

# Lab 7

## Task 1

New output is:

```
16.119092] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
16.119101] Bluetooth: BNEP filters: protocol multicast
16.119119] Bluetooth: BNEP socket layer initialized
16.321456] Bluetooth: RFCOMM TTY layer initialized
16.321478] Bluetooth: RFCOMM socket layer initialized
16.321509] Bluetooth: RFCOMM ver 1.11
8824.974948] simple_module: loading out-of-tree module taints kernel.
8824.974966] simple_module: module license 'unspecified' taints kernel.
8824.974972] Disabling lock debugging due to kernel taint
8824.975329] Hello world.
```

What is the exception level and the security state of the core with loaded LKM?

Exception level: EL1

Security state: Non-secure state

## Task 2

a

Code is:

```
static int __init simple_module_init(void)
{
    uint32_t reg;
    asm volatile("mrc p15, 0, %0, c1, c1, 0": "=r"(reg));
    printk(KERN_INFO "SCR %x.\n", reg);
    return 0;
}
/*
 * The cleanup function of the module
```

```
pi@raspberrypi:~/Desktop/lab7 $ vim simple_module4.c
pi@raspberrypi:~/Desktop/lab7 $ sudo insmod simple_module4.ko
段错误
pi@raspberrypi:~/Desktop/lab7 $
```

Then, segment fault appears.

b

SCR is in EL3 but the code is in EL1 so the privilege is insufficient.

## Task 3

a

The manual's tutorial is

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2
--------	------	-----	-----	------

DBGAUTHSTATUS, Debug Authentication Status register

0b1110	0b000	0b0111	0b1110	0b110
--------	-------	--------	--------	-------

Therefore, instruction is `mrc p14, 0, Rt, c7, c14, 6` and the code is

```
static int __init simple_module_init(void)
{
    uint32_t reg;
    asm volatile("mrc p14, 0, %0, c7, c14, 6" : "=r"(reg));
    printk(KERN_INFO "SCR %x.\n", reg);
    return 0;
}
/*
```

b

Result is `ff`

```
[13004.110588] Code: e92dd800 e24cb004 e52de004 e8bd4000 (ee171f
[13004.116675] ---[ end trace 67d31cd50ebf7abe ]---
[13497.543969] SCR 10c5383d.
[15636.053142] Exit
[15637.349863] SCR ff.
pi@raspberrypi:~/Desktop/lab7 $ vim simple module4.c
```

According to manual, the result's meaning is following which show whether different types of debug are implemented or enabled.

7	6	5	4	3	2	1	0
SNID	SID	NSNID	NSID				

- SNID: Secure Non-Invasive Debug
- SID: Secure Non-Invasive Debug
- NSNID: Non-secure Non-invasive debug
- NSID: Non-secure Invasive Debug

The manual says that if value is `0b11` then it is implemented and enabled. Since my result is `ff`, all four types are implements and enabled.

## Task 4

1. offset

a. According to manual, I get offset of `EDLAR` which is `0xfb0`

## Accessing EDLAR

EDLAR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
Debug	0xFB0	EDLAR

This interface is accessible as follows:

- When FEAT\_DoPD is not implemented or IsCorePowered(), accesses to this register are **WO**.
- Otherwise, accesses to this register generate an error response.

b. I find offset of `OSLAR` (0x300) by looking for information of external register `OSLAR_EL1`.

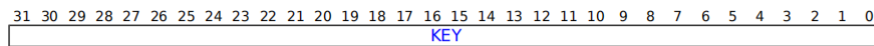
**OSLAR\_EL1** can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0x300	<b>OSLAR_EL1</b>

2. value

`iowrite32()` method write a value into an address.

a. Value `0xC5ACCE55` can unlock write accesses to debug registers according to manual of **EDLAR**.



**KEY, bits [31:0]**

Lock Access control. Writing the key value `0xC5ACCE55` to this field unlocks the lock, enabling write accesses to this component's registers through a memory-mapped interface.

Writing any other value to this register locks the lock, disabling write accesses to this component's registers through a memory mapped interface.

b. OS lock is determined by `OSLK` bit of `OSLAR_EL1` and after looking for `OSLSR_EL1` which provides status of OS lock, I know value `0` can unlock the OS lock.

**OSLK, bit [1]**

OS Lock Status.

OSLK	Meaning
0b0	OS Lock unlocked.
0b1	OS Lock locked.

The OS Lock is locked and unlocked by writing to the OS Lock Access Register.

3. Code

So the code is

```
#define EDLAR_OFFSET 0xfb0
#define OSLAR_OFFSET 0x300

// key for enabling debug
#define EDLAR_KEY 0xc5acce55
#define OSLAR_KEY 0x0

// Step 1: Unlock debug and cross trigger registers
printf(KERN_INFO "Step 1: Unlock debug and cross trigger register\n");
iowrite32(EDLAR_KEY, param->debug_register + EDLAR_OFFSET);
iowrite32(EDLAR_KEY, param->cti_register + EDLAR_OFFSET);
iowrite32(OSLAR_KEY, param->debug_register + OSLAR_OFFSET);
iowrite32(OSLAR_KEY, param->cti_register + OSLAR_OFFSET);
```

## Task 5

**a**

Instruction is `mrc p15, 0, r1, c1, c1, 0`. Using online tool in the Internet, I get machine code `\xee\x11\x1f\x11`. Therefore, code is:

```
execute_ins_via_itr(param->debug_register, 0x1f11ee11);
```

**b**

To transfer data from target to debugger, I need to firstly write value of `R1` into `DBGDTRTXint` register. The register is mapped to external register `DBGDTRTX_EL0` so I can use `ioread32()` method to read target value (`SCR`) from `DBGDTRTX_EL0`.

Code is following, and the first line corresponds to the first step and the second one corresponds to the next step.

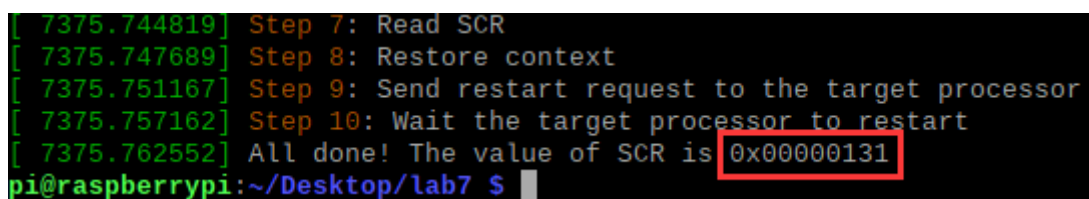
```
// mcr p14, 0, r1, c0, c5, 0
execute_ins_via_itr(param->debug_register, 0x1e15ee00);
scr = ioread32(param->debug_register + DBGDTRTX_OFFSET);
```

The whole code is:

```
// Step 7: Read the SCR
printf(KERN_INFO "Step 7: Read SCR\n");
// mrc p15, 0, r1, c1, c1, 0
execute_ins_via_itr(param->debug_register, 0x1f11ee11);
// mcr p14, 0, r1, c0, c5, 0
execute_ins_via_itr(param->debug_register, 0x1e15ee00);
scr = ioread32(param->debug_register + DBGDTRTX_OFFSET);
```

## Task 6

After successfully running the code, I get the value `0x131` which is `0b100110001`:



```
[ 7375.744819] Step 7: Read SCR
[ 7375.747689] Step 8: Restore context
[ 7375.751167] Step 9: Send restart request to the target processor
[ 7375.757162] Step 10: Wait the target processor to restart
[ 7375.762552] All done! The value of SCR is 0x00000131
pi@raspberrypi:~/Desktop/lab7 $
```

According to manual, each bit's meaning is:

1. bit [0]: PE is in non-secure state
2. bit [4]: An FIQ taken from either Security state is masked by `PSTATE.F`. When `PSTATE.F` is 0, the FIQ is taken to EL3.
3. bit [5]: External aborts taken from either Security state are masked by `PSTATE.A`. When `PSTATE.A` is 0, the abort is taken to EL3.
4. bit [8]: HVC instructions are enabled at Non-secure EL1 and EL2.

## Question 1

During this lab, what is the base address of Cross Trigger Interface in Raspberry Pi 3?

According to **Task 4**, base address of CTI is `0x40038000`

Can you find the global address of CTICONTROL register in Raspberry Pi 3 according to the Arm Reference Manual?

Offset	Name	Description
0x000	CTICONTROL	CTI Control register
0x010	CTIINTACK	CTI Output Trigger Acknowledge register
0x014	CTIAPPSET	CTI Application Trigger Set register

The offset is 0, so the address is `0x40038000 + 0 = 0x40038000`

## Question 2

Do we have another way to unlock the OS Lock in this lab except memory mapping?