How to setup Oprofile for iMX53 (ARM Cortex-A8)

In timer mode

1. **Setting up the kernel with menuconfig**

```
$ git clone git://sw-git.freescale.net/linux-2.6-imx.git
$ export CROSS_COMPILE=/opt/freescale/usr/local/gcc-4.4.4-glibc-2.11.1-multilib-1.0/arm-fsl-linux-gnueabi/bin/arm-none-li
$ cd linux-2.6-imx/
$ git checkout -b 2.6.38 --track origin/imx_2.6.38
$ make ARCH=arm CROSS_COMPILE=${CROSS_COMPILE} distclean
$ make ARCH=arm CROSS_COMPILE=${CROSS_COMPILE} imx5_defconfig
$ make ARCH=arm CROSS_COMPILE=${CROSS_COMPILE} menuconfig //described in next step
```

2. **Select the kernel modules when GUI based editor appears**

```
General Setup ---> Select [*] Profiling Support
General Setup ---> Select <*> OProfile system profiling
Kernel Hacking ---> Select [*] Kernel Debugging
<Exit>
<Exit> and Save will start the build
```

Note: do not select performance counters and monitoring support for time mode. i.e.:

```
General Setup ---> Kernel Performance Events And Counters  ---> [ ] Kernel performance events and counters
```

1. **Build the new kernel**

```
$ make ARCH=arm CROSS_COMPILE=${CROSS_COMPILE} uImage
```

2. **Find the new kernel**

```
// The new kernel image (uImage) is put in ../arch/arm/boot/
```

3. **Copy the new kernel image to SD card**

```
$ sudo dd if=uImage of=/dev/sdx bs=512 seek=2048 &sync
```

4. **Copy the vmlinux image to the /boot directory of the target rootfs**

```
$ sudo mount -l /dev/sdx1 /media/sdcard
$ sudo cp vmlinux /media/sdcard/boot/vmlinux
```

Using Event Mode and Counters

1. **Setting up the kernel with menuconfig**

```
$ git clone git://sw-git.freescale.net/linux-2.6-imx.git
$ export CROSS_COMPILE=/opt/freescale/usr/local/gcc-4.4.4-glibc-2.11.1-multilib-1.0/arm-fsl-linux-gnueabi/bin/arm-none-li
$ cd linux-2.6-imx/
$ git checkout -b 2.6.38 --track origin/imx_2.6.38
$ make ARCH=arm CROSS_COMPILE=${CROSS_COMPILE} distclean
$ make ARCH=arm CROSS_COMPILE=${CROSS_COMPILE} imx5_defconfig
$ make ARCH=arm CROSS_COMPILE=${CROSS_COMPILE} menuconfig //described in next step
```

2. **Select the kernel modules when GUI based editor appears**

```
General Setup ---> Kernel Performance Events And Counters  ---> [*] Kernel performance events and counters
General Setup ---> Select [*] Profiling Support
General Setup ---> Select <*> OProfile system profiling
Kernel Hacking ---> Select [*] Kernel Debugging
<Exit>
<Exit> and Save will start the build
```

3. **Modify the kernel source (2.6.38 in this example)**
   - In /arch/arm/kernel/pmu.c

```
static int __devinit pmu_device_probe(struct platform_device *pdev)
{
        printk("pmu_device_probe called \n" ); //Add this line
        if (pdev->id < 0 || pdev->id >= ARM_NUM_PMU_DEVICES) {
                pr_warning("received registration request for unknown
```

```
static int __devinit pmu_device_probe(struct platform_device *pdev)
{
        printk("pmu_device_probe called \n" ); //Add this line
        if (pdev->id < 0 || pdev->id >= ARM_NUM_PMU_DEVICES) {
```

```
static int __init register_pmu_driver(void)
{
        printk("register_pmu_driver called \n"); //Add this line
        return platform_driver_register(&pmu_driver);
}
device_initcall(register_pmu_driver);

struct platform_device *
reserve_pmu(enum arm_pmu_type device)
{
        struct platform_device *pdev;

        if (test_and_set_bit_lock(device, &pmu_lock)) {
                printk("test_and_set_bit_lock() failure \n"); //Add this line
                pdev = ERR_PTR(-EBUSY);
        } else if (pmu_devices[device] == NULL) {
                clear_bit_unlock(device, &pmu_lock);
                printk("pmu_devices[%d] == NULL  \n", device); //Add this line
                pdev = ERR_PTR(-ENODEV);
        } else {
                printk("pmu_devices[%d] reserved !!  \n", device);
                pdev = pmu_devices[device];
        }
```

```
        return pdev;
}
```

```
int
release_pmu(struct platform_device *pdev)
{
        printk("release_pmu called \n"); //Add this line
        if (WARN_ON(pdev != pmu_devices[pdev->id]))
                return -EINVAL;
        clear_bit_unlock(pdev->id, &pmu_lock);
        return 0;
}
```

```
init_cpu_pmu(void)
{
        int i, irqs, err = 0;
        struct platform_device *pdev = pmu_devices[ARM_PMU_DEVICE_CPU];
        printk("init_cpu_pmu called \n"); //Add this line
        if (!pdev)
```

```
init_pmu(enum arm_pmu_type device)
{
        int err = 0;
        printk("init_pmu called \n"); //Add this line
        switch (device) {
```

- In /arch/arm/mach-mx5/devices.c

```
#include <mach/imx-uart.h>
#include <mach/mx53.h> //Add this line
#include <mach/irqs.h>
#include <asm/pmu.h>
```

```
                .dma_mask = &usb_dma_mask,
                .coherent_dma_mask = DMA_BIT_MASK(32),
        },
};
static struct resource pmu_resources[] = { //Add this line
        { //Add this line
                .start = MX53_INT_PMU, //Add this line
                .end = MX53_INT_PMU, //Add this line
                .flags = IORESOURCE_IRQ, //Add this line
        }, //Add this line
}; //Add this line

struct platform_device pmu_device = { //Add this line
        .name = "arm-pmu", //Add this line
        .id = ARM_PMU_DEVICE_CPU, //Add this line
        .num_resources = ARRAY_SIZE(pmu_resources), //Add this line
        .resource = pmu_resources, //Add this line
}; //Add this line

static struct resource usbh2_wakeup_resources[] = {
        {
```

- In /arch/arm/mach-mx5/devices.h

```
extern struct platform_device mxc_usbh1_wakeup_device;
extern struct platform_device mxc_usbh2_wakeup_device;
extern struct platform_device pmu_device; //Add this line
```

- In /arch/arm/mach-mx5/board-mx53_loco.c

```
        imx53_add_ipuv3(0, &ipu_data);

        mxc_register_device(&pmu_device, NULL); //Add this line

        for (i = 0; i < ARRAY_SIZE(loco_fb_data); i++)
```

4. **Build the new kernel**

```
$ make ARCH=arm CROSS_COMPILE=${CROSS_COMPILE} uImage
```

5. **Find the new kernel**

```
// The new kernel image (uImage) is put in ../arch/arm/boot/
```

6. **Copy the new kernel image to SD card**

```
$ sudo dd if=uImage of=/dev/sdx bs=512 seek=2048 &sync
```

7. **Copy the vmlinux image to the /boot directory of the target rootfs**

```
$ sudo mount -l /dev/sdx1 /media/sdcard
$ sudo cp vmlinux /media/sdcard/boot/vmlinux
```

Running oprofile on i.MX53 target

1. **Change the name of vmlinux to vmlinux-'uname -r'**

```
$ cd /boot
$ sudo cp vmlinux vmlinux-'uname -r'
$ cd ~
```

2. **Install oprofile (on ubuntu)**

```
$ sudo apt-get install oprofile
```

Run oprofile in timer mode

1. **Command sequence**

```
$ sudo opcontrol --init
$ sudo opcontrol --vmlinux=/boot/vmlinux-'uname -r'
$ sudo opcontrol --start
$ sudo opcontrol --status
// do stuff
$ sudo opcontrol --dump
$ sudo opcontrol --stop
```

2. **Run opreport**

```
$ sudo opreport
```

Run oprofile in event mode

1. **Setting up RVDS**

To use oprofile in events mode you will need to connect the RVDS debugger to the ARM core via JTAG. In order to run the kernel while connceted to the JTAG debugger you must include the kernel argument "jtag=on" in the bootloader arguments. Once this is done you will also need to modify the settings of your ARM Cortex-A8 connection.

- Include 'jtag=on kernel argument (from uboot)
- Disable "semihosting" when running linux (from RVDS)
- Disable the vector catch (from RVDS)
- Change configuration "no reset, no stop" (from RVDS)

1. **Booting Linux**
    - Insert SD card
    - Reset board

- Boot to command propt
- Connect RVDS
- Run command sequence (below)

2. **Command sequence**

```
$ sudo opcontrol --init
$ sudo opcontrol --vmlinux=/boot/vmlinux-'uname -r'
$ sudo opcontrol --event=DCACHE_ACCESS:1000:0:1:1
$ sudo opcontrol --start
$ sudo opcontrol --status
// do stuff
$ sudo opcontrol --dump
$ sudo opcontrol --stop
```

3. **Run opreport**

```
$ sudo opreport
```