**1. 현재 GPIO의 Kernel Internal API**  
GPIO를 가지고 뭔가 하기 위해서 다음 헤더 파일을 include 한다.

#include <linux/gpio.h>

현재의 커널은 모든 GPIO를 unsigned integer로 나타낸다. 그래서 platform data나 device tree를 통해 GPIO 번호를 넘기는 것이 가능하다.  
  
GPIO는 사용 전에 allocation 되어야 한다. 현재 구현은 이를 강제하고 있지는 않다. allocation 은 다음 함수를 이용한다.

int gpio\_reqeust(unsigned int gpio, const char \*label);

label은 나중에 sysfs 로 나타날 수 있다. 반환값 0은 성공, 다른 값은 음수로 에러 번호를 반환한다. 다음을 통해 GPIO를 다시 돌려줄 수 있다.

void gpio\_free(unsigned int gpio);

몇 가지 변형이 있는데 gpio\_request\_one() 은 초기 설정을 같이 할 수 있고, gpio\_request\_array() 는 몇 개의 GPIO를 한번에 설정하면서 request 할 수 있다. "managed" 버전(예를 들면, devm\_gpio\_request())은 개발자가 잊어버렸을 때 cleanup을 자동적으로 처리한다.  
  
  
GPIO가 입력으로 사용되면,

int gpio\_direction\_input(unsigned int gpio);

출력으로 사용되면,

int gpio\_direction\_output(unsigned int gpio, int value);

output 일 때 value는 꼭 0 이나 1로 지정되어야 한다. 둘 다 성공 시에는 0을, 아니면 음수로 에러 번호를 반환한다.  
  
  
GPIO 입력값을 읽을 때는,

int gpio\_get\_value(unsigned int gpio);

에러가 발생하려면 gpio\_direction\_input()을 호출했을 때 에러가 발생한다고 생각하기 때문에 별도 에러 체크는 하지 않는다. 그러므로 gpio\_direction\_input()의 반환값을 꼭 체크해야 한다.  
  
  
GPIO 출력값을 설정할 때는 gpio\_direction\_output()을 사용할 수도 있지만, 이미 출력모드로 되어 있으면, 다음을 사용하는게 더 낫다.

void gpio\_set\_value(unsigned int gpio, int value);

몇몇 GPIO 컨트롤러는 GPIO 입력 값이 바뀌면 인터럽트를 걸어주는데, 이 IRQ 번호를 얻고 싶으면,

int gpio\_to\_irq(unsigned int gpio);

여기서의 gpio는 gpio\_request() 로 꼭 얻어야 하고, gpio는 일단 입력 모드로 된다. irq 번호 대신 음수로 된 에러 번호가 반환될 수 있다. 반환된 irq는 request\_irq()로 인터럽트 핸들러를 셋팅할 수 있다.  
  
  
GPIO를 sysfs로 export 해서 user space에서 제어할 수 있도록 하려면,

int gpio\_export(unsigned int gpio, bool direction\_may\_change);

direction\_may\_change는 보이는 것처럼, user space에서 direction을 바꿀 수 있는지 셋팅해주기 위한 것이다. 안하는 게 낫다. gpio\_unexport() 로 sysfs를 없앨  수 있고, gpio\_export\_link() 를 사용하면 다른 이름으로 export 할 수도 있다.  
  
더 많은 사항은 [Documentation/gpio.txt](http://lwn.net/Articles/532717/) 를 참고.  
  
  
2. Descriptor-based API (gpiod\_\*)  
  
현재의 GPIO API 가 integer를 사용하는데, 이 대신에 [descriptor-based GPIO 인터페이스](http://lwn.net/Articles/531848/)를 제공하는 방법이다. 이를 위해 struct gpio\_desc \* 포인터 타입을 사용한다. 이 인터페이스에 대한 반응은 꽤 좋다고 한다. 거의 곧 머지될 거 같다나..  
현재 GPIO API와 매우 비슷하다.

#include <linux/gpio/consumer.h>  
  
int gpiod\_direction\_input(struct gpio\_desc \*desc);  
int gpiod\_direction\_direction\_output(struct gpio\_desc \*desc, int value);  
int gpiod\_get\_value(struct gpio\_desc \*desc);  
void gpiod\_set\_value(struct gpio\_desc \*desc, int value);  
int gpiod\_to\_irq(struct gpio\_desc \*desc);  
int gpiod\_export(struct gpio\_desc \*desc, bool direction\_may\_change);  
int gpiod\_export\_link(struct device \*dev, const char \*name, struct gpio\_desc \*desc);  
void gpiod\_unexport(struct gpio\_desc \*desc);

gpio\_ prefix 대신 gpiod\_ prefix를 사용하고 GPIO 번호 integer대신 struct gpio\_desc \* 를 사용한다.  
  
GPIO 번호로 descriptor를 얻을 수도 있다.

struct gpio\_desc \*gpio\_to\_desc(unsigned gpio);

반대인 desc\_to\_gpio() 도 있다. desc로 gpio 번호를 얻으면 포인터에서 얻어진 것이므로 현재의 기존 GPIO API에 안심하고 쓸 수 있다. 그러나 GPIO 번호로 descriptor를 얻는 건 별로 좋지 않다. 그래서 다음을 제공한다.

struct gpio\_desc \*gpiod\_get(struct device \*dev, const char \*name);

dev는 GPIO 라인을 제공하는 device여야 하고, "name"은 라인을 말한다.  
GPIO 라인 접근을 제거하기 위해서 gpiod\_put() 도 제공한다.  
  
  
3. Block GPIO  
  
위에 설명한 것들은 개별 GPIO 라인을 관리하는데 초점을 둔다. 근데 GPIO들은 그룹으로 함께 쓰일 때가 많다.  
어떤 하드웨어들은 또 한번의 I/O 메모리 write 연산으로 여러 라인을 동시에 바꿀 수 있다. 그래서 여러 GPIO를 하나의 그룹처럼 조작할 때, 하나의 블록으로 엮어서 사용하기 위해 [Block GPIO 패치](http://lwn.net/Articles/533557/)가 시도되었다. Descriptor-based GPIO API 에 비하면 별로 크게 다루어지지는 않았다.

struct gpio\_block \*gpio\_block\_create(unsigned int \*gpios, size\_t size, const char \*name);

gpios는 하나의 블록으로 그룹지어지는, GPIO 번호들의 size 크기의 Array 이다. name은 user spcae에서 그 블록으로 동작하는데 사용될 수 있다. gpios 안의 GPIO 들은 gpio\_request()로 이전에 이미 요청되어져 있어야 하고, 그 direction 도 각각 셋팅되어져 있어야 한다. GPIO 들이 흩어져 있거나, 드라이버가 Internal Block APIO를 구현하지 않으면, 그냥 지금처럼 개별 라인으로 접근한다.  
  
GPIO 블록의 조작은,

unsigned long gpio\_block\_get(struct gpio\_block \*block, unsigned long mask);  
void gpio\_block\_set(struct gpio\_block \*block, unsigned long mask, unsigned long values);

mask 로 블록 안의 GPIO를 고를 수 있는데, mask는 위의 gpio\_block\_create() 로 전달되는 array에 해당하는 비트가 쓰인다. 여기서 알 수 있는 점은 한 개 블록의 GPIO 의 갯수는 long의 비트 수까지만 될 수 있다는 점이다.  
gpio\_block\_get() 은 가능하면 한꺼번에 지정된 라인들을 읽고, 비트 마스크로 결과를 반환한다. gpio\_block\_set()은 한꺼번에 셋팅할 수 있다.  
  
  
GPIO 볼록은 다음으로 제거 가능하다.

void gpio\_block\_free(struct gpio\_block \*block);

User space에서 sysfs 로 GPIO 블록을 셋팅하거나 읽을 수 있도록하는, 등록 함수도 있다.

int gpio\_block\_register(struct gpio\_block \*block);  
void gpio\_block\_unregister(struct gpio\_block \*block);

등록하면 gpio\_block\_create()에서 사용한 이름으로 device\_node도 생성된다. 그 디바이스를 읽으면, 블록 안의 GPIO의 상태를 반환하고, 쓰면, 그에 따른 GPIO를 셋팅할 수도 있다. read, write 동작에 사용하는 mask를 셋팅하는데-커맨드 번호로 0을 사용- ioctl() 도 제공한다.