IMX6 definitive GPIO guide

From KoanSoftware Wiki

Contents

Mapping names to Numbers

GPIOs on the i.MX6 linux system are confusing, at best.

There are several naming conventions at play.

Hardware Naming Convention

- Every ball on the physical i.MX6 chip has a "pad name" by Freescale.
- Each ball has a unique alphanumeric identifier which maps it to a location in the BGA.
- Each pad has a "canonical name". This is the name that is shown on the schematic symbol inside the part, next to the pin and pin number. It has no essential connection to the function's actual signal mapping.
- The schematics assign a "net name" to the functional wire connected to the pad. This attempts to be a description of what the wire is actually used for and doesn't attempt to reconcile against the pad name in any way.

There is a strict 1:1:1 mapping of balls to canonical names to net names. For example:

- Ball C20 <-> Pad SD1_DAT1 <-> Net FPGA_EXP_ON (FPGA expansion power on)
- Ball R6 <-> Pad GPIO_4 <-> Net SD2_CD (SD slot 2 card detect)
- Ball R20 <-> Pad DISP0_DAT13 <-> Net FPGA_RESET_N (FPGA reset, active low)

User manual/register map naming Convention

Unlike hardware names, pad names are re-used, and every pad has up to 8 potential actual functions.

The selection of these signal mappings is controlled by a pad/group register, whose name is derived from the canonical pad name. This is a bit confusing, because for example, a pad named SD1_DAT1 has a pad mux register named IOMUXC_SW_*MUX*_CTL_PAD_SD1_DATA1 (asterisk for emphasis only), and it's responsible for configuring SD1_DATA1, ECSPI5_SS0, PWM3_OUT, GPT_CAPTURE2, and GPIO1_IO17 function multiplexing to the SD1_DAT1 pad.

There's also IOMUXC_SW_*PAD*_CTL_PAD_SD1_DATA1 that's responsible for configuring the physical drive characteristic of the pad, which could, again, be mapped to any of the 5 functions.

Again, you can have up to 8 functions per pad, and importantly, it's not always the case that a pad's default function matches its canonical name.

Almost every pad has a GPIO function, and GPIO functions are internally tracked by a "bank/bit" convention. There are 7 banks of GPIOs with up to 32 bits each, and as is quite typical of hardware naming conventions, the index is 1-based, not 0 based; but the register addresses are all 0-based, requiring you to subtract 1 from the name.

Linux Userspace Naming Convention

- Almost every pad has, as one of its 8 possible functions, a GPIO role.
- Linux uses a single integer to enumerate all pads, therefore Freescale's bank/bit notation for GPIOs must be mapped.
- The bank/bit to Linux userspace formula is:

```
linux gpio number = (gpio_bank - 1) * 32 + gpio_bit
```

So, GPIO1_IO17 maps to (1 - 1) * 32 + 17 = 17.

A more complex example is DISP0_DAT13. It maps to GPIO5_IO07, so its userspace number is (5 - 1) * 32 + 7 = 135.

Thus, the algorithm to go from a schematic pin to a Linux userspace number is as follows:

1. Find the function on the schematic (e.g. PCIE_PWRON)
2. Trace it to the pad on a CPU. Note the canonical name (e.g. GPIO_17)
3. Look the canonical name up in table 28-4 of the reference manual to resolve the GPIO bit/bank notation (e.g., GPIO_17 = GPIO7_IO12), or search for the pad name in the table on this page (derived from the i.MX6Q Reference Manual Rev 1 4/2013).
4. Apply the translation formula: (7 - 1) * 32 + 12 = linux userspace gpio204

Note in the above example, the "pad name" GPIO_17 had *nothing* to do with the actual GPIO bit/bank notation. So don't let the names fool you. Think of pad names as unique strings with no inherent meaning, and your life will be easier.

Linux Device Tree Naming Convention

There's another place where you might want to call out a GPIO: the device tree.

Novena's device tree entry would be found in arch/arm/boot/dts/imx6q-novena.dts.

When referring to a GPIO in a device tree node, you use the bit/bank mapping scheme. For example, AUD_PWRON (schematic net name) maps to DISP0_DAT23 (i.MX6Q pad name). This maps to GPIO5_IO17 (per lookup in table below), and therefore the device tree entry for this is

```
power-gpio = <&gpio5 17 0>
```

The third field is for gpio polarity (0 = active high, 1 = active low). In the context of an interrupt, the third field specifies the polarity of the edge trigger, see Documentation/devicetree/bindings/gpio/fsl-imx-gpio.txt

Sometimes you have to "hog" a pin in the device tree. To do that, look for the node iomuxc, in the pinctrl_hog:hoggrp. Here, you have to use a series of names and numbers that you look up in the pin control file.

Easiest way I've found is to start by searching the bank/bit name in the file, whose name is "imx6q-pinfuc.h". Note the format for the name is "GPIOn_IOmm" (I always forget the IO before the bit name).

Searching the canonical pad name is confusing because the pad name is re-used up to 8 times, and it doesn't sufficiently disambiguate the function.

```
MX6QDL_PAD_DISP0_DAT23__GPIO5_IO17 0x80000000
```

According to Documentation/devicetree/bindings/pinctrl/fsl,imx6q-pinctrl.txt that number to the right of the hog spec can be used to set up things like pull-ups, pull-downs, keepers, drive strength, etc. The value 0x80000000 is special and means "I don't know and don't change from the default". Otherwise, it's set according to the following chart (basically the bitfields of the IOMUXC lower 17 bits):

```
PAD_CTL_HYS                     (1 << 16)
PAD_CTL_PUS_100K_DOWN           (0 << 14)
PAD_CTL_PUS_47K_UP              (1 << 14)
PAD_CTL_PUS_100K_UP             (2 << 14)
PAD_CTL_PUS_22K_UP              (3 << 14)
PAD_CTL_PUE                     (1 << 13)
PAD_CTL_PKE                     (1 << 12)
PAD_CTL_ODE                     (1 << 11)
PAD_CTL_SPEED_LOW               (1 << 6)
PAD_CTL_SPEED_MED               (2 << 6)
PAD_CTL_SPEED_HIGH              (3 << 6)
PAD_CTL_DSE_DISABLE             (0 << 3)
PAD_CTL_DSE_240ohm              (1 << 3)
PAD_CTL_DSE_120ohm              (2 << 3)
PAD_CTL_DSE_80ohm               (3 << 3)
PAD_CTL_DSE_60ohm               (4 << 3)
PAD_CTL_DSE_48ohm               (5 << 3)
PAD_CTL_DSE_40ohm               (6 << 3)
PAD_CTL_DSE_34ohm               (7 << 3)
PAD_CTL_SRE_FAST                (1 << 0)
PAD_CTL_SRE_SLOW                (0 << 0)
```

Pro tip: don't use a value of 0, it doesn't work.

Using Sysfsgpio on Linux

As an example, we'll use the controllable 5V power source.

The Device Tree lists it as gpio3,19:

```
gpios = <&gpio3 19 GPIO_ACTIVE_HIGH>
```

To get the Linux GPIO number, use the formula above: (bank - 1 * 32) + pin:

```
(3-1 * 32) + 19 = 83
```

Therefore, the 5v (fan)source is GPIO 83.

Expose the GPIO to userspace:

```
echo 83 > /sys/class/gpio/export
```

Make it an output:

```
echo out > /sys/class/gpio/gpio83/direction
```

And set it high:

```
echo 1 > /sys/class/gpio/gpio83/value
```

i.MX6Q GPIO bit/bank to Pad name Table

This table should *only* be used for the i.MX6Q. If you're mapping a Dual-Lite variant, please refer to the user manual (which you can download from the Freescale website).

```
GPIO1_IO00 - GPIO_0
GPIO1_IO01 - GPIO_1
GPIO1_IO02 - GPIO_2
GPIO1_IO03 - GPIO_3
GPIO1_IO04 - GPIO_4
GPIO1_IO05 - GPIO_5
GPIO1_IO06 - GPIO_6
GPIO1_IO07 - GPIO_7
GPIO1_IO08 - GPIO_8
GPIO1_IO09 - GPIO_9
GPIO1_IO10 - SD2_CLK
GPIO1_IO11 - SD2_CMD
GPIO1_IO12 - SD2_DAT3
GPIO1_IO13 - SD2_DAT2
GPIO1_IO14 - SD2_DAT1
GPIO1_IO15 - SD2_DAT0
GPIO1_IO16 - SD1_DAT0
GPIO1_IO17 - SD1_DAT1
GPIO1_IO18 - SD1_CMD
GPIO1_IO19 - SD1_DAT2
GPIO1_IO20 - SD1_CLK
GPIO1_IO21 - SD1_DAT3
GPIO1_IO22 - ENET_MDIO
GPIO1_IO23 - ENET_REF_CLK
GPIO1_IO24 - ENET_RX_ER
GPIO1_IO25 - ENET_CRS_DV
GPIO1_IO26 - ENET_RXD1
GPIO1_IO27 - ENET_RXD0
GPIO1_IO28 - ENET_TX_EN
GPIO1_IO29 - ENET_TXD1
GPIO1_IO30 - ENET_TXD0
GPIO1_IO31 - ENET_MDC
GPIO2_IO00 - NANDF_D0
GPIO2_IO01 - NANDF_D1
GPIO2_IO02 - NANDF_D2
GPIO2_IO03 - NANDF_D3
GPIO2_IO04 - NANDF_D4
GPIO2_IO05 - NANDF_D5
GPIO2_IO06 - NANDF_D6
GPIO2_IO07 - NANDF_D7
GPIO2_IO08 - SD4_DAT0
GPIO2_IO09 - SD4_DAT1
```

GPIO2_IO10 - SD4_DAT2
GPIO2_IO11 - SD4_DAT3
GPIO2_IO12 - SD4_DAT4
GPIO2_IO13 - SD4_DAT5
GPIO2_IO14 - SD4_DAT6
GPIO2_IO15 - SD4_DAT7
GPIO2_IO16 - EIM_A22
GPIO2_IO17 - EIM_A21
GPIO2_IO18 - EIM_A20
GPIO2_IO19 - EIM_A19
GPIO2_IO20 - EIM_A18
GPIO2_IO21 - EIM_A17
GPIO2_IO22 - EIM_A16
GPIO2_IO23 - EIM_CS0
GPIO2_IO24 - EIM_CS1
GPIO2_IO25 - EIM_OE
GPIO2_IO26 - EIM_RW
GPIO2_IO27 - EIM_LBA
GPIO2_IO28 - EIM_EB0
GPIO2_IO29 - EIM_EB1
GPIO2_IO30 - EIM_EB2
GPIO2_IO31 - EIM_EB3
GPIO3_IO00 - EIM_DA0
GPIO3_IO01 - EIM_DA1
GPIO3_IO02 - EIM_DA2
GPIO3_IO03 - EIM_DA3
GPIO3_IO04 - EIM_DA4
GPIO3_IO05 - EIM_DA5
GPIO3_IO06 - EIM_DA6
GPIO3_IO07 - EIM_DA7
GPIO3_IO08 - EIM_DA8
GPIO3_IO09 - EIM_DA9
GPIO3_IO10 - EIM_DA10
GPIO3_IO11 - EIM_DA11
GPIO3_IO12 - EIM_DA12
GPIO3_IO13 - EIM_DA13
GPIO3_IO14 - EIM_DA14
GPIO3_IO15 - EIM_DA15
GPIO3_IO16 - EIM_D16
GPIO3_IO17 - EIM_D17
GPIO3_IO18 - EIM_D18
GPIO3_IO19 - EIM_D19
GPIO3_IO20 - EIM_D20
GPIO3_IO21 - EIM_D21
GPIO3_IO22 - EIM_D22
GPIO3_IO23 - EIM_D23
GPIO3_IO24 - EIM_D24
GPIO3_IO25 - EIM_D25
GPIO3_IO26 - EIM_D26
GPIO3_IO27 - EIM_D27
GPIO3_IO28 - EIM_D28
GPIO3_IO29 - EIM_D29
GPIO3_IO30 - EIM_D30
GPIO3_IO31 - EIM_D31
GPIO4_IO05 - GPIO_19
GPIO4_IO06 - KEY_COL0
GPIO4_IO07 - KEY_ROW0
GPIO4_IO08 - KEY_COL1
GPIO4_IO09 - KEY_ROW1
GPIO4_IO10 - KEY_COL2
GPIO4_IO11 - KEY_ROW2
GPIO4_IO12 - KEY_COL3
GPIO4_IO13 - KEY_ROW3
GPIO4_IO14 - KEY_COL4
GPIO4_IO15 - KEY_ROW4
GPIO4_IO16 - DI0_DISP_CLK
GPIO4_IO17 - DI0_PIN15
GPIO4_IO18 - DI0_PIN2
GPIO4_IO19 - DI0_PIN3
GPIO4_IO20 - DI0_PIN4
GPIO4_IO21 - DISP0_DAT0
GPIO4_IO22 - DISP0_DAT1
GPIO4_IO23 - DISP0_DAT2
GPIO4_IO24 - DISP0_DAT3
GPIO4_IO25 - DISP0_DAT4

GPIO4_IO26 - DISP0_DAT5
GPIO4_IO27 - DISP0_DAT6
GPIO4_IO28 - DISP0_DAT7
GPIO4_IO29 - DISP0_DAT8
GPIO4_IO30 - DISP0_DAT9
GPIO4_IO31 - DISP0_DAT10
GPIO5_IO00 - EIM_WAIT
GPIO5_IO02 - EIM_A25
GPIO5_IO04 - EIM_A24
GPIO5_IO05 - DISP0_DAT11
GPIO5_IO06 - DISP0_DAT12
GPIO5_IO07 - DISP0_DAT13
GPIO5_IO08 - DISP0_DAT14
GPIO5_IO09 - DISP0_DAT15
GPIO5_IO10 - DISP0_DAT16
GPIO5_IO11 - DISP0_DAT17
GPIO5_IO12 - DISP0_DAT18
GPIO5_IO13 - DISP0_DAT19
GPIO5_IO14 - DISP0_DAT20
GPIO5_IO15 - DISP0_DAT21
GPIO5_IO16 - DISP0_DAT22
GPIO5_IO17 - DISP0_DAT23
GPIO5_IO18 - CSI0_PIXCLK
GPIO5_IO19 - CSI0_MCLK
GPIO5_IO20 - CSI0_DATA_EN
GPIO5_IO21 - CSI0_VSYNC
GPIO5_IO22 - CSI0_DAT4
GPIO5_IO23 - CSI0_DAT5
GPIO5_IO24 - CSI0_DAT6
GPIO5_IO25 - CSI0_DAT7
GPIO5_IO26 - CSI0_DAT8
GPIO5_IO27 - CSI0_DAT9
GPIO5_IO28 - CSI0_DAT10
GPIO5_IO29 - CSI0_DAT11
GPIO5_IO30 - CSI0_DAT12
GPIO5_IO31 - CSI0_DAT13
GPIO6_IO00 - CSI0_DAT14
GPIO6_IO01 - CSI0_DAT15
GPIO6_IO02 - CSI0_DAT16
GPIO6_IO03 - CSI0_DAT17
GPIO6_IO04 - CSI0_DAT18
GPIO6_IO05 - CSI0_DAT19
GPIO6_IO06 - EIM_A23
GPIO6_IO07 - NANDF_CLE
GPIO6_IO08 - NANDF_ALE
GPIO6_IO09 - NANDF_WP_B
GPIO6_IO10 - NANDF_RB0
GPIO6_IO11 - NANDF_CS0
GPIO6_IO14 - NANDF_CS1
GPIO6_IO15 - NANDF_CS2
GPIO6_IO16 - NANDF_CS3
GPIO6_IO17 - SD3_DAT7
GPIO6_IO18 - SD3_DAT6
GPIO6_IO19 - RGMII_TXC
GPIO6_IO20 - RGMII_TD0
GPIO6_IO21 - RGMII_TD1
GPIO6_IO22 - RGMII_TD2
GPIO6_IO23 - RGMII_TD3
GPIO6_IO24 - RGMII_RX_CTL
GPIO6_IO25 - RGMII_RD0
GPIO6_IO26 - RGMII_TX_CTL
GPIO6_IO27 - RGMII_RD1
GPIO6_IO28 - RGMII_RD2
GPIO6_IO29 - RGMII_RD3
GPIO6_IO30 - RGMII_RXC
GPIO6_IO31 - EIM_BCLK
GPIO7_IO00 - SD3_DAT5
GPIO7_IO01 - SD3_DAT4
GPIO7_IO02 - SD3_CMD
GPIO7_IO03 - SD3_CLK
GPIO7_IO04 - SD3_DAT0
GPIO7_IO05 - SD3_DAT1
GPIO7_IO06 - SD3_DAT2
GPIO7_IO07 - SD3_DAT3
GPIO7_IO08 - SD3_RST

```
GPIO7_IO09 - SD4_CMD
GPIO7_IO10 - SD4_CLK
GPIO7_IO11 - GPIO_16
GPIO7_IO12 - GPIO_17
GPIO7_IO13 - GPIO_18
```

Credits : Sutajio Ko-Usagi's wiki

GPIO read and write

A very frequent misunderstanding when using GPIOs is that sometimes you could pretend to read the state of an output.

In a sentence : **Is not possible to read the state of an output**.

An output can be only written, and an input can be only read.

For example let's consider the i.MX 6ULL Applications Processor Reference Manual (https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors/i-mx-6-processors/i-mx-6ull-single-core-processor-with-arm-cortex-a7-core:i.MX6ULL)

In chapter 28.4.1 "GPIO Function"

A GPIO signal can operate as a general-purpose input/output when the IOMUX is set to GPIO mode. Each GPIO signal may be independently configured as either an input or an output using the GPIO direction register (GPIO_GDIR). When configured as an output (GPIO_GDIR bit = 1), the value in the data bit in the GPIO data register (GPIO_DR) is driven on the corresponding GPIO line. When a signal is configured as an input (GPIO_GDIR bit = 0), the state of the input can be read from the corresponding GPIO_PSR bit.