02-3 GPIO via mmap()

GPIO via sysfs

- So far we've been access the GPIO pins via sysfs
- You can turn a USR LED on with

```
bone$ cd /sys/class/leds/beaglebone\:green\:usr3
bone$ echo none > trigger
bone$ echo 1 > brightness
```

- sysf is portable, but can be slow
- What if speed is needed?

GPIO via mmap()

- All the IO on the am335x is memory mapped
- You can look them up on the am335x Technical Reference Manual (TRM)

USR3 LED

```
bone$ cd ~/exercises/gpio
bone$ sudo ./findGPIO.js USR3
{ name: 'USR3', gpio: 56, led: 'usr3',
mux: 'gpmc_a8', key: 'USR3',
muxRegOffset: '0x060',
options: [ 'gpmc_a8', 'gmii2_rxd3',
'rgmii2_rd3', 'mmc2_dat6', 'gpmc_a24',
'pr1_mii1_rxd0', 'mcasp0_aclkx', 'gpio1_24' ] }
USR3 (gpio 56) mode: 7 (gpio1_24) 0x060 pullup
pin 24 (44e10860): (MUX UNCLAIMED) (GPIO
UNCLAIMED)
```

GPIO port 1

From Table 2-3 of TRM

Table 2-3. L4_PER Peripheral Memory Map (continued)

Device Name	Start_address (hex)	End_address (hex)	Size	Description
DMTIMER5	0x4804_6000	0x4804_6FFF	4KB	DMTimer5 Registers
	0x4804_7000	0x4804_7FFF	4KB	Reserved
DMTIMER6	0x4804_8000	0x4804_8FFF	4KB	DMTimer6 Registers
	0x4804_9000	0x4804_9FFF	4KB	L4 Interconnect
DMTIMER7	0x4804_A000	0x4804_AFFF	4KB	DMTimer7 Registers
	0x4804_B000	0x4804_BFFF	4KB	Reserved
GPIO1	0x4804_C000	0x4804_CFFF	4KB	GPIO1 Registers
	0x4804_D000	0x4804_DFFF	4KB	Reserved
Reserved	0x4804_E000	0x4804_FFFF	8KB	Reserved

- Base address is **0x4804_C000.**
- Click on GPIO1

Table 25-5. GPIO Registers

Offset	Acronym	Register Name	Section
0h	GPIO_REVISION		Section 25.4.1.1
10h	GPIO_SYSCONFIG		Section 25.4.1.2
20h	GPIO_EOI		Section 25.4.1.3
24h	GPIO_IRQSTATUS_RAW_0		Section 25.4.1.4
28h	GPIO_IRQSTATUS_RAW_1		Section 25.4.1.5
2Ch	GPIO_IRQSTATUS_0		Section 25.4.1.6
30h	GPIO_IRQSTATUS_1		Section 25.4.1.7
34h	GPIO_IRQSTATUS_SET_0		Section 25.4.1.8
38h	GPIO_IRQSTATUS_SET_1		Section 25.4.1.9
3Ch	GPIO_IRQSTATUS_CLR_0		Section 25.4.1.10
40h	GPIO_IRQSTATUS_CLR_1		Section 25.4.1.11
44h	GPIO_IRQWAKEN_0		Section 25.4.1.12
48h	GPIO_IRQWAKEN_1		Section 25.4.1.13
114h	GPIO_SYSSTATUS		Section 25.4.1.14
130h	GPIO_CTRL		Section 25.4.1.15
134h	GPIO_OE		Section 25.4.1.16
138h	GPIO_DATAIN		Section 25.4.1.17
13Ch	GPIO_DATAOUT		Section 25.4.1.18
140h	GPIO_LEVELDETECT0		Section 25.4.1.19
144h	GPIO_LEVELDETECT1		Section 25.4.1.20
148h	GPIO_RISINGDETECT		Section 25.4.1.21
14Ch	GPIO_FALLINGDETECT		Section 25.4.1.22
150h	GPIO_DEBOUNCENABLE		Section 25.4.1.23
154h	GPIO_DEBOUNCINGTIME		Section 25.4.1.24
190h	GPIO_CLEARDATAOUT		Section 25.4.1.25
194h	GPIO_SETDATAOUT		Section 25.4.1.26

Table 25-5 of TRM

0x4804_c000 + 13c = 0x4804_c13c

Address for GPIO_DATAOUT

devmem2

- A program that reads/writes any memory location

```
bone$ sudo devmem2 0x4804c13c
/dev/mem opened.
Memory mapped at address 0xb6f99000.
Read at address 0x4804C13C (0xb6f9913c): 0x01800000
```

Physical Address | Virtual Address | Contents

gpio1_24

Bit 24 shows that status of the LED

```
bone$ wget http://free-electrons.com/pub/mirror/devmem2.c
bone$ gcc -o devmem2 devmem2.c
```

Toggle the LED - PIC

- The PIC way
  - Read register
  - XOR with (1<<24)
  - Write register
- 3 operations

Toggle the LED

- Use **GPIO_SETDATAOUT** and **GPIO_CLEARDATAOUT**

150h	GPIO_DEBOUNCENABLE	Section 25.4.1.23
154h	GPIO_DEBOUNCINGTIME	Section 25.4.1.24
190h	GPIO_CLEARDATAOUT	Section 25.4.1.25
194h	GPIO_SETDATAOUT	Section 25.4.1.26

- Write to **GPIO_SETDATAOUT** a value with 1's for the pins to be set to 1
- Write to **GPIO_CLEARDATAOUT** a value with 1's for the pins to be cleared to 0
- Use 0x190 to Clear

Turn LED off then on

GPIO_DATAOUT

- Off

```
bone$ sudo devmem2 0x4804c190 w 0x01000000
/dev/mem opened.
Memory mapped at address 0xb6f53000.
Read at address 0x4804C190 (0xb6f53190):  0x01800000
Write at address 0x4804C190 (0xb6f53190): 0x01000000,
readback 0x01000000
```

- On

```
bone$ sudo devmem2 0x4804c194 w 0x01000000
/dev/mem opened.
Memory mapped at address 0xb6f9f000.
Read at address 0x4804C194 (0xb6f9f194):  0x00800000
Write at address 0x4804C194 (0xb6f9f194): 0x01800000,
readback 0x01800000
```

mmap()

- The same can be done more quickly from a C program using **mmap()**
- **mmap()** is a way of mapping a physical address space into a user-space program

```
volatile void *gpio_addr;
volatile unsigned int *gpio_setdataout_addr;
int fd = open("/dev/mem", O_RDWR);
gpio_addr = mmap(0, GPIO1_SIZE, PROT_READ | PROT_WRITE,
                 MAP_SHARED, fd, GPIO1_START_ADDR);
gpio_setdataout_addr = gpio_addr + GPIO_SETDATAOUT;
   0x4804c190        0x4804c000       0x190
*gpio_setdataout_addr = USR3;
                      (1<<24)
```

Exercise GPIO via mmap

- Homework has you work through some examples
- **gpioThru.c** copies an input pin to an output