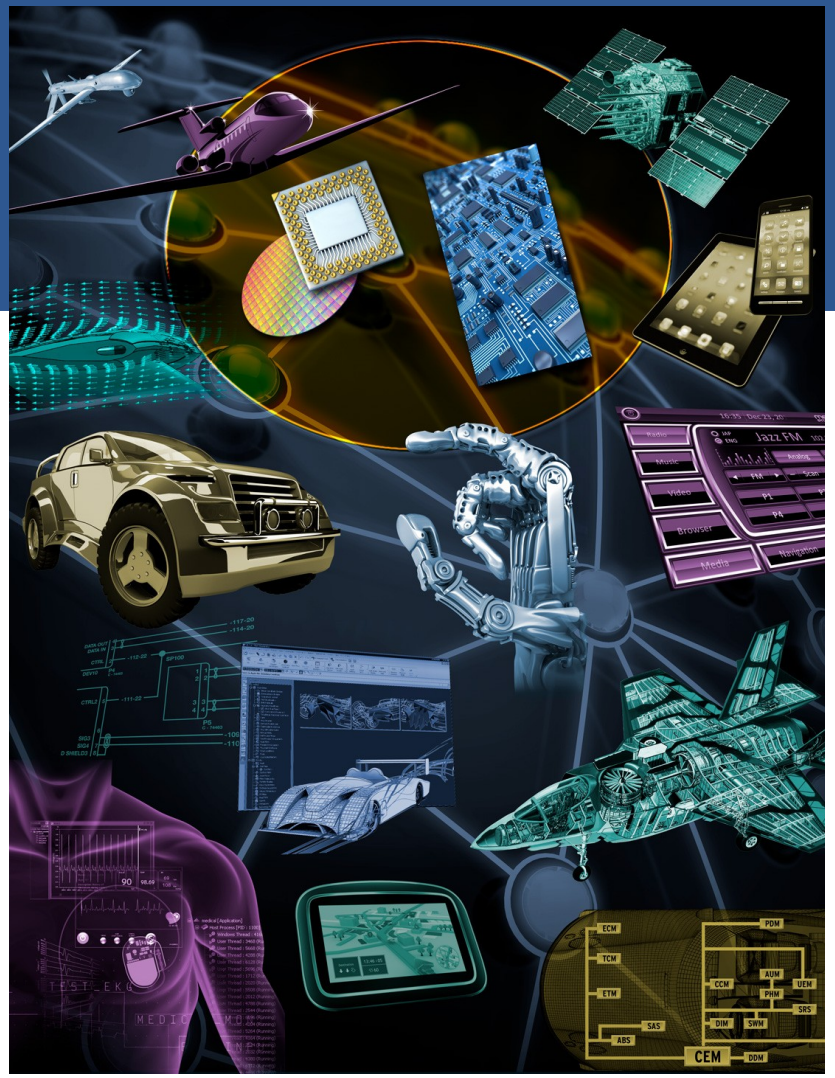


Bridging of Control Interfaces over Multimedia Serial Links

Vladimir Zapolskiy

Open Source Senior Software Engineer

Mentor Automotive

June 22, 2018

Mentor®
A Siemens Business

Introduction

- Embedded Linux developer since 2006
- Open Source Software contributor since 2010
- Contributed to Linux, U-Boot, OpenEmbedded etc.

  — saved Renesas SH2/SH3/SH4 from removal in U-Boot
- Most of contributions to open source is done as a hobbyist
- Since 2014 I work as a developer and maintainer of Linux kernel forks for Mentor Automotive customers

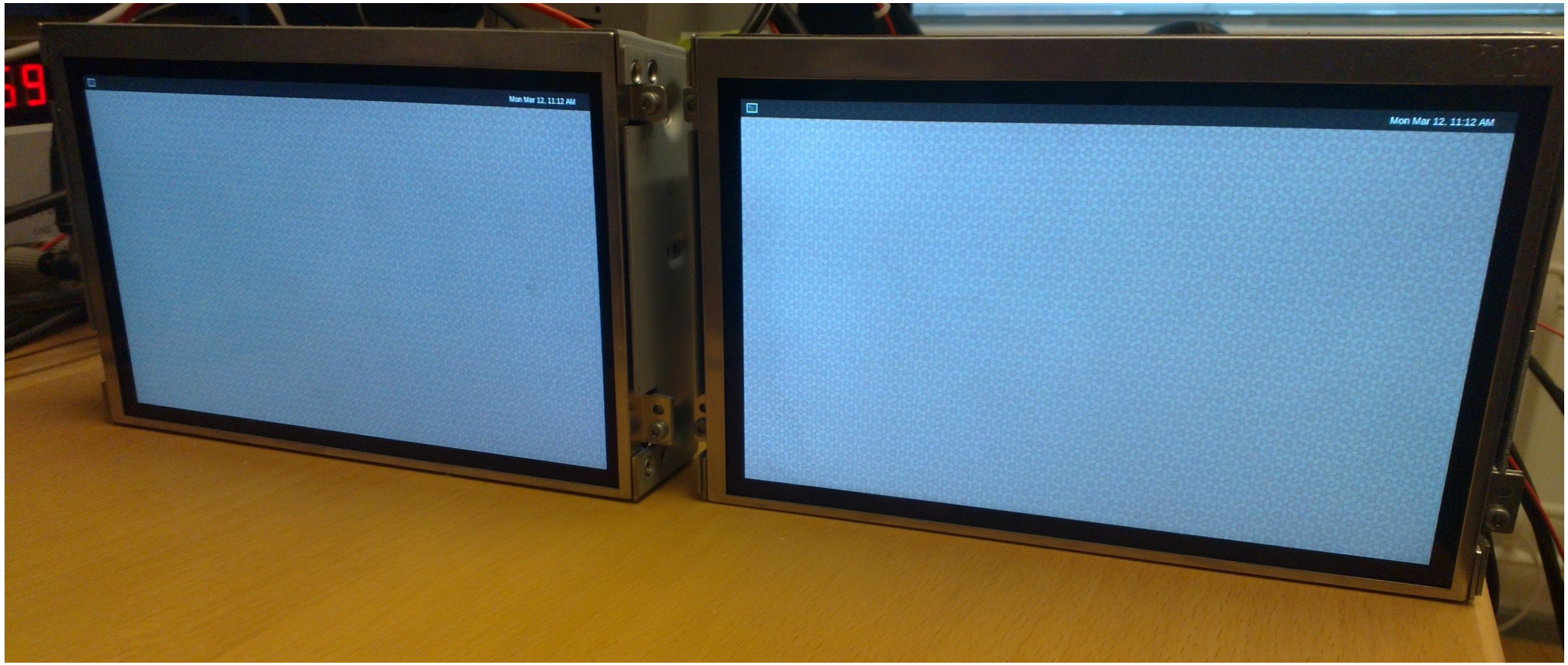
Summary

- Hardware, FPD-Link III and TI DS90Ux9xx IC overview
- Challenges to support TI DS90Ux9xx in Linux
- DS90Ux9xx Linux device drivers
- DS90Ux9xx device tree bindings
- Future work
- Questions

Board with serializers connected to a SoC

XSe/Mentor AXSB reference board for Automotive

Display Modules

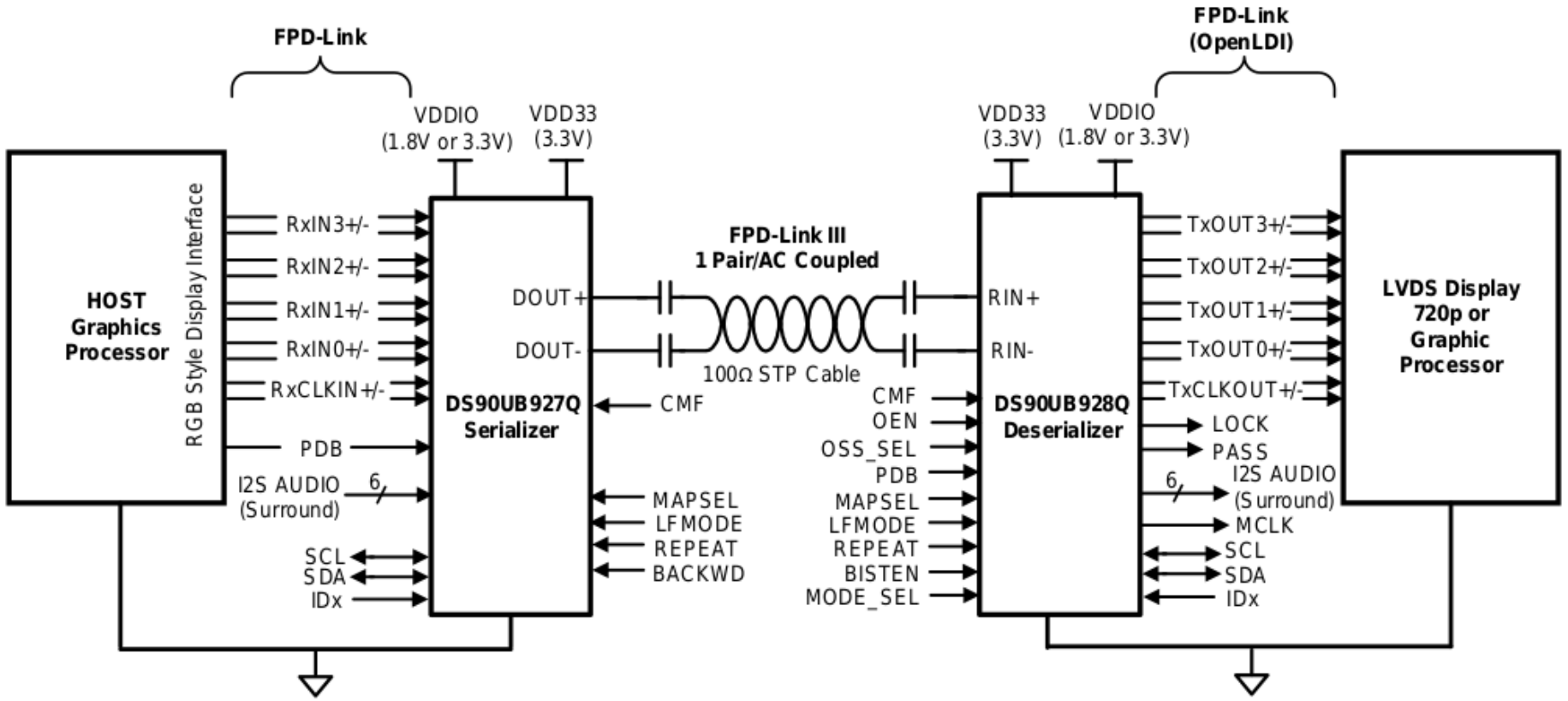


LVDS display modules with Atmel touchscreen and TI DS90UB928 deserializers

Challenges to be solved in software

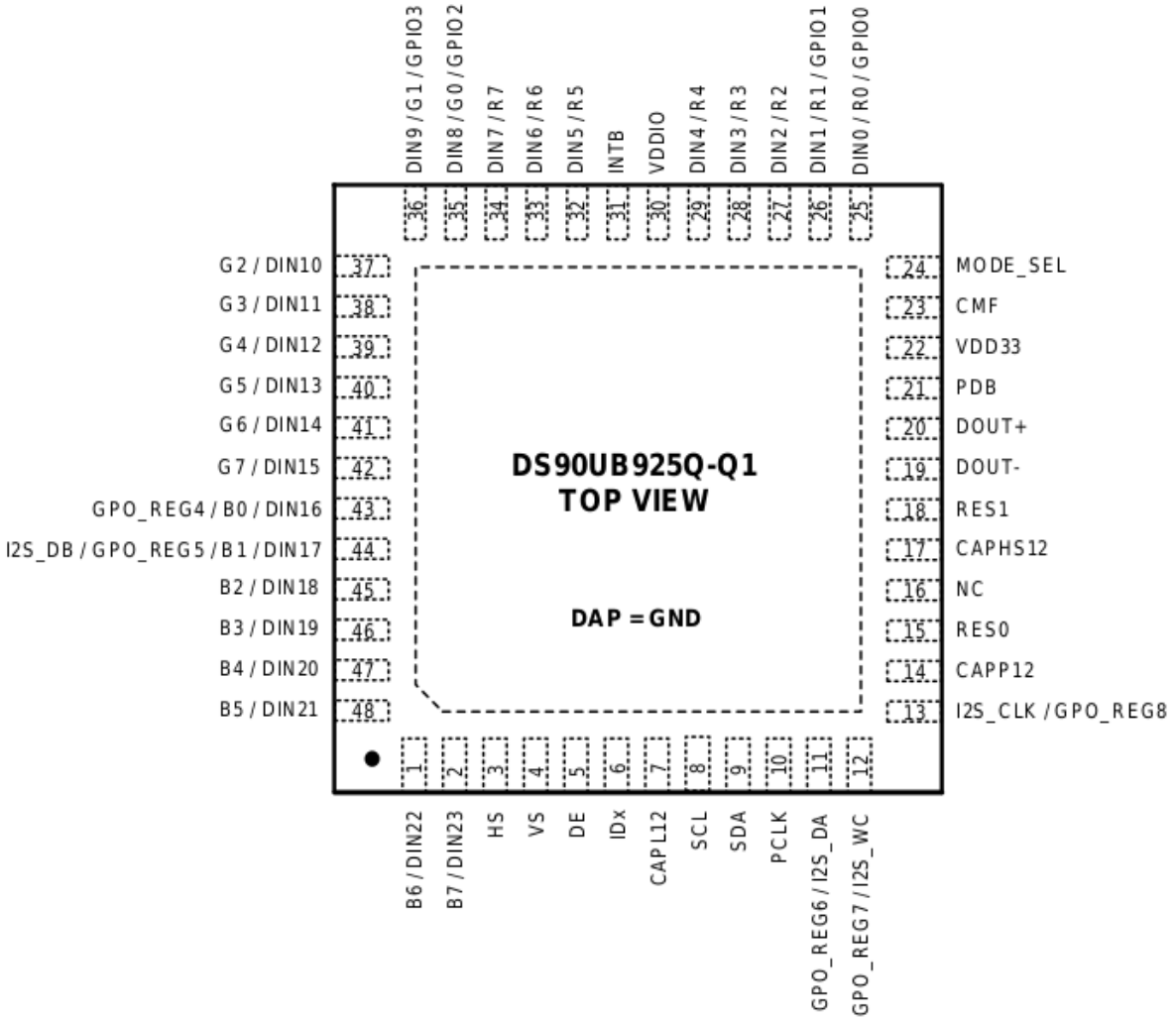
- Link hotplug
- Issues related to display panel modules:
  - Panel hardware description in device tree
  - Description of non-identical panels by same device node
  - Connection of identical panels on one I2C bus
- Bridging of video/audio data and control signals:
  - Interrupt line bridging
  - Bi-directional bridging of GPIO signals
  - Bi-directional bridging of I2C data

TI FPD-Link III



www.ti.com/lit/ds/symlink/ds90ub928q-q1.pdf

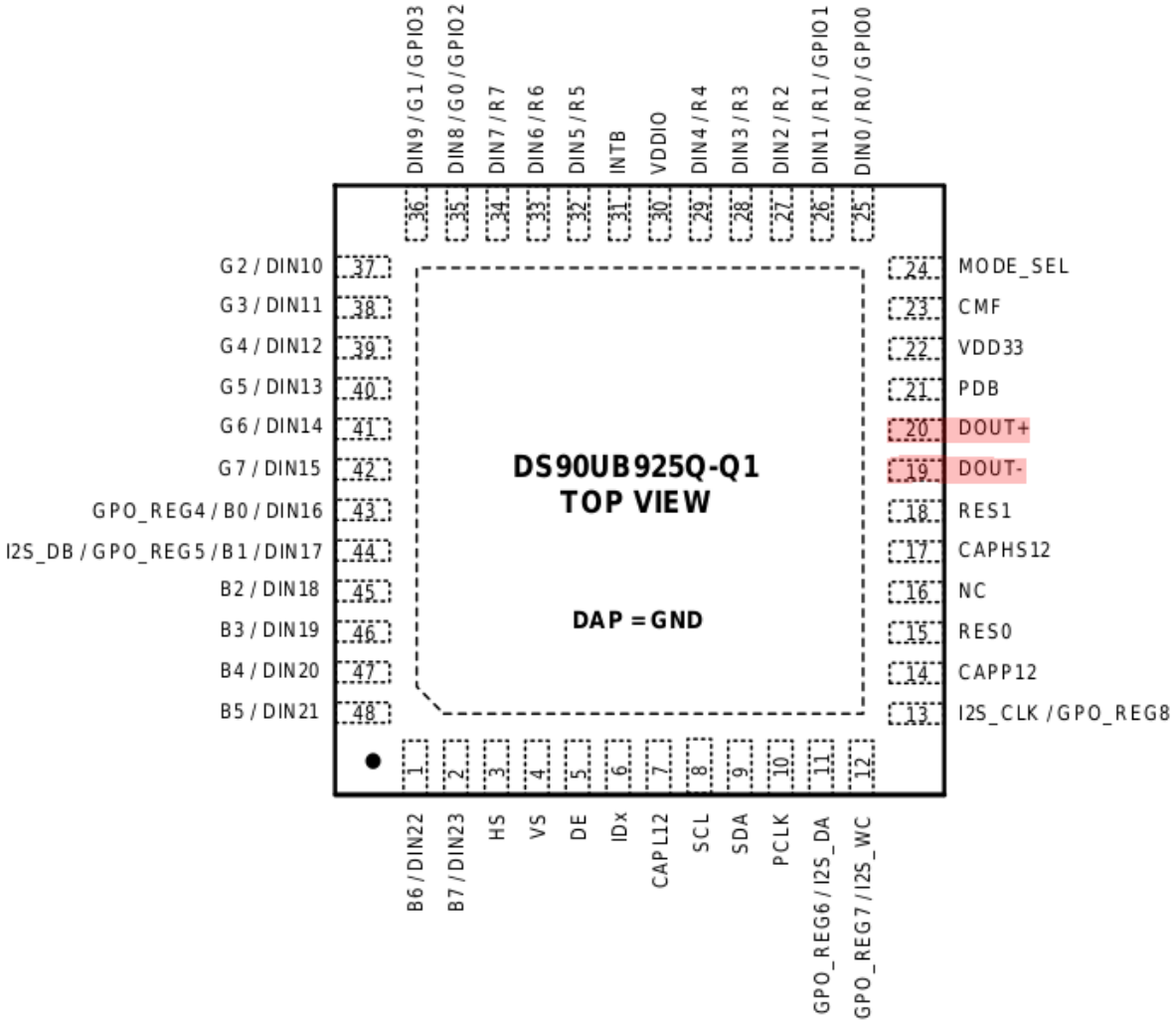
TI DS90Ux9xx pins



- FPD-Link III
- IC controls
- I2C device
- Parallel RGB666 / RGB888
- I2S
- GPIO
- Interrupt

www.ti.com/lit/ds/symlink/ds90ub925q-q1.pdf

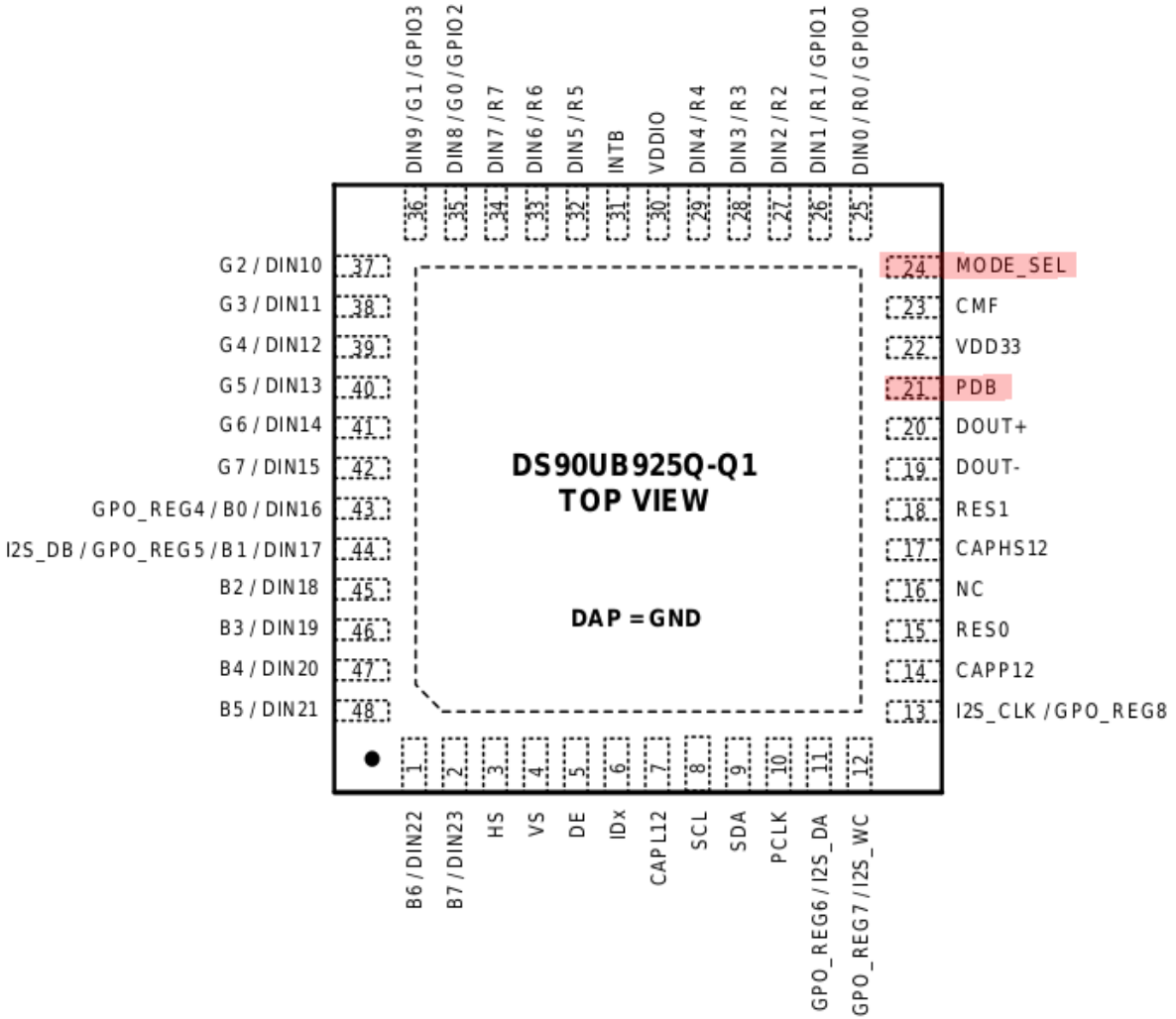
TI DS90Ux9xx pins



- **FPD-Link III**
- IC controls
- I2C device
- Parallel RGB666 / RGB888
- I2S
- GPIO
- Interrupt

www.ti.com/lit/ds/symlink/ds90ub925q-q1.pdf

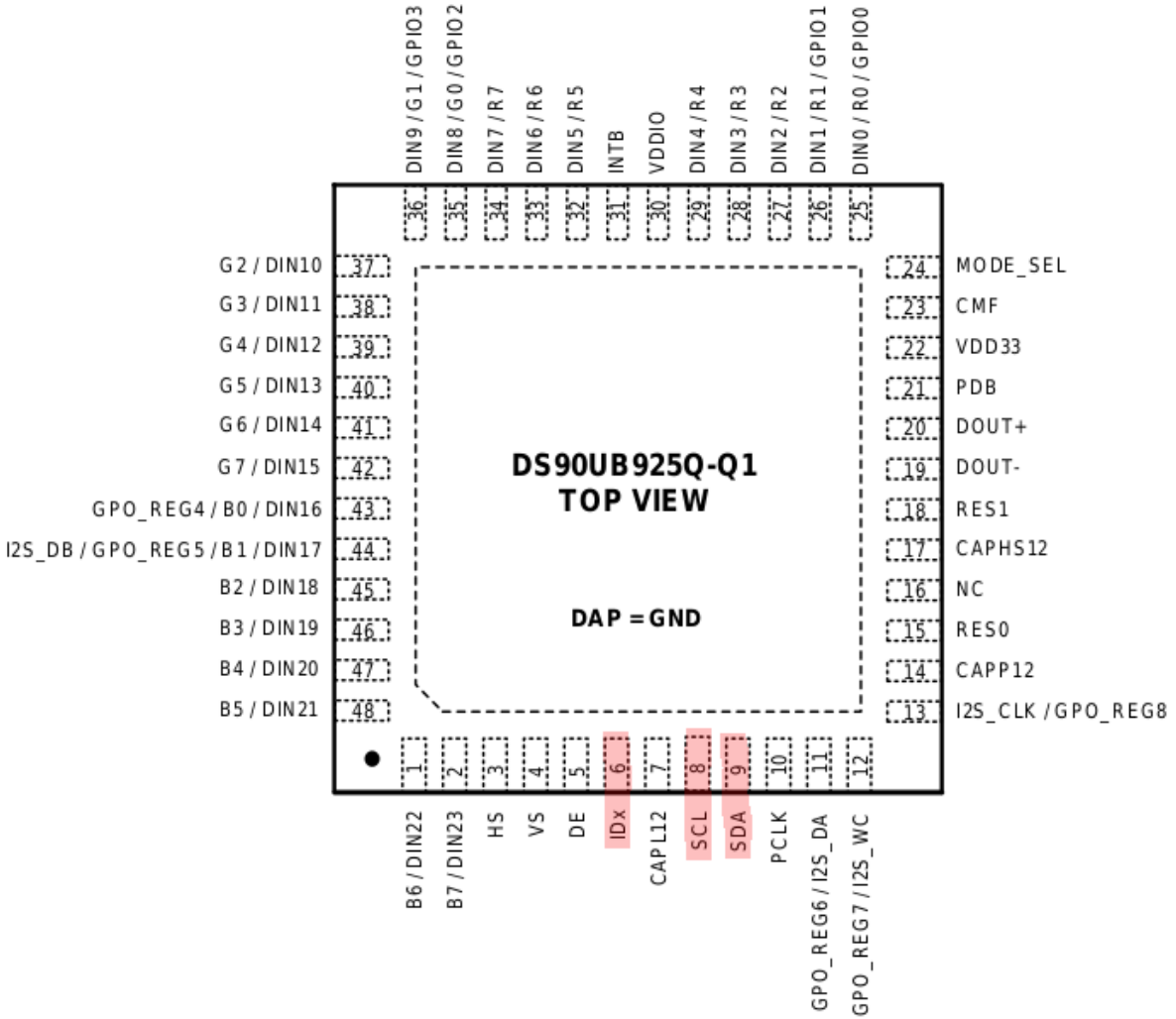
TI DS90Ux9xx pins



- FPD-Link III
- **IC controls**
- I2C device
- Parallel RGB666 / RGB888
- I2S
- GPIO
- Interrupt

www.ti.com/lit/ds/symlink/ds90ub925q-q1.pdf

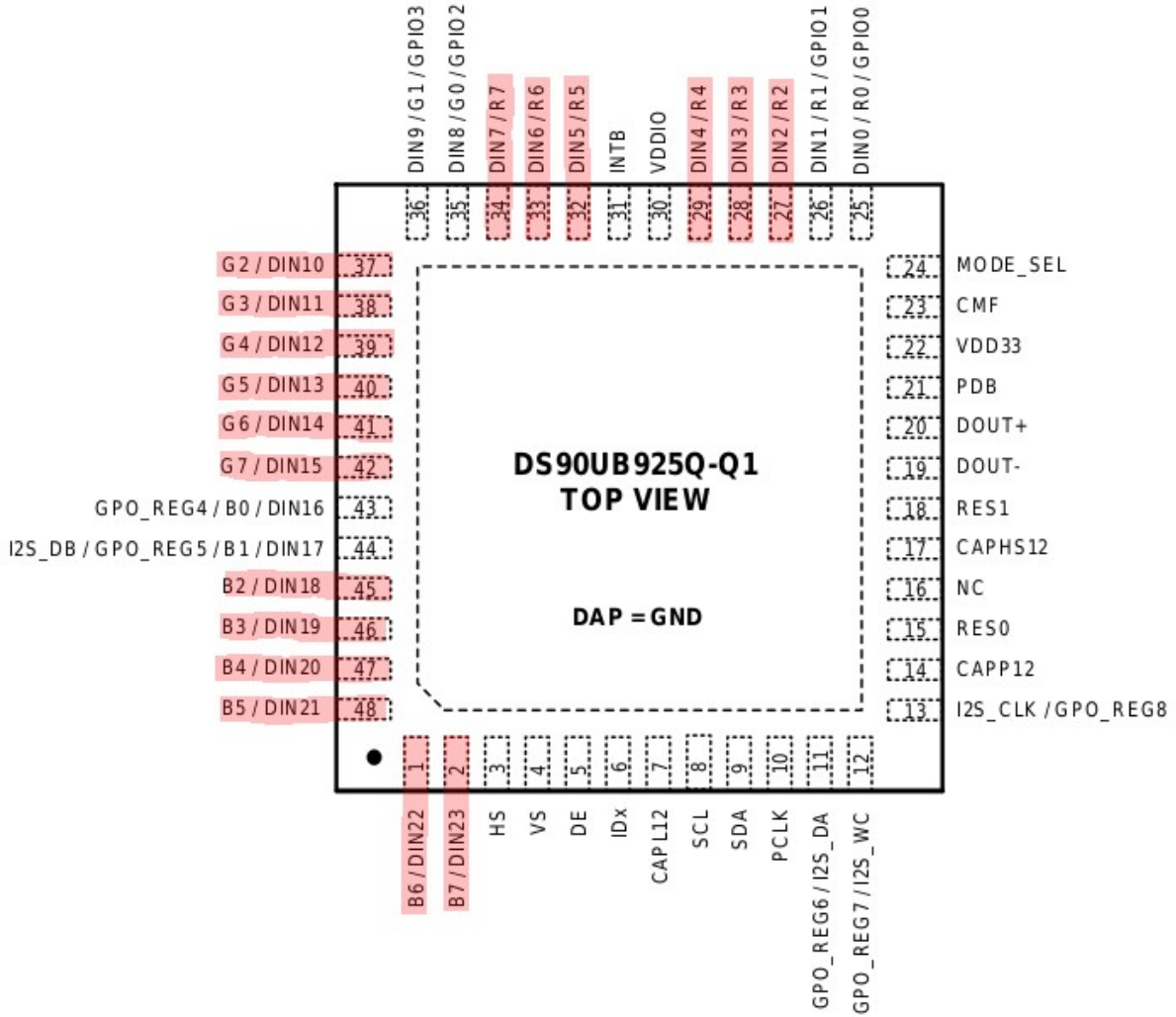
TI DS90Ux9xx pins



- FPD-Link III
- IC controls
- **I2C device**
- Parallel RGB666 / RGB888
- I2S
- GPIO
- Interrupt

www.ti.com/lit/ds/symlink/ds90ub925q-q1.pdf

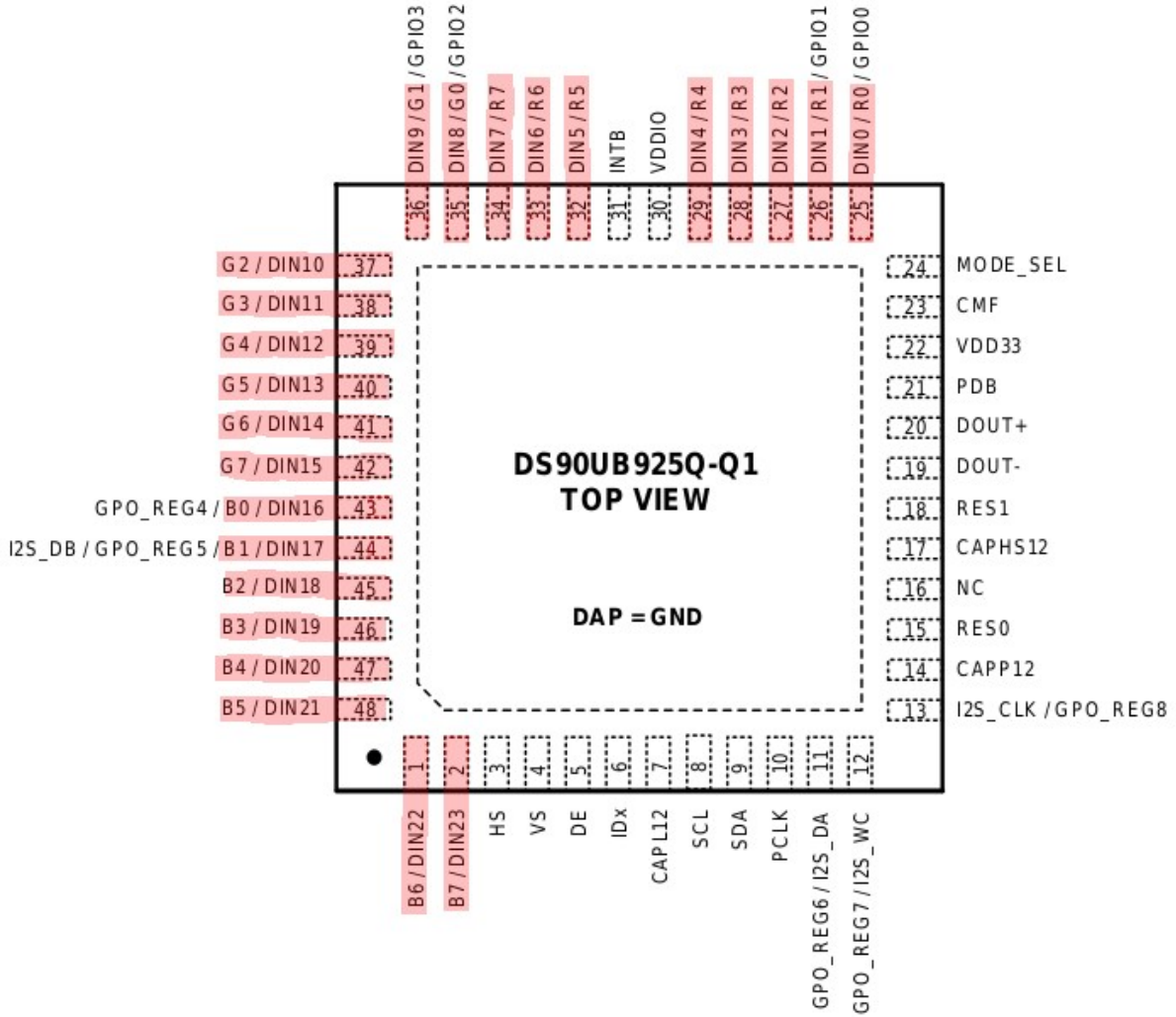
TI DS90Ux9xx pins



- FPD-Link III
- IC controls
- I2C device
- Parallel **RGB666** / RGB888
- I2S
- GPIO
- Interrupt

www.ti.com/lit/ds/symlink/ds90ub925q-q1.pdf

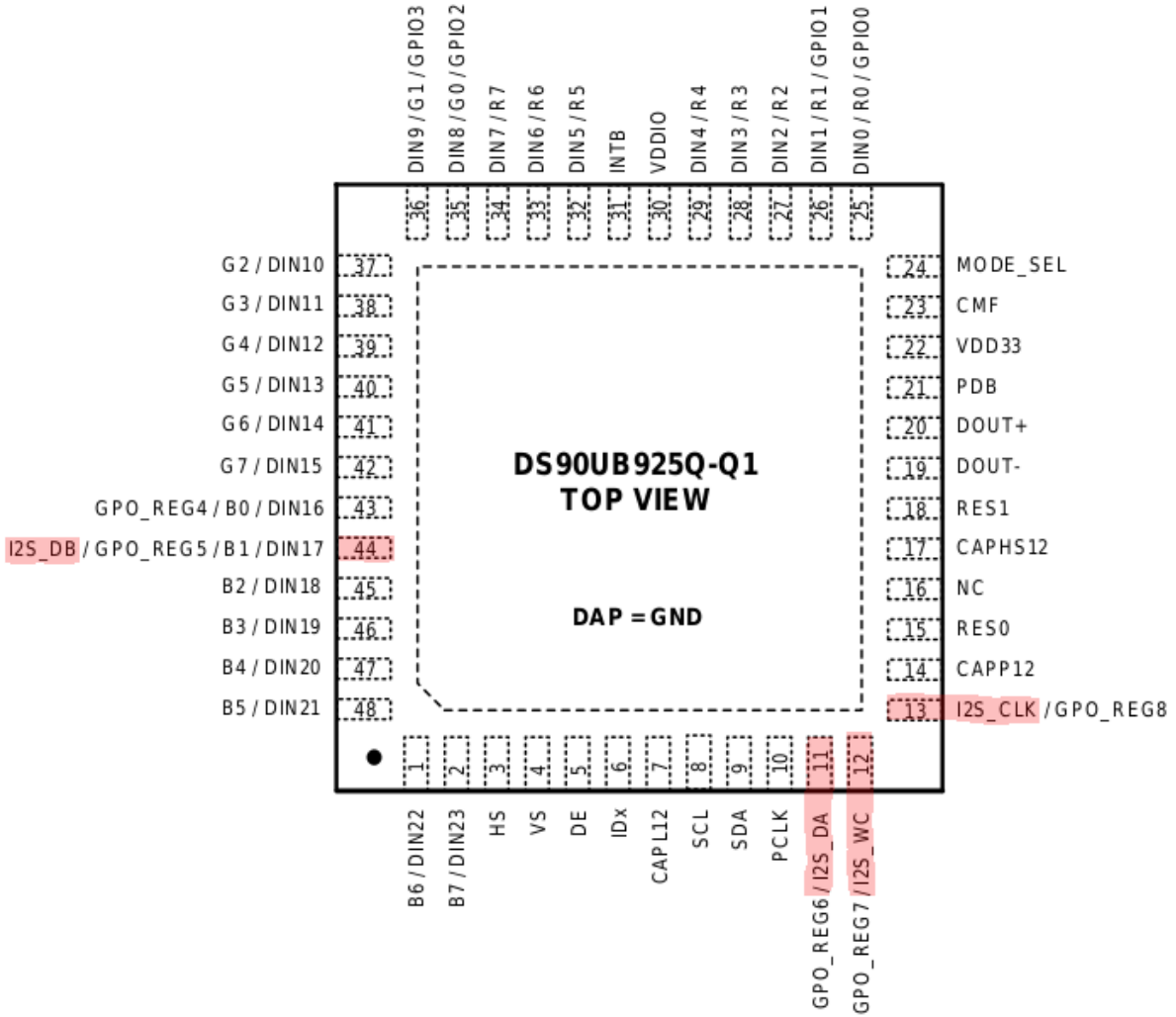
TI DS90Ux9xx pins



- FPD-Link III
- IC controls
- I2C device
- Parallel RGB666 / **RGB888**
- I2S
- GPIO
- Interrupt

www.ti.com/lit/ds/symlink/ds90ub925q-q1.pdf

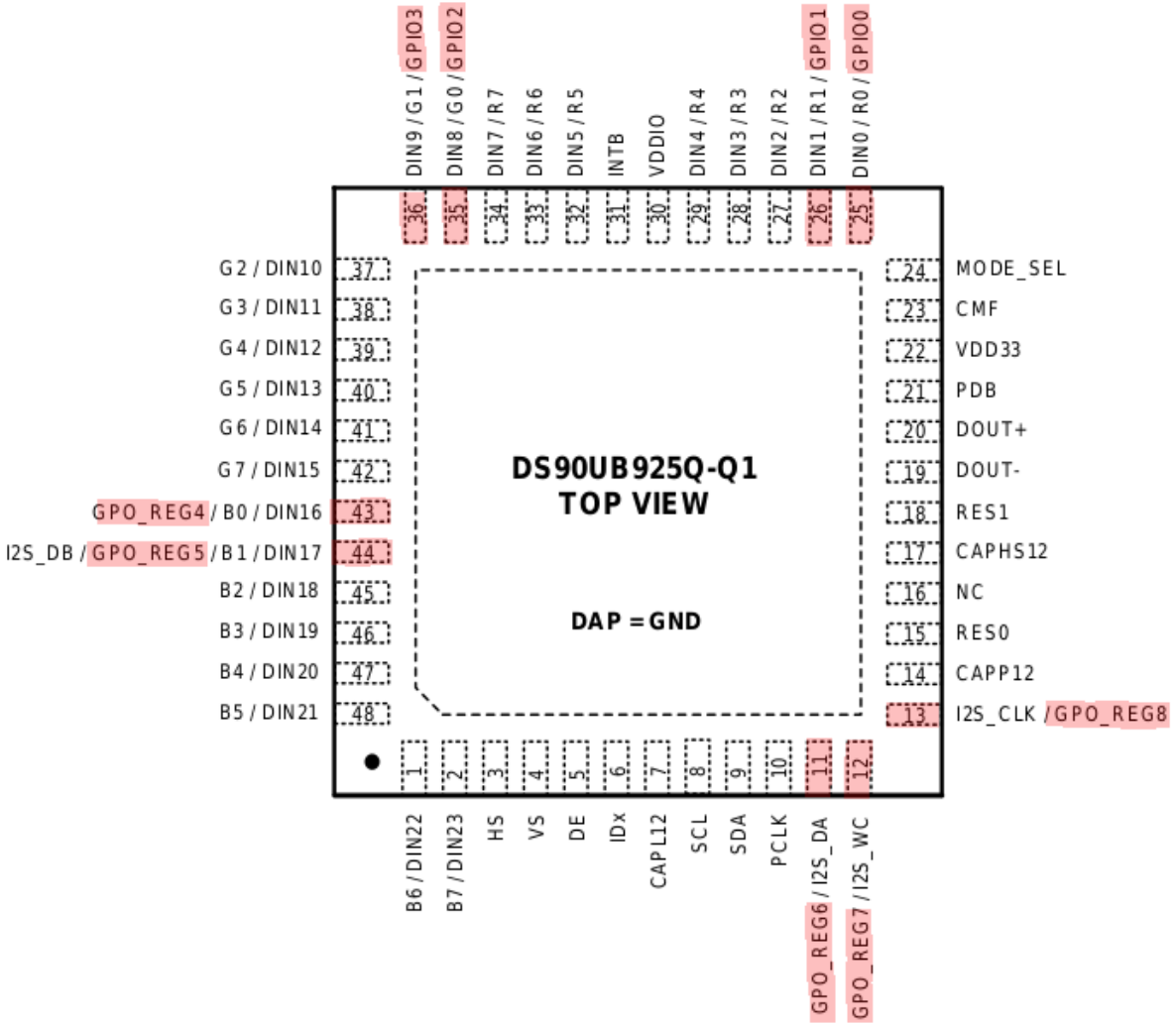
TI DS90Ux9xx pins



- FPD-Link III
- IC controls
- I2C device
- Parallel RGB666 / RGB888
- **I2S**
- GPIO
- Interrupt

www.ti.com/lit/ds/symlink/ds90ub925q-q1.pdf

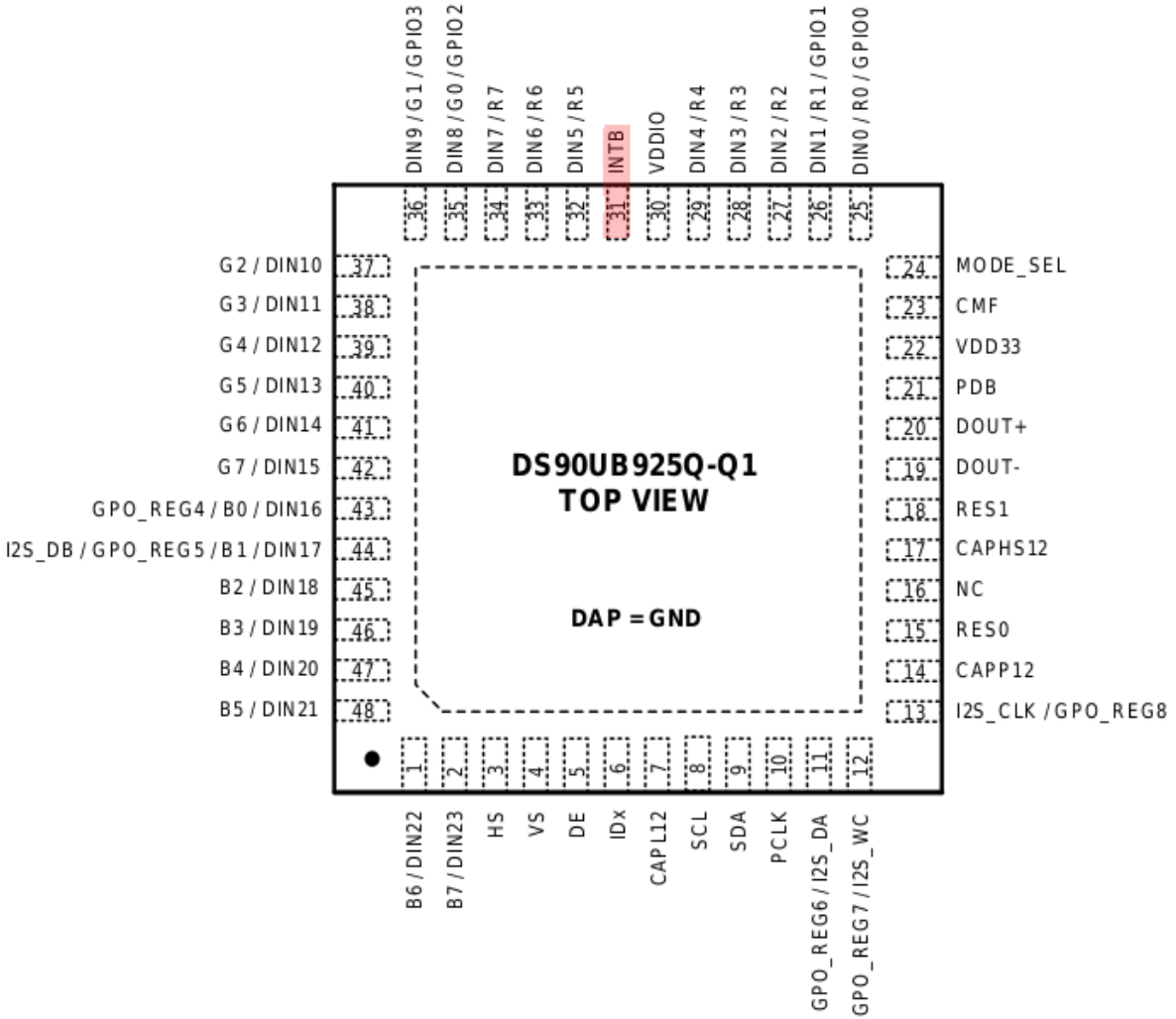
TI DS90Ux9xx pins



- FPD-Link III
- IC controls
- I2C device
- Parallel RGB666 / RGB888
- I2S
- **GPIO**
- Interrupt

www.ti.com/lit/ds/symlink/ds90ub925q-q1.pdf

TI DS90Ux9xx pins



- FPD-Link III
- IC controls
- I2C device
- Parallel RGB666 / RGB888
- I2S
- GPIO
- **Interrupt**

www.ti.com/lit/ds/symlink/ds90ub925q-q1.pdf

Multi-function device, why not?

- One IC and one control interface (I2C, SPI, UART), multiple functions, where one type of a bridge is one IC function
- Deserializer and serializer ICs from TI DS90Ux9xx family are similar in sense that most of them can be described as a particular set of sub-devices (bridges), and sub-devices are practically equal on various ICs
- Video bridge IC function is mapped well to the current

  Linux DRM video bridge framework (host data to serializer)

  - ... however an ideal symmetry of deserializers and serializers connected to a host is behind the horizon

Multi-function device

```
static const struct ds90ux9xx_device_property ds90ux9xx_devices[] =  {
        /*
         * List of TI DS90Ux9xx properties:
         *      serializer, HDCP, BW/LF modes, SSCG, pixel clock edge, mapsel,
         *      RGB DE Gate
         */
        DS90UX9XX_DEVICE(UB925, 1, 0, 1, 0, 1, 0, 1),
        DS90UX9XX_DEVICE(UH925, 1, 1, 1, 0, 1, 0, 1),
        DS90UX9XX_DEVICE(UB927, 1, 0, 1, 0, 0, 1, 1),
        DS90UX9XX_DEVICE(UH927, 1, 1, 1, 0, 0, 1, 1),
        DS90UX9XX_DEVICE(UB926, 0, 0, 1, 1, 1, 0, 0),
        DS90UX9XX_DEVICE(UH926, 0, 1, 1, 1, 1, 0, 0),
        DS90UX9XX_DEVICE(UB928, 0, 0, 1, 0, 0, 1, 0),
        DS90UX9XX_DEVICE(UH928, 0, 1, 1, 0, 0, 1, 0),
        DS90UX9XX_DEVICE(UB940, 0, 0, 0, 0, 0, 0, 1),
        DS90UX9XX_DEVICE(UH940, 0, 1, 0, 0, 0, 0, 1),
};
```

Multi-function device, probe sub-devices

```
ds90ux9xx->dev = &client->dev;
ds90ux9xx->regmap = devm_regmap_init_i2c(client,
                                         &ds90ux9xx_regmap_config);
if (IS_ERR(ds90ux9xx->regmap))
        return PTR_ERR(ds90ux9xx->regmap);

/*  ...  */

i2c_set_clientdata(client, ds90ux9xx);

ret = ds90ux9xx_config_properties(ds90ux9xx);
if (ret)
        return ret;

ret = sysfs_create_groups(&ds90ux9xx->dev->kobj, ds90ux9xx_attr_groups);
if (ret)
        return ret;

ret = devm_of_platform_populate(ds90ux9xx->dev);
if (ret)
        sysfs_remove_groups(&ds90ux9xx->dev->kobj,
                            ds90ux9xx_attr_groups);
```

Multi-function device, interface to subdevices

```
enum ds90ux9xx_device_id {
        /* Supported serializers */
        TI_DS90UB925,
        TI_DS90UH925,
        TI_DS90UB927,
        TI_DS90UH927,

        /* Supported deserializers */
        TI_DS90UB926,
        TI_DS90UH926,
        TI_DS90UB928,
        TI_DS90UH928,
        TI_DS90UB940,
        TI_DS90UH940,
};

bool ds90ux9xx_is_serializer(struct device *dev);
int ds90ux9xx_update_bits_indirect(struct device *dev, u8 reg, u8 mask, u8 val);
int ds90ux9xx_write_indirect(struct device *dev, u8 reg, u8 val);
enum ds90ux9xx_device_id ds90ux9xx_get_ic_type(struct device *dev);
```

Multi-function device, device tree bindings

```
/* Application processor side */

&i2c2 {
        status = "okay";
        clock-frequency = <400000>;

        ds90ux9xx_i2c2: serializer@c {
                compatible = "ti,ds90ux9xx", "simple-mfd";
                reg = <0xc>;
                power-gpios = <&gpio5 2 1>;
                ti,backward-compatible-mode = <0>;
                ti,low-frequency-mode = <0>;
                /*  MFD subdevices / bridges  */
        };
        /*    */
};
```

```
/* Display module side */

/ {
        /*    */
        external-pcbs {
                display_0: deserializer0 {
                        compatible = "ti,ds90ux9xx", "simple-mfd";
                        /*  MFD subdevices / bridges  */
                };
                /*    */
        };
        /*    */
};
```

Pinctrl, GPIO controller and GPIO bridging

- pinctrl driver is combined with a GPIO controller driver
- GPIO controller has additional line output “gpio-remote”
  - “gpio-remote” output is a pinctrl function

```
static const struct ds90ux9xx_pin ds90ux926_pins[] = {
        DS90UX9XX_GPIO(0, "gpio0", GPIO_SIMPLE(0x1d, 0, 1), REMOTE, PARALLEL),
        DS90UX9XX_GPIO(1, "gpio1", GPIO_SIMPLE(0x1e, 0, 1), REMOTE, PARALLEL),
        DS90UX9XX_GPIO(2, "gpio2", GPIO_SIMPLE(0x1e, 1, 1), REMOTE, PARALLEL),
        DS90UX9XX_GPIO(3, "gpio3", GPIO_SIMPLE(0x1f,  0, 1), REMOTE, PARALLEL),
        DS90UX9XX_GPIO(4, "gpio4", GPIO_SIMPLE(0x1f, 1, 0),  NONE,     PARALLEL),
        DS90UX9XX_GPIO(5, "gpio5", GPIO_SIMPLE(0x20, 0, 0), I2S_2,     PARALLEL),
        DS90UX9XX_GPIO(6, "gpio6", GPIO_SIMPLE(0x20, 1, 0), I2S_1,     NONE),
        DS90UX9XX_GPIO(7, "gpio7", GPIO_SIMPLE(0x21, 0, 0), I2S_1,     NONE),
        DS90UX9XX_GPIO(8, "gpio8", GPIO_SIMPLE(0x21, 1, 0), I2S_1,     NONE),
};
```

Pinctrl and GPIO controller, device tree bindings

pinctrl is combined with a GPIO controller device:

```
ds90ux928_0_pctrl: pin-controller {
	compatible = "ti,ds90ux9xx-pinctrl";
	gpio-controller;
	#gpio-cells = <2>;
	gpio-ranges = <&ds90ux928_0_pctrl 0 0 8>;

	pinctrl-names = "default";
	pinctrl-0 = <&ds90ux928_0_pins>;

	ds90ux928_0_pins: pinmux {
		gpio-remote {
			pins = "gpio0", "gpio1", "gpio2", "gpio3";
			function = "gpio-remote";
		};
	};

	rst {
		gpio-hog;
		gpios = <4 GPIO_ACTIVE_HIGH>;
		output-high;
	};
};
```

```
ds90ux927_0_pctrl: pin-controller {
	compatible = "ti,ds90ux9xx-pinctrl";
	gpio-controller;
	#gpio-cells = <2>;
	gpio-ranges = <&ds90ux927_0_pctrl 0 0 8>;

	pwm-backlight {
		gpio-hog;
		gpios = <0 GPIO_ACTIVE_HIGH>;
		output-low;
	};

	lvds-pwrdwn {
		gpio-hog;
		gpios = <1 GPIO_ACTIVE_HIGH>;
		output-high;
	};

	backlight-on {
		gpio-hog;
		gpios = <2 GPIO_ACTIVE_HIGH>;
		output-high;
	};

	lcd-on {
		gpio-hog;
		gpios = <3 GPIO_ACTIVE_HIGH>;
		output-low;
	};
};
```

GPIO controller, runtime

```
root@myboard:~# cat /sys/kernel/debug/gpio

[snip]

gpiochip9: GPIOs 488-495, parent: platform/21a4000.i2c:serializer@c:pin-controller, ds90ux927, can sleep:
 gpio-488 (                    |pwm-backlight       ) out lo
 gpio-489 (                    |lvds-pwrdwn         ) out hi
 gpio-490 (                    |backlight-on        ) out hi
 gpio-491 (                    |lcd-on              ) out lo

gpiochip8: GPIOs 496-503, parent: platform/external-pcbs:pcb-display-0:pin-controller, ds90ux928, can sleep:
 gpio-500 (                    |rst                 ) out hi

root@myboard:~# cat /sys/kernel/debug/pinctrl/external-pcbs:pcb-display-0:pin-controller-ds90ux928/pinmux-pins
Pinmux settings per pin
Format: pin (name): mux_owner|gpio_owner (strict) hog?
pin 0 (gpio0): device external-pcbs:pcb-display-0:pin-controller function gpio-remote group gpio0
pin 1 (gpio1): device external-pcbs:pcb-display-0:pin-controller function gpio-remote group gpio1
pin 2 (gpio2): device external-pcbs:pcb-display-0:pin-controller function gpio-remote group gpio2
pin 3 (gpio3): device external-pcbs:pcb-display-0:pin-controller function gpio-remote group gpio3
pin 4 (gpio5): GPIO ds90ux928:500
pin 5 (gpio6): UNCLAIMED
pin 6 (gpio7): UNCLAIMED
pin 7 (gpio8): UNCLAIMED
```

Bridging of I2C data

- I2C bridge sub-device has its own device driver
- DS90U**B**9xx serializers/deserializers don’t provide link connection interrupt, therefore a connection status should be polled by reading a link status register of a local IC,
- On connection a remote side serializer/deserializer is added as a new I2C device on the same I2C bus
- I2c-mux is NOT used, IC with I2C bridge function operates as an I2C slave device with multiple I2C addresses, while i2c-mux adds another logical I2C bus, parallel data transfers over two “independent” I2C buses are not serialized and it causes fancy effects (on old kernels)

Bridging of I2C data, connection event loop

Establishing a link connection to a remote side serializer/deserializer is ruled by multiple states:

```
ret = ds90ux9xx_get_lock_status(i2c_bridge, &lock);
if (ret)
        goto sleep;

if (lock) {
        ret = ds90ux9xx_read_active_link(i2c_bridge, &link);
        if (ret)
                goto sleep;
}

linked = &i2c_bridge->linked[i2c_bridge->link];
remote = &linked->remote;

if (remote->i2c && lock && i2c_bridge->link == link) {
        if (linked->fixed_address)
                goto sleep;

<continued>
```

```
<continued>
                ret = ds90ux9xx_get_remote_addr(i2c_bridge, &addr);
                if (ret < 0)
                        goto sleep;

                if (remote->addr == addr)
                        goto sleep;
        }

        if (remote->i2c)
                ds90ux9xx_handle_disconnection(i2c_bridge);

        if (!remote->i2c && lock) {
                ret = ds90ux9xx_handle_connection(i2c_bridge, link);
                if (ret < 0)
                        dev_err(i2c_bridge->dev, "Can't connect\n");
        }
sleep:
        msleep(DS90UX9XX_CONN_TIME_MSEC);
```

Bridging of I2C data, register remote devices

```
static void ds90ux9xx_add_bridged_device(struct ds90ux9xx_i2c *i2c_bridge,
                                         struct ds90ux9xx_i2c_bridged *bridged)
{
        struct i2c_board_info info = {};
        int ret;

        dev_dbg(i2c_bridge->dev, "Add I2C device '%s'\n", bridged->np->name);

        info.addr = bridged->addr;
        info.of_node = bridged->np;

        /* Non-critical, in case of the problem report it and fallback */
        ret = of_modalias_node(bridged->np, info.type, sizeof(info.type));
        if (ret)
                dev_err(i2c_bridge->dev, "Cannot get module alias for '%s'\n",
                        bridged->np->full_name);

        bridged->i2c = i2c_new_device(i2c_bridge->adapter, &info);
        if (!bridged->i2c)
                dev_err(i2c_bridge->dev, "Cannot add new I2C device\n");
}
```

I2C bridge, device tree bindings

```
/* Host side serializer */

&i2c1 {
        status = "okay";
        clock-frequency = <400000>;

        ds90ux9xx_i2c1: serializer@c {
                compatible = "ti,ds90ux9xx", "simple-mfd";
                reg = <0xc>;
                power-gpios = <&gpio5 12 GPIO_ACTIVE_HIGH>;
                /* ti,fpd-iii-links = <&display_0 0 0x3b>; */

                i2c-bridge {
                        compatible = "ti,ds90ux9xx-i2c-bridge";
                        ti,i2c-bridges = <&display_0 0 0x3b>;
                        ti,i2c-bridge-maps = <0 0x4b 0x64>;
                };

                /*  ...  */
        };
```

```
/* Remote side deserializer */

external-pcbs {
        display_0: pcb-display-0 {
                compatible = "ti,ds90ux9xx", "simple-mfd";

                i2c-bridge {
                        compatible = "ti,ds90ux9xx-i2c-bridge";
                        #address-cells = <1>;
                        #size-cells = <0>;

                        touchscreen@4b {
                                compatible = "atmel,maxtouch";
                                reg = <0x4b>;
                                interrupt-parent = <&intc0>;
                                interrupts = <0>;
                                atmel,mtu = <200>;
                        };
                };
                /*  ...  */
```

I2C bridge, runtime

Link is connected:

```
root@myboard:~# i2cdetect -r -y 0
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- UU -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- UU -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- UU -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Link is disconnected:

```
root@myboard:~# i2cdetect -r -y 0
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- UU -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

- 0x0c is a serializer connected to an application processor
- 0x3b is a remote side deserializer
- 0x64 is a touchscreen controller on a remote PCB

Interrupt line bridging

regmap_add_irq_chip() from `regmap-irq.c` implicitly assumes that an interrupt controller device should be the same as a device on regmap-supported bus, the assumption does not cover an interrupt controller sub-device on MFD:

```
if (irq_base)
        d->domain = irq_domain_add_legacy(map->dev->of_node,
                                          chip->num_irqs, irq_base, 0,
                                          &regmap_domain_ops, d);
else
        d->domain = irq_domain_add_linear(map->dev->of_node,
                                          chip->num_irqs,
                                          &regmap_domain_ops, d);
```

Interrupt line (wanted)

```
display_0: pcb-display-0 {
        compatible = "ti,ds90ux9xx", "simple-mfd";

        i2c-bridge {
                compatible = "ti,ds90ux9xx-i2c-bridge";
                #address-cells = <1>;
                #size-cells = <0>;

                touchscreen@4b {
                        compatible = "atmel,maxtouch";
                        reg = <0x4b>;
                        interrupt-parent = <&intc0>;
                        interrupts = <0>;
                        atmel,mtu = <200>;
                };
        };
        /*   ...    */
};
```

```
ds90ux9xx_i2c1: serializer@c {
        compatible = "ti,ds90ux9xx", "simple-mfd";
        reg = <0xc>;
        power-gpios = <&gpio5 12 GPIO_ACTIVE_HIGH>;
        ti,backward-compatible-mode = <0>;
        ti,low-frequency-mode = <0>;

        intc0: interrupt-controller {
                compatible = "ti,ds90ux9xx-intc";
                interrupt-parent = <&gpio5>;
                interrupts = <5 0>;
                interrupt-controller;
                #interrupt-cells = <1>;
        };
        /*    ...    */
};
```

Future work

- Send the existing code for upstream inclusion
- Device drivers for TI DS90Ux9xx are quite good, but there are more features to support (HDCP, video bridge controls, I2S bridge etc.) and more IC types like FPD-Link III hubs
- Consider a generalization of FPD-Link III, GMSL and MOST, is there a need for a new driver framework?
  - I2C bridge changes for connection detection and registration of remote side I2C devices may be reused
  - Device tree bindings are wanted to be similar

Questions

Thank you for your attention!

Questions and comments are welcome.