# The Analysis of PIN CONTROL Subsystem

Nadya A. Mohamed and Xu Chen
ECE Department, Rice University

## Abstract

In this report, we focus on a specific Linux subsystem - the PIN CONTROL subsystem, which is a centralized mechanism that can arrange and configure pins. We discuss the hardware background of the PIN CONTROL subsystem, the reason why it is necessary, and its three major functions. Finally, we perform our own analysis for this subsystem. We leverage cloc and cscope as analyzers, and observe several interesting facts about PIN CONTROL subsystem through the analysis.

## 1. Introduction

Pins are equal to pads, fingers, balls or whatever packaging input or output line you want to control. A very large number of different types of packages exist and some of the most common are, dual in-line package, ball grid array, and quad flat packages.

In a system on chip (SoC), pins are connected to pads on a silicon die (chip). Each pad is driven by a specific piece of logic called a cell. The basic components of the cell as shown in Fig. 1 are the output driver, comparator circuit (Schmitt Trigger), pull-up resistor and pull down resistors. Output drivers are used to make sure that the output signal will be able to drive the output load. The Schmitt Trigger is an electronic circuit that provides two different threshold voltage levels for rising and falling edge to avoid the errors when we have noisy input signals. Pull up and pull down resistors are used to ensure a well-defined logical level at a pin under all conditions. Each cell has a number of discrete control signals such as: OEN output enable, IEN input enable, PUEN pull-up enable, and PDEN pull-down enable. The cell control signals may be hard-coded to certain state or connected to register maps that are controlled by software. The pad cell could be used for various things and one specific use is general purpose input output (GPIO).


ISBN [to be supplied]. . . $15.00

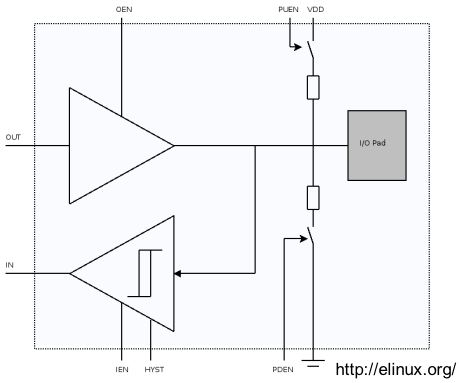



Figure 1: Pad Cell Components and Control Signals

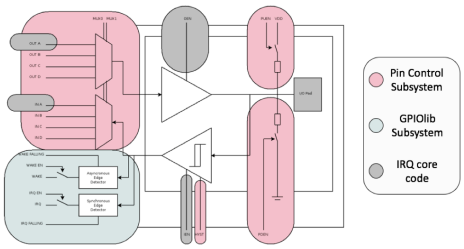



Figure 2: GPIO Logic

A GPIO block is formed by taking a number of pad cells and lumping them into a block, i.e. 8 cells to create 8 bits GPIO block. Control registers for each cell are then typically grouped into a 32 bit register where you can set any control signal by setting or clearing bit N in the register. The GPIO logic typically adds asynchronous and synchronous edge detection logic for rising, falling or both edges when the pin is used as an input and builds interrupt request (IRQ) signals and wakeup signals from edges. IRQ and Wakeup signal are used to interrupt the processor from external source or to wake up the processor if it is in a sleep mode. GPIO logic also adds pin multiplexing in order to be able to use each pin for other things than GPIO. Fig. 2 shows the details of GPIO logic.

In Linux Kernel three subsystems are involved at once to control the piece of hardware shown in Fig. 2, GPIOlib subsystem handles the GPIO pin portions, setting pins to in-

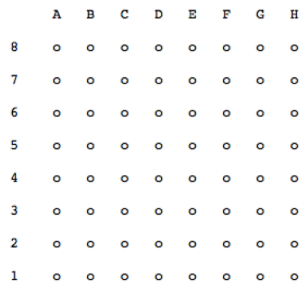



Figure 3: A single PGA chip [3]

put or output (selecting direction) and driving them high or low, it also interacts with IRQ core to map a certain pin to a certain interrupt line. The IRQ core code (irqchips.h) handles the IRQ enable/disable, edge selection and wakeup enable/disable for the pin [2]. Pin control and pin multiplexing is the subject addressed by the pin control subsystem that is covered in this paper.

## 2. PIN CONTROL Subsystem

A system-on-chip (SOC) has plenty of pins on it. To ensure that a SOC work can normally, all pins should be properly configured. However, managing such complex configuration is quite challenging since the correct pin configuration is board-specific, and hence lots of duplicate codes are required in the pin configuration [2].

The PIN CONTROL subsystem, as a centralized mechanism, has three major functions that help address these challenges, pin enumerating, pin multiplexing (e.g., the way to reuse a single pin or a single group of pins for different purposes), and pin configuration [3].

### 2.1 Pin Enumerating

An important function of PIN CONTROL subsystem is enumerating, which means that it can enumerate all controllable pins that a specific processor provides. In the example of a PGA chip board seen from the underneath, as shown in the Fig. 3, we observe that PIN CONTROL subsystem is able to register a pin controller and name all pins by associating them with integers [3]. These integers are called pin numbers. Fig. 4 displays a typical industrial standard to name all controllable pins in a given board. Specifically, the pins A8, B8, C8 are named as 0, 1, 2 in the beginning, and the pins F1, G1, H1 are denoted as 61, 62, 63 in the end.

It is important to note that this naming strategy is only a typical example. Other naming principles are also allowed. The association between pins and pin numbers can be regarded as a bridge that connects the hardware and software, hence making further analysis from the perspective of software much easier.

In addition, controllers usually deal with groups of pins when performing functions, instead of a single pin. Therefore, a mechanism that can enumerate and retrieve ”groups

```
#include <linux/pinctrl/pinctrl.h>

const struct pinctrl_pin_desc foo_pins[] = {
      PINCTRL_PIN(0, "A8"),
      PINCTRL_PIN(1, "B8"),
      PINCTRL_PIN(2, "C8"),
      ...
      PINCTRL_PIN(61, "F1"),
      PINCTRL_PIN(62, "G1"),
      PINCTRL_PIN(63, "H1"),
};

static struct pinctrl_desc foo_desc = {
        .name = "foo",
        .pins = foo_pins,
        .npins = ARRAY_SIZE(foo_pins),
        .owner = THIS_MODULE,
};

int __init foo_probe(void)
{
        int error;

        struct pinctrl_dev *pctl;

        error = pinctrl_register_and_init(&foo_desc, <PARENT>, NULL, &pctl);
        if (error)
                return error;

        return pinctrl_enable(pctl);
}
```

Figure 4: Enumerating pins [3]

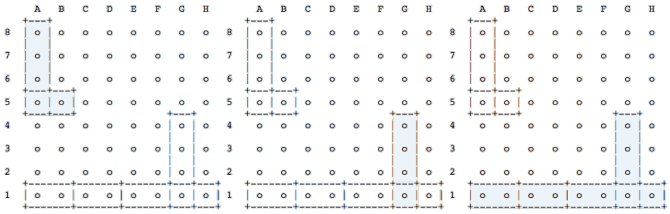
of pins” is highly desirable [2]. These groups of pins are defined as the pin groups, which often correspond to a specific function [3]. For example, we can have one groups of pins 0, 8, 16, 24 (i.e., pin numbers) dealing with SPI interface, and the other groups of pins 24, 25 dealing with I2C interface. Once the pin groups are clearly defined, then the function in PINCTRL code get_group_count will be able to determine the total number of legal selectors needed for other functions to retrieve the names and pins of each pin group [3].

### 2.2 PINMUX

Given a system-on-chip, there are multiple ways to utilize its pins. However, a common situation that users might encounter is that some pins need to be reused for different tasks. To this end, a channel that can regulate the way to reuse pins for multiple mutually exclusive functions becomes necessary. PIN CONTROL subsystem has the PINMUX functionality, which helps address this issue.

Fig. 5 shows a example of how the PINMUX mechanism works for a single chip. This example again utilizes a single PGA chip seen from the underneath [3]. When this chip is used, some pins will be taken by feeding power to the chip. They will be connected to VCC or GND. Another groups of pins will be occupied by large ports, such as the external memory interface [2]. After then, the remaining pins on the board will be subject to pin multiplexing (i.e., PINMUX) [3].

In the case where the pins A8, A7, A6, A5 are used to deal with the SPI port, and the pins A5, B5 are exploited to deal with I2C port, we observe that the A5 pin is actually ”shared” by two ports. Due to this sharing, SPI and I2C ports could not be used at the same time. However, PINMUX settings enable the SPI logic to be routed out on the pins G4, G3, G2, G1, such that the conflict at A5 can be eliminated, and the SPI, I2C ports then can be exploited simultaneously. Similarly, while a 8-bit MMC bus is being used, the routed SPI port then will not be allowed to be utilized at the same time. PINMUX settings control all these processes [3].



(a) Pins for SPI and I2C (b) Routed pins for SPI (c) SPI and MMC (8-bit)

Figure 5: Example of PINMUX use [3]

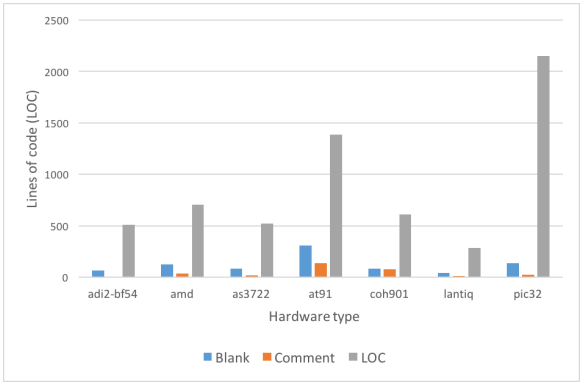



Figure 6: LOC of PINCTRL for various hardware v.4.11

### 2.3 Pin Configuration

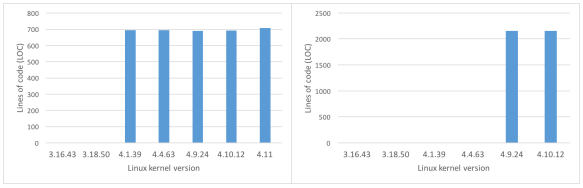
The last function of the PINCTRL subsystem is pin configuration. The key idea of pin configuration is that pins can be software-configured since pins also have electronic properties [3].

Specifically, in the Fig. 1, once the switch in the upper of this circuit is closed, an input pin will be attached to VDD to provide power. We also notice that there is a pull-up resistor, which ensures that the input signal will be maintained at a stable level even when pins are disconnected, or when output pins are connected to a high impedance [5]. The lower part of this circuit contains a pull-down resistor, which serves a similar purpose, keeping the signal at low level when the pin is unconnected [6]. Pin configuration codes integrated in the PINCTRL subsystem is able to help set all these electronic operations of pins (i.e., hardware operation) from the software perspective [3].

## 3. PINCTRL Subsystem Analysis

### 3.1 Line of Codes (LOC) Analysis

To start with, we first discuss our findings in the lines of code (LOC) analysis. Fig. 6 displays the number of physical code lines, comment lines, and blank lines in the PINCTRL codes designed for multiple hardware in the version 4.11, the latest version we study. We pick up the several typical hardware examples that PINCTRL codes can work for, including ADI




(a) PINCTRL for hardware amd (b) PINCTRL for hardware pic32

Figure 7: LOC revolution of PINCTRL for two hardware

GPIO2 controller on bf54x processor (i.e., adi2-bf54) [7], GPIO driver for AMD (amd) [8], the power management unit AS3722 (as3722) [9], Atmel AT91SAM SoC (at91) [10], ST-Ericsson U300 Series (coh901) [11], Lantiq SOC (lantiq) [12], and PIC32 pin controller driver (pic32) [3]. In Fig. 6, the major observation is that most PINCTRL codes do have much shorter comment lines compared to the physical lines of code (LOC). We envision that it is partly since the documentation [3] is comprehensive enough such that only a few of comment lines are in need to highlight the hardware-specific requirements.

As shown in Fig. 7, we also study the temporal development of the LOC in PINCTRL codes. According to the official document [4], the overall PINCTRL subsystem was added to the Linux kernel from the version 3.13. The codes designed for GPIO driver of AMD, and PIC32 pin controller driver, were not developed until the version 4.1.39 and 4.9.24, respectively [4]. But after they were introduced, these two codes have only experienced minor modifications in the past few versions.

Then we extend our analysis to the version-by-version difference of LOC in PINCTRL codes for different hardware. As shown in Fig. 12, we note that the PINCTRL codes for ADI GPIO2 controller and Lantiq SOC experience no change starting from their released dates. That means codes designed for them are comparably robust for long-term use. On the other hand, PINCTRL codes designed for Atmel AT91SAM SoC have been extended a lot from 3.16 to 3.18, and are shrinked from the version 4.1.39 to 4.9.24, which indicates the features in the initial version might not meet user demands. Finally, the PINCTRL subsystem for the power management unit AS3722 and ST-Ericsson U300 Series both have experienced minor changes from one version to another.

The above hardware's PINCTRL codes show that starting from the Linux kernel version 3.13 where the PINCTRL subsystem is initially introduced, the large-scale revisions of PINCTRL codes do not happen very often. Thus, generally the PINCTRL subsystem is stable and robust in the long-term use.

### 3.2 Evolution Analysis

In this section, we investigate the evolution of the Linux pin control subsystem using code authorship and copyright mea-

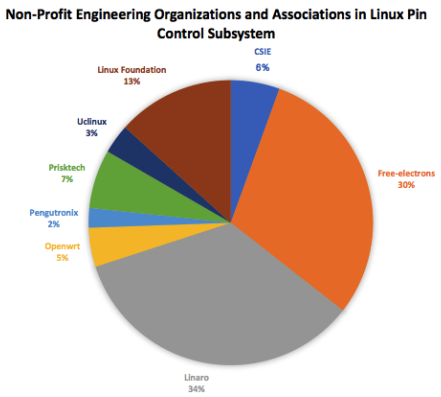



Figure 8: Participation of non-profit engineering organizations and associations in PINCTRL subsystem

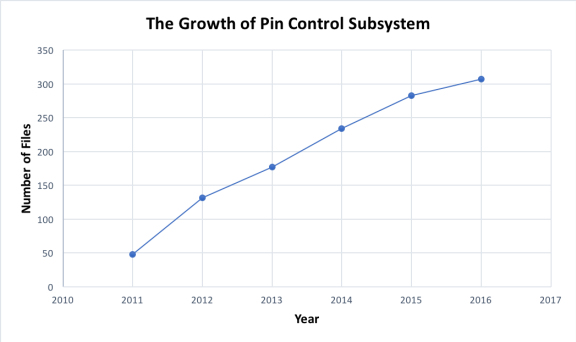



Figure 9: The growth of PINCTRL subsystem

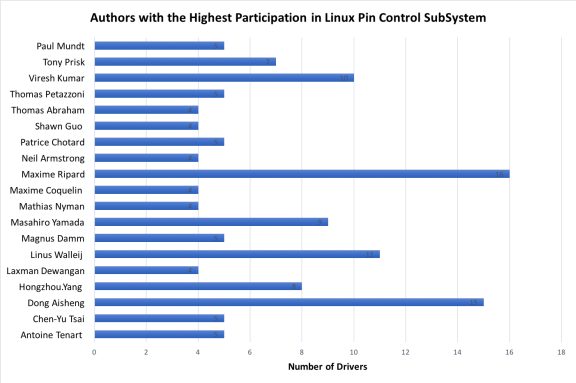



Figure 10: Authors with the highest participation in PINCTRL subsystem

sures. We provided answers to four main questions: (1)What is the proportion of Non-profit technology associations in Linux pin control subsystem development? (2)What is the number of files per author?(3)How the pin control subsystem has grown through the years?(4)What is the proportion of technology corporates in Linux pin control subsystem development?

Our analysis was based on Linux v.4.10 available at [4] and Cscope [14] as a tool for browsing the source code authorship and copyright information.

#### 3.2.1 The proportion of non-profit engineering organizations and associations

The pie chart in Fig. 8 shows that Linaro engineering organization has the highest proportion in Linux pin control subsystem, followed by Free-electrons organization. Both organizations focus in embedded Linux and more generally in free and open source software for embedded systems. The Linux Foundation which is anon-profittechnologytrade association came in third place. The remaining proportions were by smaller associations compared with the previous three.

#### 3.2.2 The Growth of PINCTRL Subsystem

As mentioned above, we used copyright year as an indication of the growth of Linux pin control subsystem over years. For, example the number of copyrights that belong to the year 2011 were around 50 as shown in Fig. 9. In 2012, there were around 84 copyrights, which lead the total number of copyrights to grow to 132 in 2012. In 2013, the growth was less than 2012. From the line graph, we can conclude that the Linux pin control subsystem is growing and it will continue to grow through the coming years with the upcoming technologies.

#### 3.2.3 The Author Analysis of PINCTRL Subsystem

The bar graph in Fig. 10 shows the developers with the highest participation in Linux pin control subsystem. Maxime Ripard who is an embedded Linux engineer at Free Electrons organization had the highest contribution of total of sixteen drivers. The second, third and forth places were taken by Linaro organization engineers, Dong Aisheng, Linus Walleij, and Veresh Kumar consecutively. The results we get in this analysis agree with the results we get from analyzing the proportion of non-profit engineering organizations and associations.

#### 3.2.4 Technology Corporates in PINCTRL Subsystem

The, last analysis was to measure technology corporates participation in the growth of Linux pin control subsystem. The bar graph in Fig. 11 shows that the semiconductor manufacturers Renesas Electronics Corp., Freescale Semiconductor Inc., Allwinner Technology have the highest number of drivers, followed by Nvidia Corp. which is a graphics processing units design corporation and ST-Ericsson, an industry leader in design, development and creation of mobile platforms and semiconductors.

### 3.3 Methods, Tools, and Procedures

To start with, we download all source codes of Linux Kernel after the version 3.16.43, from the website [1] to perform our analysis. It is because the PINCTRL subsystem is introduced

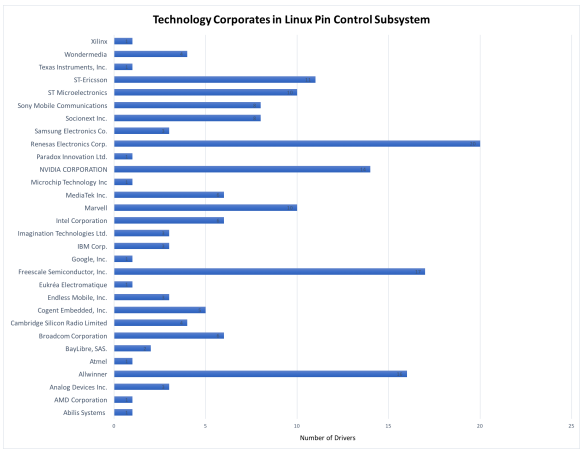



Figure 11: Technology corporates in PINCTRL subsystem

after the version 3.13. The PINCTRL codes are stored in the directory "Linux/drivers/pinctrl".

In LOC analysis, we leverage the tool **cloc** [13]. This toolkit can display the lines of code (LOC) for all the files in a single folder. It can deal with the files written in different languages (e.g., C, C++, HTML, XML, etc).

After installing this toolkit using Homebrew, we then place the PINCTRL target codes we aim to analyze (i.e., codes for different hardware) to different folders. The use of toolkit is based on command lines in Mac OSX Terminal. Using command lines, we first reach different folders that store the PINCTRL codes for different hardware, and then call **cloc** to detect physical lines, comment lines and blank lines for all files in different folders.

Moreover, this tool has several interesting functions, such as removing comments from source code, and analyzing codes within compressed archives. We also play with these functions during this project.

In evolution analysis, we used authorship as a measure of the amount of work performed by non-profit technology organizations and associations, and the amount of work performed by Linux pin control subsystem authors/developers. We used copyright year as an indication of the growth, and as a measure of technology corporates participation in Linux pin control subsystem.

As mentioned previously the evolution analysis was based on Linux v.4.10 and Cscope as an analyzer for browsing the source code authorship and copyright information. Cscope has been installed and used in a virtual box Linux Mint 18.

## 4. Conclusion

In this report, we studied the PINCTRL subsystem extensively by examining its background information (i.e., hardware) and its three major functions. We then perform our own analysis for the codes of PINCTRL subsystem, using cloc and cscope tools. This process is fruitful and we have found a lot of interesting facts about the LOC, the authors, and the technology associations related to the PINCTRL subsystem.

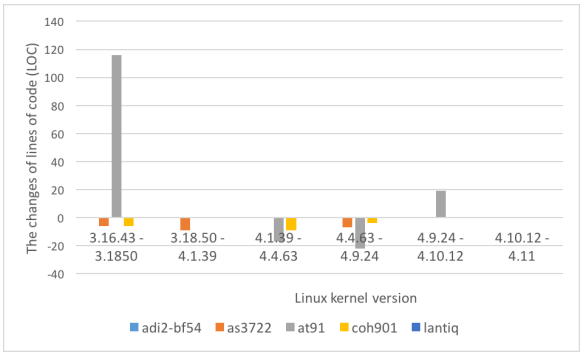



Figure 12: LOC difference across different versions

## References


[1] The Linux Kernel Archives, https://www.kernel.org/

[2] The pin control subsystem, https://lwn.net/Articles/468759/

[3] PINCTRL subsystem https://www.kernel.org/doc/Documentation/pinctrl.txt

[4] Linux Cross References, http://lxr.free-electrons.com/source/drivers/pinctrl/

[5] Pull-up resistor, https://en.wikipedia.org/wiki/Pull-up_resistor

[6] Pull-down resistor, http://playground.arduino.cc/CommonTopics/PullUpDownResistor

[7] Pinctrl: ADI PIN control driver for the GPIO controller on bf54x and bf60x, https://lwn.net/Articles/565586/

[8] AMD GPIO Driver http://drivers.softpedia.com/downloadTag/AMD%20GPIO%20Driver http://drivers.softpedia.com/downloadTag/AMD%20GPIO%20Driver

[9] AS3722 Power Management Unit http://ams.com/eng/Products/Power-Management/Power-Management-Units/AS3722

[10] AT91SAM Community, http://www.at91.com/

[11] ST-Ericsson U300 Device Tree Bindings https://www.kernel.org/doc/Documentation/devicetree/bindings/arm/ste-u300.txt

[12] Lantiq SOCs https://wiki.openwrt.org/doc/hardware/soc/soc.lantiq

[13] Count Lines of Code, http://cloc.sourceforge.net/

[14] Cscope, http://cscope.sourceforge.net/