FT5406DQ9 Touch Driver Design Document

Introduction :

This document describes the FT5406 touchsreen controller which allows kernel drivers to report details for an arbitrary number of contacts.

FT5406DQ9 :

The FT5x06 Series ICs are single-chip capacitive touch panel controller ICs with a built-in 8 bit Micro-controller unit (MCU).They adopt the mutual capacitance approach, which supports true multi-touch capability.

Some features:
* Mutual Capacitive Sensing Techniques.
* True Multi-touch with up to 10 Points of Absolution X and Y Coordinates.
* Immune to RF Interferences .
* Auto-calibration:Insensitive to Capacitance and Environ-mental Variations.
* Supports up to 26 Transmit Lines and 16 Receive Lines .
* Supports up to 8” Touch Screen.

The FT5x06 is comprised of five main functional parts listed below,

1. Touch Panel Interface Circuits **:**
The main function for the AFE and AFE controller is to interface with the touch panel. It scans the panel by sending AC signals to the panel and processes the received signals from the panel. So, it supports both Transmit (TX) and Receive (RX) functions. Key pa-rameters to configure this circuit can be sent via serial interfaces, which will be explained in detail in a later section.

2. 8051-based MCU :
This MCU is 8051 compatible with some enhancements. For instant, larger program and data memories are supported. In addition, a Multiplication-Division unit (MDU) is implemented to speed up the touch detection algorithms. Furthermore, a Flash ROM is implemented to store program s and some key parameters.
Complex signal processing algorithms are implemented with firmware running on this MCU to process further the received signals in order to detect the touches reliably. Comm unication protocol software is also implemented on this MCU to exchange data and control information with the host processor.

3. External Interface:
* I2C/SPI: an interface for data exchange with host .
* INT: an interrupt signal to inform the host processor that touch data is ready for read .
* WAKE: an interrupt signal for the host to change FT5x06 from Hibernate to Active mode .
* /RST: an external low signal reset the chip .

4. A watch dog timer is implemented to ensure the robustness of the chip.

5. A voltage regulator to generate 1.8V for digital circuits from the input VDD3 supply.

Operation Modes :

FT5x06 operates in the following three modes:

Active Mode :
When in this mode, FT5x06 actively scans the panel. The default scan rate is 60 frames per second. The host processor can configure FT5x06 to speed up or to slow down.

Monitor Mode :
When in this mode, FT5x06 scans the panel at a reduced speed. The default scan rate is 25 frames per second and the host processor can increase or decrease this rate. When in this mode, most algorithms are stopped. A simpler algorithm is being executed to de-termine if there is a touch or not. When a touch is detected, FT5x06 shall enter the Active mode immediately to acquire the touch information quickly. During this mode, the serial port is closed and no data shall be transferred with the host processor.

Hibernate Mode :
In this mode, the chip is set in a power down mode. It shall only respond to the “WAKE” or “RESET” signal from the host processor. The chip therefore consumes very little current, which help prolong the standby time for the portable devices.

Host Interface :

The bellow figure shows the interface between a host processor and FT5x06. This interface consists of the following three sets of signals:

* Serial Interface.
* Interrupt from FT5x06 to the Host.
* Wake-up Signal from the Host to FT5x06.

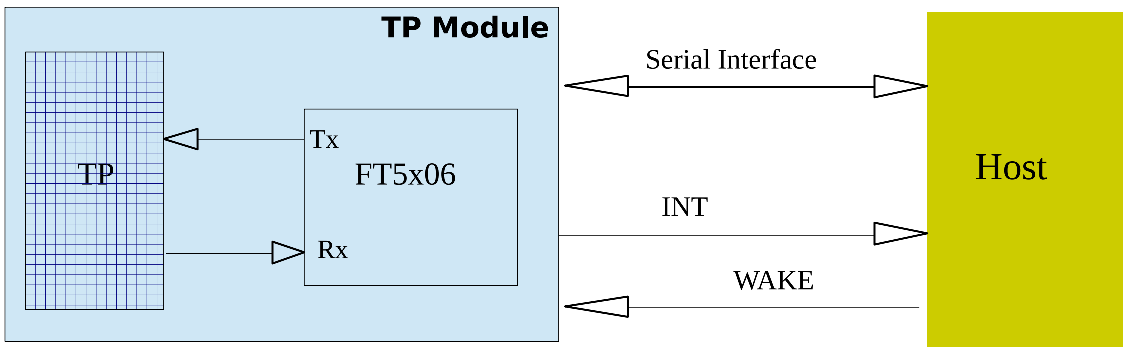


Figure : Host Interface Diagram

The serial interfaces of FT5x06 is I2C or SPI. The details of this interface are described in detail in Section 2.5. The interrupt signal (/INT) is used for FT5x06 to inform the host that data are ready for the host to receive. The /WAKE signal is used for the host to wake up FT5x06 from the Hibernate mode. After exiting the Hi
bernate mode, FT5x06 shall enter the Active mode.

Serial Interface :
FT5x06 supports the I2C or SPI interfaces, which can be used by a host processor or other devices.

Control Signal	Pin Number
INT	38
WAKE	37

Product Name	Package Type	# TX Pins	# RX Pins
FT5206GE1	QFN-40L	15	10
FT5306DE4	QFN-48L	20	12
FT5406DQ9	QFN-56L	26	16
FT5406EE8	QFN-68L	28	16

Raspberry pi 7" Multitouch Display :

In this build, we're using the official 7” multitouch display from the Raspberry Pi foundation. It features a beautiful IPS display and includes the drivers to work on a Raspberry Pi 3.

Raspberry pi 3 :

The Raspberry Pi display is an LCD display which connects to the Raspberry Pi through the DSI connector. In some situations, it allows for the use of both the HDMI and LCD displays at the same time (this requires software support).

The DSI connector on the Model A/B boards does not have the I2C connections required to talk to the touchscreen controller and DSI controller.

Raspberry Pi LCD DSI Display Connector

The Raspberry Pi connector S2 is a *display serial interface* (DSI) for connecting a *liquid crystal display* (LCD) panel using a 15-pin ribbon cable. The *mobile industry processor interface* (MIPI) inside the Broadcom BCM2835 IC feeds graphics data directly to the display panel through this connector. This article looks at the connector pinout, and some of the display panels compatible with the port.

If you look at the BCM2835 Block Diagram, you will see that it has internal circuitry to drive a colour LCD display panel. The S2 connector provides a fast high-resolution display interface dedicated for the purposes of sending video data directly from the GPU to a compatible display.

DSI(Display serieal Interface) :

DSI (Display serial interface) is a high-speed serial interface based on a number of (1GBits) data lanes.

DSI Connector Pinout :

Socket S2 Pin	Function
1	Ground
2	Data Lane 1 N
3	Data Lane 1 P
4	Ground
5	Clock N
6	Clock P
7	Ground
8	Data Lane 0 N
9	Data Lane 0 P
10	Ground
11	
12	
13	Ground
14	+3.3 V
15	+3.3 V

Input subsysteam :

The input subsystem is the part of the Linux kernel that manages the various input devices that a user uses to interact with the kernel, command line and graphical user interface.
Examples:-

1.Keyboards
2.Mice
3.Joysticks
4.Touch Screens

This subsystem is part of the kernel because these devices usually are accessed through special hardware interfaces (such as serial ports,PS/2, SPI, I2C and USB), which are protected and managed by the kernel itself.

The event interface allows you to query which features and capabilities are available for a particular device. The types of events supported by the event interface are:

A wide range of individual codes can be found within each event type. For example, the EV_ABS specifies codes to distinguishes between ABS_X, ABS_Y and ABS_Z axes and ABS_PRESSURE and many more. Similarly, the EV_KEY feature type includes literally hundreds of different key and button codes.

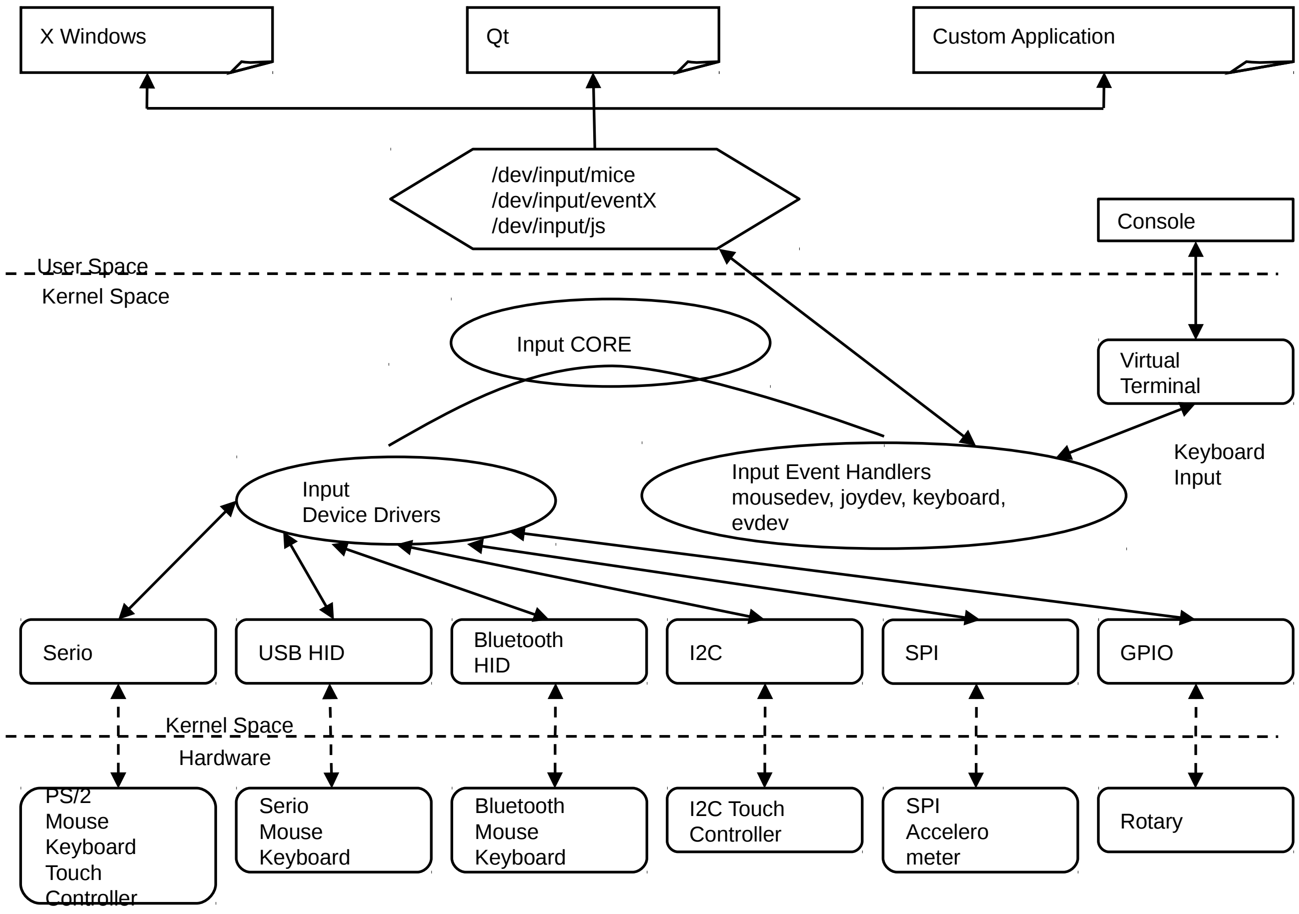
The three major elements of the input subsystem are:

1.Input Core
2.Event Handlers
3.Input Device Drivers

Note that while the normal path is from low-level hardware to drivers, drivers to input core, input core to handler and handler to user space, there usually is a return path as well.

This return path allows for such things as setting the LEDs on a keyboard and providing motion commands to force feedback joysticks.

The code work flow of input devices:-



Multi-Point versus Multi-Touch :

single-point:
single-point interaction. One x/y coordinate, button/key state.

single-touch:
not a single point, but a single area. touching with a thumb is different than touching with a stylus. Touch screens that only do single point interaction are technically no different than a mouse.

multi-point:

multiple points of interaction.

multi-touch:
multiple separate input contact areas.

Event Semantics :

The word "contact" is used to describe a tool which is in direct contact with the surface. A finger or a pen all classify as contacts.

ABS_MT_TOUCH_MAJOR:
The length of the major axis of the contact. The length should be given in surface units. If the surface has an X times Y resolution, the largest possible value of ABS_MT_TOUCH_MAJOR is sqrt(X^2 + Y^2), the diagonal.

ABS_MT_TOUCH_MINOR:
The length, in surface units, of the minor axis of the contact. If the contact is circular, this event can be omitted.

ABS_MT_WIDTH_MAJOR:
The length, in surface units, of the major axis of the approaching tool. This should be understood as the size of the tool itself. The orientation of the contact and the approaching tool are assumed to be the same.

ABS_MT_WIDTH_MINOR:
The length, in surface units, of the minor axis of the approaching tool. Omit if circular.

ABS_MT_PRESSURE:
The pressure, in arbitrary units, on the contact area. May be used instead of TOUCH and WIDTH for pressure-based devices or any device with a spatial signal intensity distribution.

ABS_MT_POSITION_X:
The surface X coordinate of the center of the touching ellipse.

ABS_MT_POSITION_Y:
The surface Y coordinate of the center of the touching ellipse.

ABS_MT_TOOL_TYPE:
The type of approaching tool.
The protocol currently supports:
MT_TOOL_FINGER
MT_TOOL_PEN

A lot of kernel drivers cannot distinguish between different tool types, such as a finger or a pen. In such cases, the event should be omitted.

ABS_MT_BLOB_ID:
The BLOB_ID groups several packets together into one arbitrarily shaped contact. This is a low-level anonymous grouping, and should not be confused with the high-level trackingID.
Most kernel drivers will not have blob capability, and can safely omit the event.

ABS_MT_TRACKING_ID
The TRACKING_ID identifies an initiated contact throughout its life cycle. There are currently only a few devices that support it, so this event should normally be omitted.

Event Computation :

The diversity of different hardware unavoidably leads to some devices fitting better to the MT protocol than others. To simplify and unify the mapping, this section gives recipes for how to compute certain events.

For devices reporting contacts as rectangular shapes, signed orientation cannot be obtained. Assuming X and Y are the lengths of the sides of the touching rectangle, here is a simple formula that retains the most information possible:

```
  ABS_MT_TOUCH_MAJOR := max(X, Y)
  ABS_MT_TOUCH_MINOR := min(X, Y)
  ABS_MT_ORIENTATION := bool(X > Y)
```

The range of ABS_MT_ORIENTATION should be set to [0, 1], to indicate that the device can distinguish between a finger along the Y axis (0) and a finger along the X axis (1).

Finger Tracking :

The kernel driver should generate an arbitrary enumeration of the set of anonymous contacts currently on the surface. The order in which the packets appear in the event stream is not important.

The process of finger tracking, i.e., to assign a unique trackingID to each initiated contact on the surface, is left to user space; preferably the multi-touch X driver. In that driver, the trackingID stays the same and unique until the contact vanishes (when the finger leaves the surface). The problem of assigning a set of anonymous fingers to a set of identified fingers is a euclidian bipartite matching problem at each event update, and relies on a sufficiently rapid update rate.

There are a few devices that support trackingID in hardware. User space can make use of these native identifiers to reduce bandwidth and cpu usage.

Calibrations :

Touch screens are finding their way into a variety of embedded products. Most touch-enabled devices will require a calibration routine. Here's a good one.

Driver Implementation:

Raspberry Pi uses a FT5406 attached to the VideoCore's I2C bus. The firmware runs a thread that periodicaly does a register dump to a shared memory window. The interrupt and wake signals seem disconnected.

Overaly, this whole "just have the firmware run all of our device drivers" is something I'm not a fan of. I understand that we need to compromise on a few things (thermal appears to be one), but a generic i2c driver for a touchscreen doesn't seem like such a case.

My plan had been to write a native driver once I got DSI going. We should be able to use pinctrl to steal the pins to an i2c adapter that we own. Hopefully someone from the Foundation could clarify which i2c bus we can definitely own -- I think i've been seeing issues with the firmware talking to the busses behind my back

FT5406DQ9 Touch driver implementation steps from Patch :

* Raspberry Pi 7" Touch Screen Display :

Raspberry Pi uses a FT5406 attached to the VideoCore's I2C bus. The firmware
runs a thread that periodicaly does a register dump to a shared memory window.
The interrupt and wake signals seem disconnected.

* Required properties:
 - compatible: "raspberrypi,raspberrypi-ft5406"
 - reg: address range of the shared memory with register dump

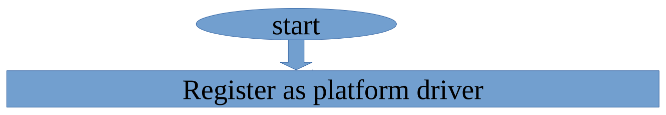
Example:

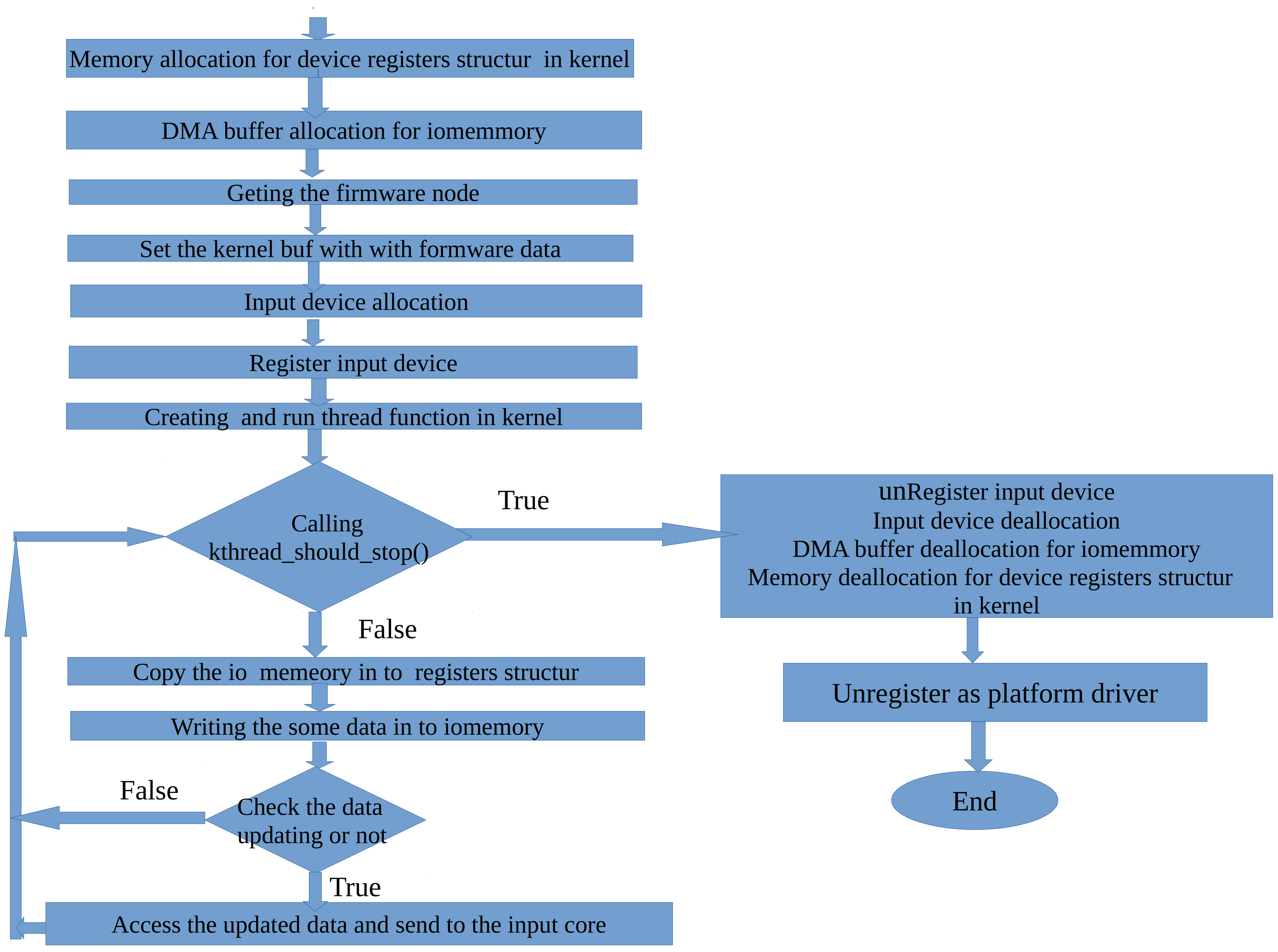
```
	touchscreen: rpi-ft5406@3e40ea40 {
		compatible = "raspberrypi,raspberrypi-ft5406";
		reg = <0x3e40ea40 63>;
	};
```

* Driver for ft5406 touchscreen controller accessible via Raspberry Pi firmware.

* For this driver implement the thread function. that function polls the memory based register copy of the ft5406 registers using the number of points register to know whether the copy has been updated (we write 99 to the memory copy, the GPU will write between 0 - 10 points).

Touch Screen driver Flow chart Diagram:




Memory allocation for device registers structur in kernel
DMA buffer allocation for iomemmory
Geting the firmware node
Set the kernel buf with with formware data
Input device allocation
Register input device
Creating and run thread function in kernel
Calling kthread_should_stop()
True
unRegister input device
Input device deallocation
DMA buffer deallocation for iomemmory
Memory deallocation for device registers structur in kernel
Unregister as platform driver
End
False
Copy the io memeory in to registers structur
Writing the some data in to iomemory
False
Check the data updating or not
True
Access the updated data and send to the input core

References :
For FT5x06 data sheet: https://www.buydisplay.com/download/ic/FT5206.pdf

For Raspberry PI 3 data sheet:
http://docseurope.electrocomponents.com/webdocs/14ba/0900766b814ba5fd.pdf

For GPU with touch screen on Raspberry PI 3 :
http://optisimon.com/raspberrypi/touch/ft5406/2016/07/13/raspberry-pi-7-inch-touchscreen-hacking/