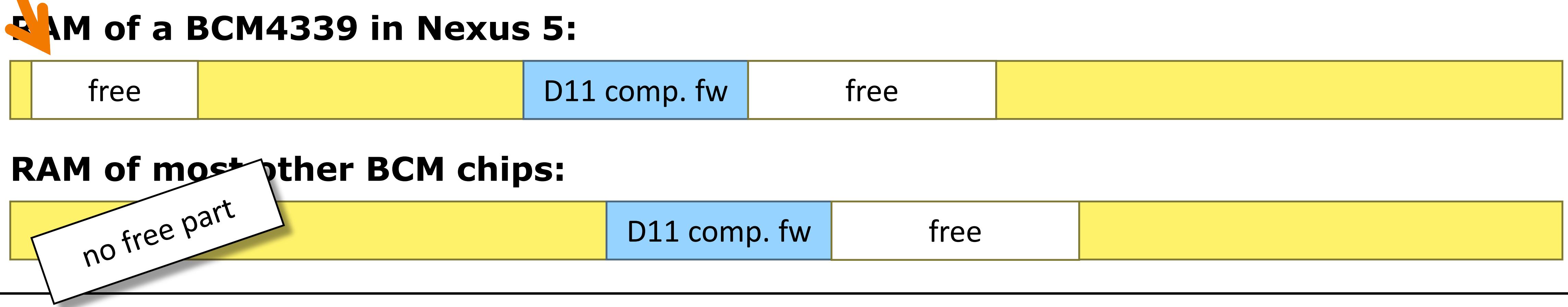


RAM of most other BCM chips:



Patch Placement

```
#pragma NEXMON targetregion "patch"

void target_function_name(int a) {
    so_something(a);
}

__attribute__((at(0x2220A, ”flashpatch", CHIP_..., FW_...)))
BLPatch(hook_name, target_function_name);
```

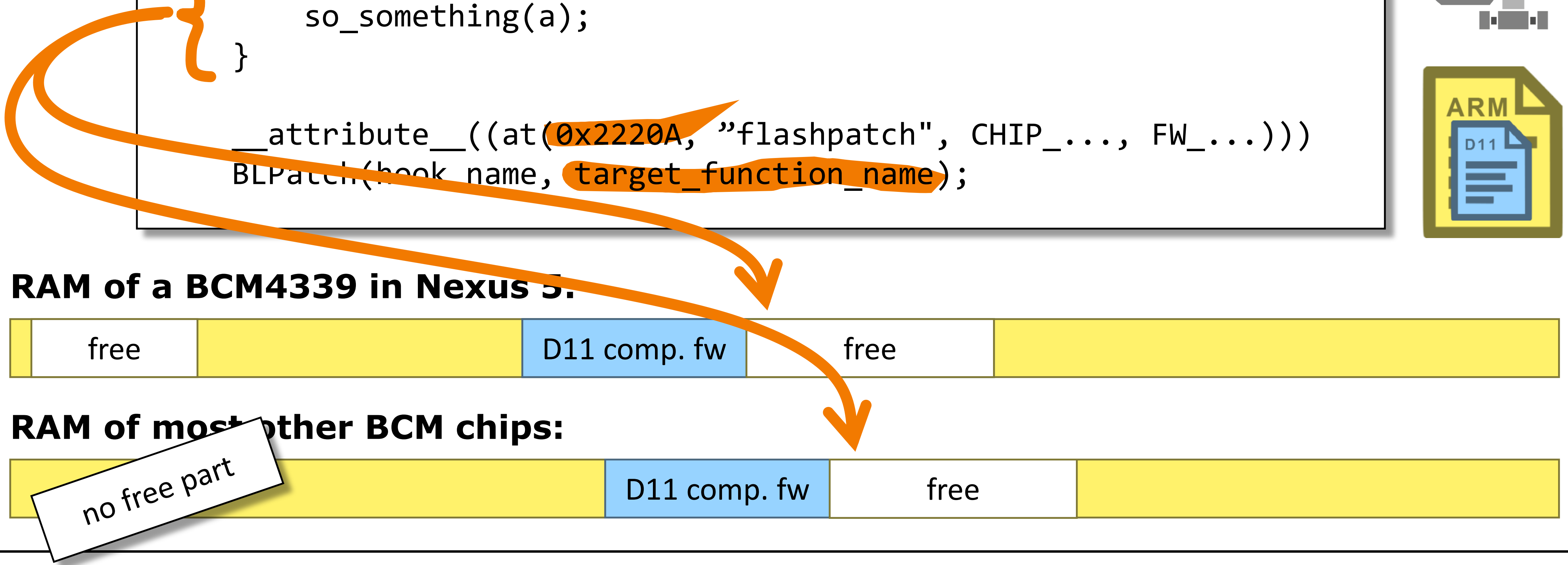


RAM of a BCM4339 in Nexus 5:

RAM of most other BCM chips:

Patch Placement

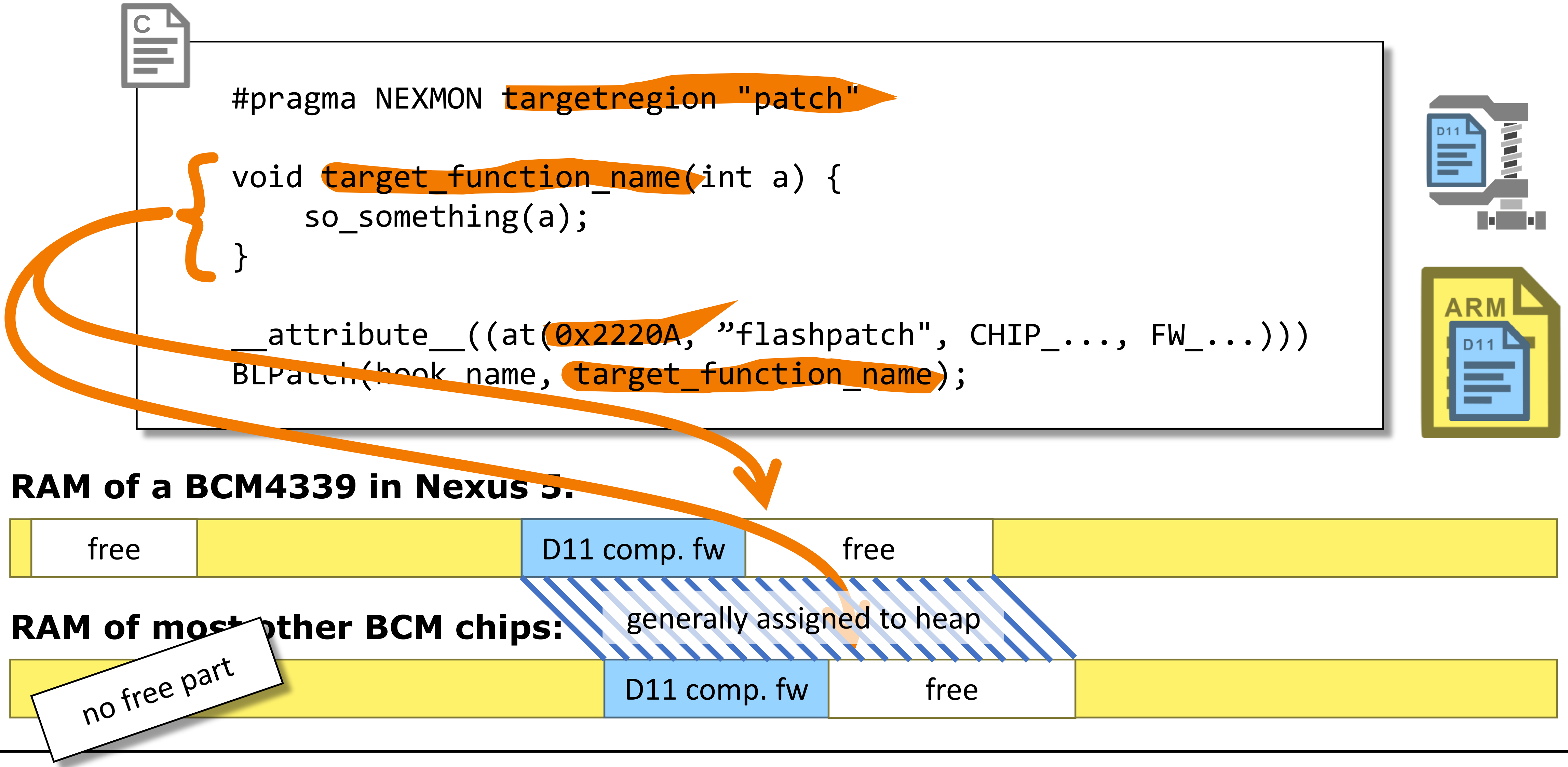
```
#pragma NEXMON targetregion "patch"

void target_function_name(int a) {
    so_something(a);
}

__attribute__((at(0x2220A, ”flashpatch", CHIP_..., FW_...)))
BLPatch(hook_name, target_function_name);
```



RAM of a BCM4339 in Nexus 5:

RAM of most other BCM chips:

Patch Placement



```
#pragma NEXMON targetregion "patch"

void target_function_name(int a) {
    so_something(a);
}

__attribute__((at(0x2220A, ”flashpatch", CHIP_..., FW_...)))
BLPatch(hook_name, target_function_name);
```

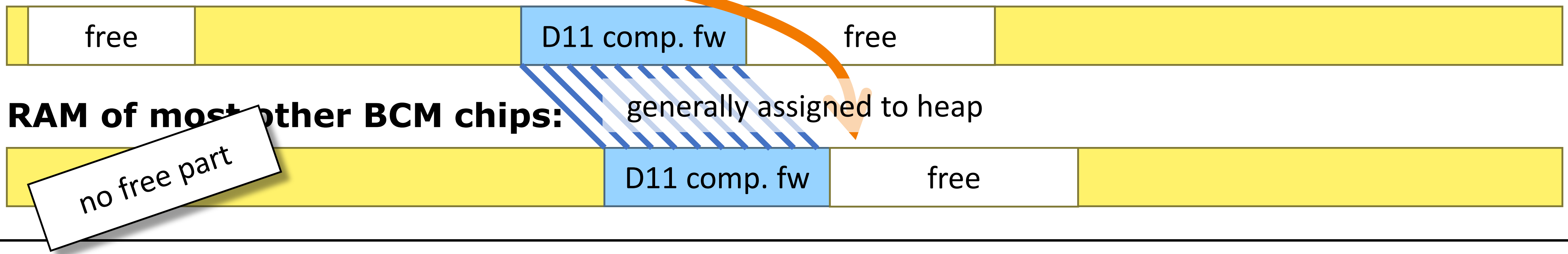
RAM of a BCM4339 in Nexus 5:

RAM of most other BCM chips:

Patch Placement

```
#pragma NEXMON targetregion "patch"

void target_function_name(int a) {
    so_something(a);
}

__attribute__((at(0x2220A, ”flashpatch", CHIP_..., FW_...)))
BLPatch(hook_name, target_function_name);
```

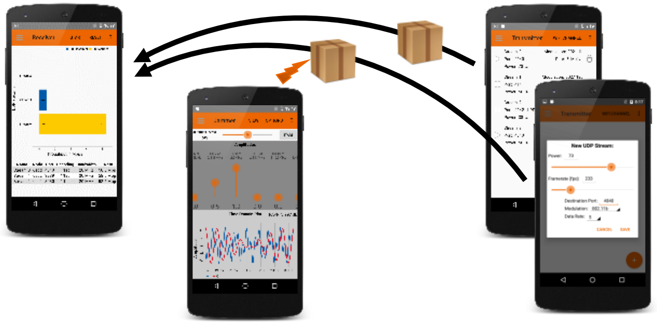
RAM of a BCM4339 in Nexus 5:



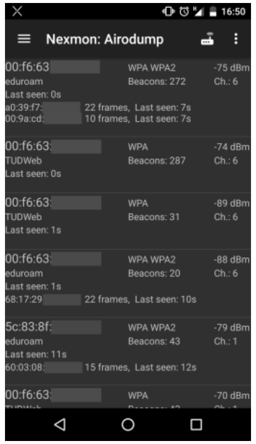
RAM of most other BCM chips:

Applications using Nexmon

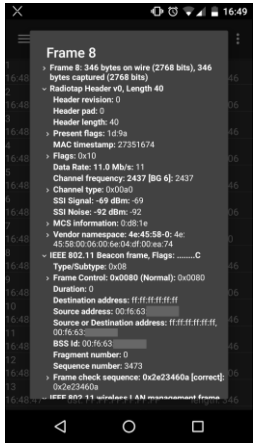
Massive Reactive Smartphone-Based Jamming using Arbitrary Waveforms and Adaptive Power Control, Best Paper at WiSec’17



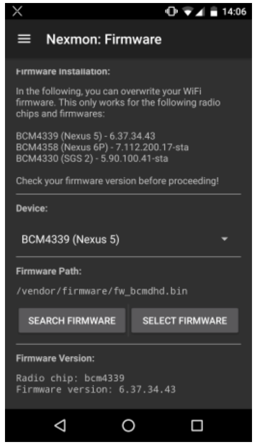
Nexmon Penetration Testing App with Monitor Mode, Frame Injection, Airodump View and Wireshark Dissection, Google Play Store



```
16:50
Nexmon: Airodump
00:f6:63[redacted]          WPA WPA2       -75 dBm
eduroam                    Beacons: 272   Ch.: 6
Last seen: 0s
a0:39:f7:[redacted] 22 frames,  Last seen: 7s
00:9a:cd:[redacted] 10 frames,  Last seen: 7s

00:f6:63:[redacted]         WPA            -74 dBm
TUDWeb                     Beacons: 287   Ch.: 6
Last seen: 0s

00:f6:63:[redacted]         WPA            -89 dBm
TUDWeb                     Beacons: 31    Ch.: 6
Last seen: 1s

00:f6:63:[redacted]         WPA WPA2       -88 dBm
eduroam                    Beacons: 20    Ch.: 6
Last seen: 1s
68:17:29[redacted] 22 frames,  Last seen: 10s

5c:83:8f:[redacted]         WPA WPA2       -79 dBm
eduroam                    Beacons: 43    Ch.: 1
Last seen: 11s
60:03:08:[redacted] 15 frames,  Last seen: 12s

00:f6:63:[redacted]         WPA            -70 dBm
```



```
16:49
Frame 8
› Frame 8: 346 bytes on wire (2768 bits), 346
  bytes captured (2768 bits)
⌄ Radiotap Header v0, Length 40
    Header revision: 0
    Header pad: 0
    Header length: 40
  › Present flags: 1d:9a
    MAC timestamp: 27351674
  › Flags: 0x10
    Data Rate: 11.0 Mb/s: 11
    Channel frequency: 2437 [BG 6]: 2437
  › Channel type: 0x00a0
    SSI Signal: -69 dBm: -69
    SSI Noise: -92 dBm: -92
  › MCS information: 0:d8:1e
  › Vendor namespace: 4e:45:58-0: 4e:
    45:58:00:06:00:6e:04:df:00:ea:74
⌄ IEEE 802.11 Beacon frame, Flags: ........C
    Type/Subtype: 0x08
  › Frame Control: 0x0080 (Normal): 0x0080
    Duration: 0
    Destination address: ff:ff:ff:ff:ff:ff
    Source address: 00:f6:63[redacted]
    Source or Destination address: ff:ff:ff:ff:ff:ff,
    00:f6:63:[redacted]
    BSS Id: 00:f6:63[redacted]
    Fragment number: 0
    Sequence number: 3473
  › Frame check sequence: 0x2e23460a [correct]:
    0x2e23460a
```



14:06

Nexmon: Firmware

Firmware Installation:

In the following, you can overwrite your WiFi firmware. This only works for the following radio chips and firmwares:

BCM4339 (Nexus 5) - 6.37.34.43
BCM4358 (Nexus 6P) - 7.112.200.17-sta
BCM4330 (SGS 2) - 5.90.100.41-sta

Check your firmware version before proceeding!

Device: BCM4339 (Nexus 5)

Firmware Path: `/vendor/firmware/fw_bcmdhd.bin`

Firmware Version:

```
Radio chip: bcm4339
Firmware version: 6.37.34.43
```

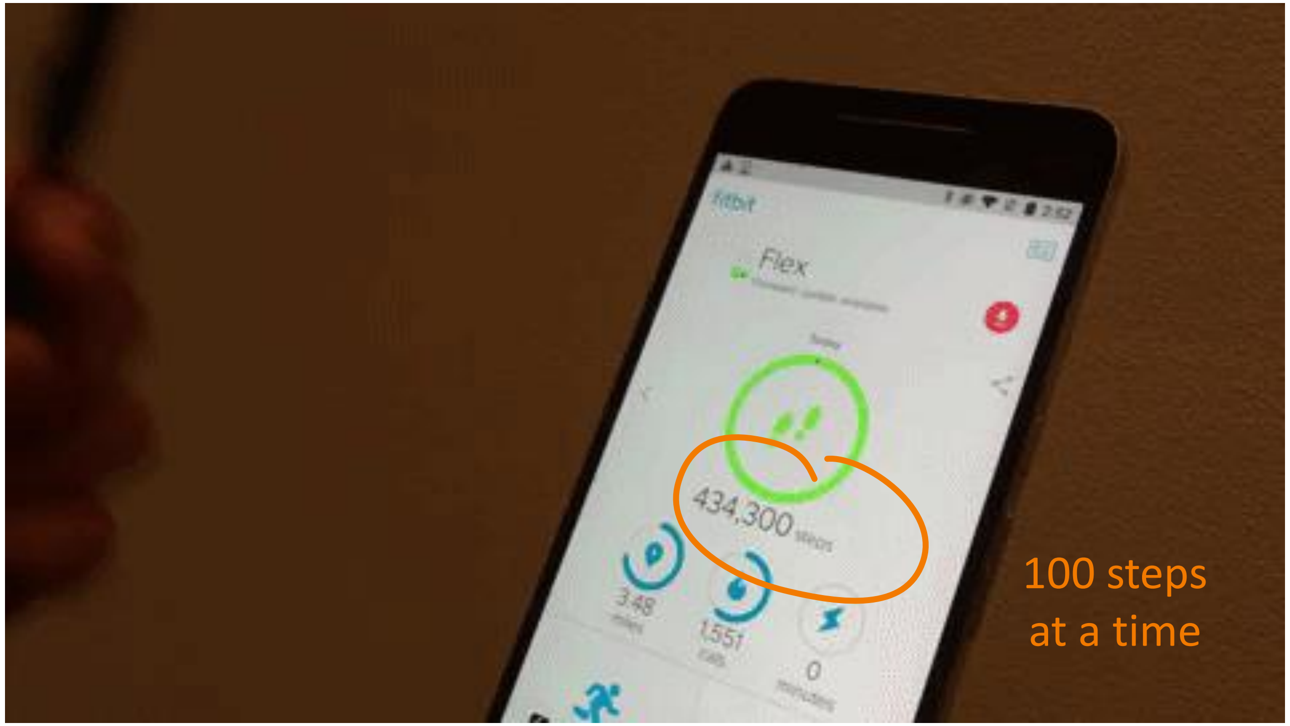
Compressive Millimeter-Wave Sector Selection in Off-the-Shelf IEEE 802.11ad Devices, CoNEXT’17

Source Code and Results available at https://nexmon.org

Third-Party Applications using Nexmon

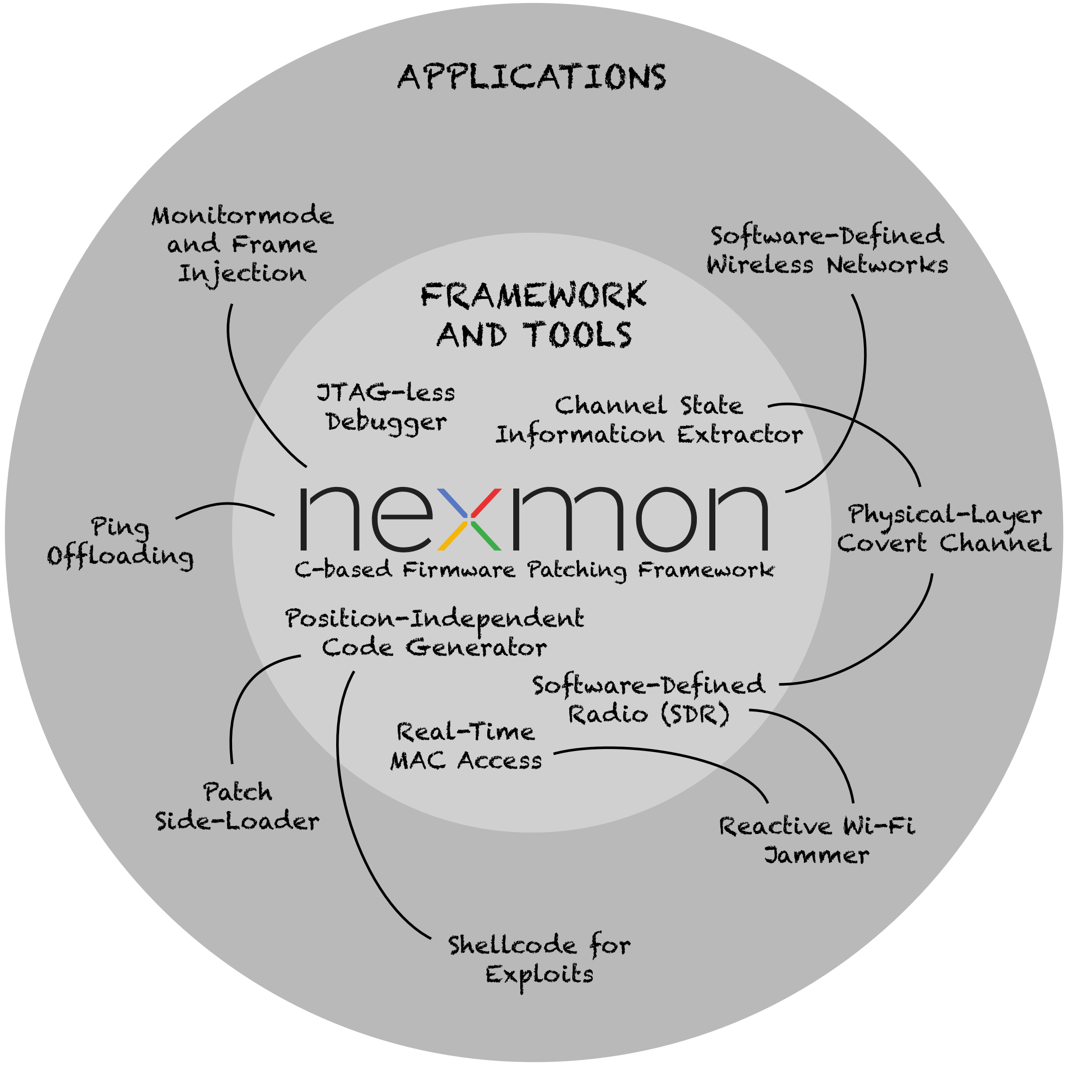
Reversing IoT: Xiaomi Ecosystem
Gain cloud independence and
additional functionality
by firmware modification,
Recon BRX ‘18

Breaking Fitness Records without Moving:
Reverse Engineering and Spoofing Fitbit,
RAID‘18



100 steps
at a time

More Exciting Nexmon Results



My PhD defense
February 26, 2018 at 1 pm
@ TU Darmstadt, Germany

SEEMOO

SECURE MOBILE NETWORKING

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Matthias Schulz
Department of Computer Science

SEEMOO
Mornewegstr. 32
64293 Darmstadt/Germany
mschulz@seemoo.tu-darmstadt.de

Twitter	@nexmon_dev
Phone	+49 6151 16-25478
Fax	+49 6151 16-25471

www.seemoo.tu-darmstadt.de