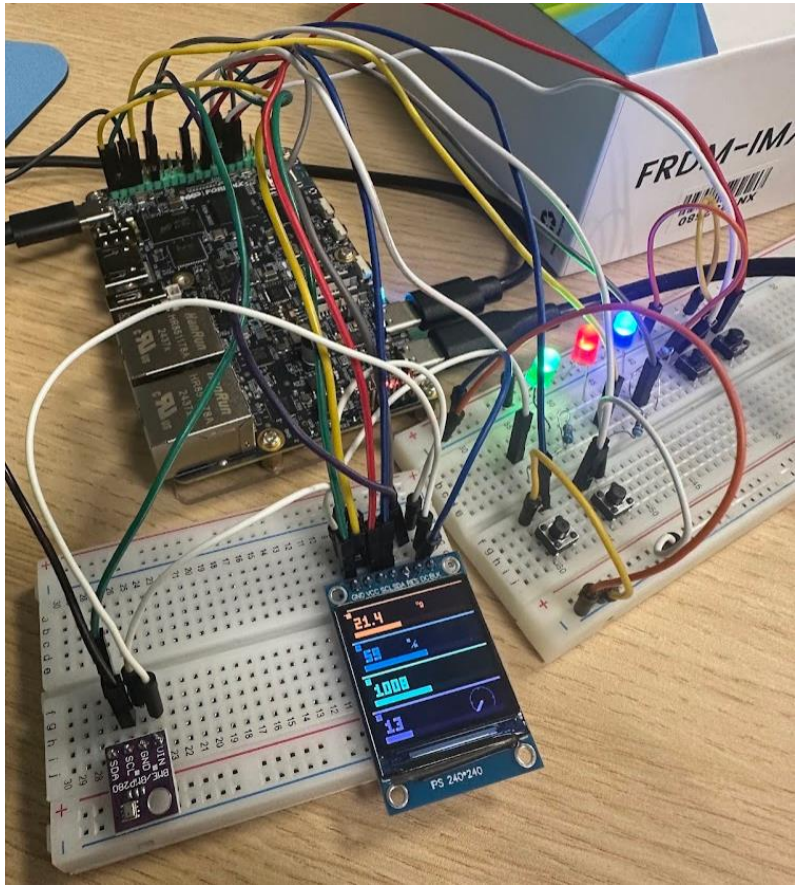
LKSS Day 3: SPI bus and the ST7789 display

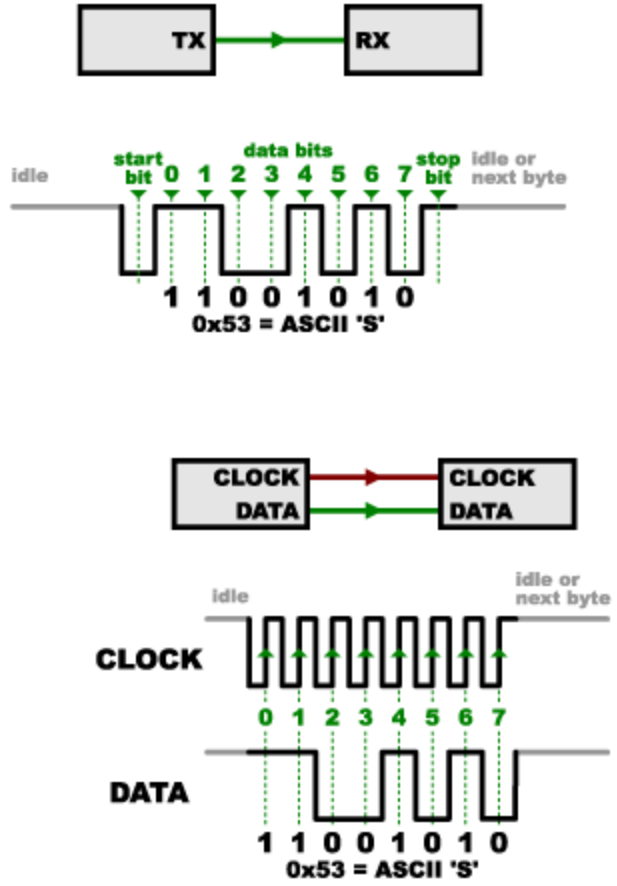
Connect the ST7789 display to i.MX93 FRDM!



Asynchronous vs Synchronous Communication Protocols

- Asynchronous (e.g UART)
  - Sides must agree on a communication speed
  - Insert START/STOP bits for synchronization

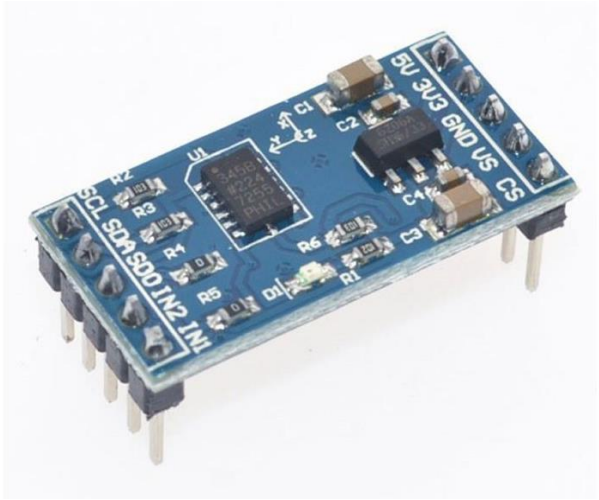
- Synchronous (e.g SPI, I2C)
  - Device share a common clock signal
  - Data is sampled on clock edges
  - Requires an additional SCLK
  - Higher data rates



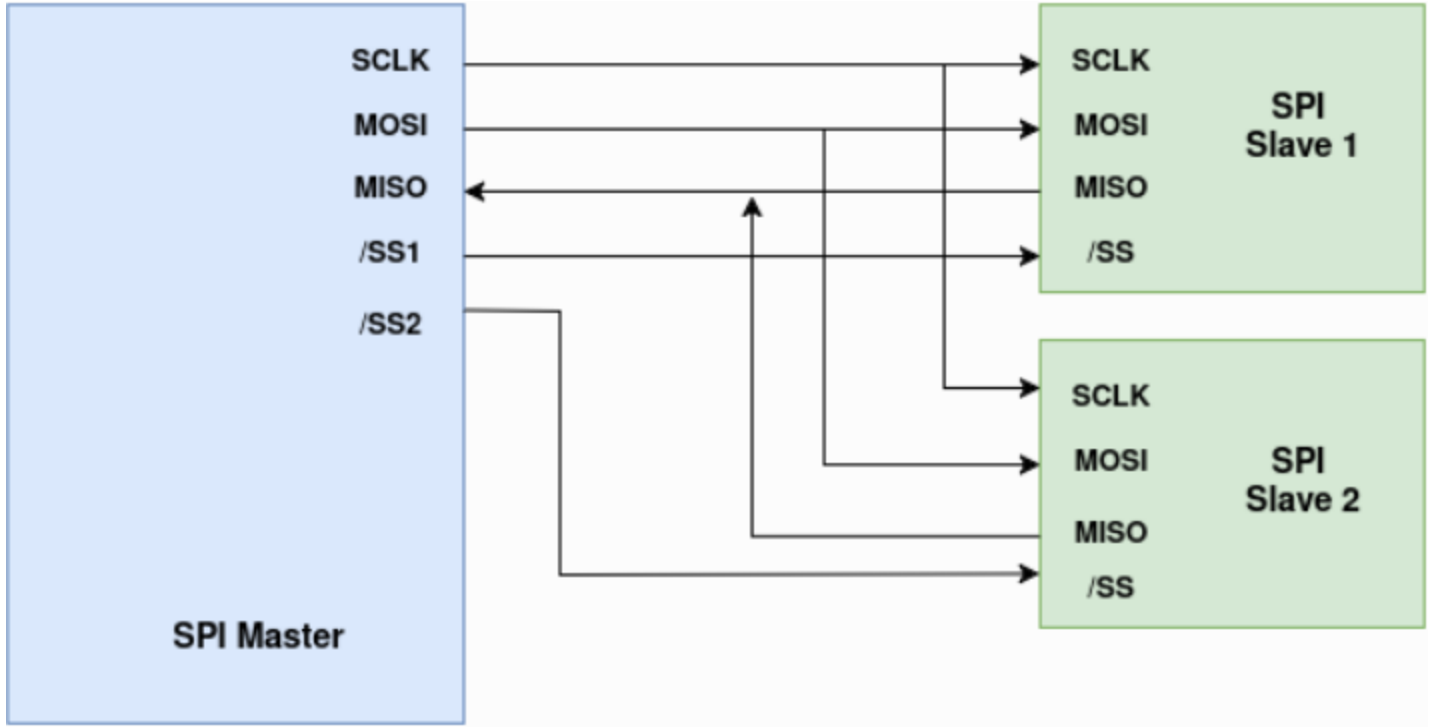
Serial Peripheral Interface (SPI)

- Synchronous
  - Dedicated clock line
- Full duplex
  - Separate lines for TX/RX
- High speed communication (1-100Mhz)
  - Master (Controller)
  - Slaves (Peripherals)
- SPI signals
  - SCLK (clock)
  - CS / SS (Chip Select / Slave Select)
  - MOSI (Master Out / Slave In)
  - MISO (Master In / Slave Out)

Signal	Direction
SCLK	Master -> Slave
MOSI	Master -> Slave
MISO	Slave -> Master
CS/SS	Master -> Slave



Connecting multiple peripherals to SPI bus



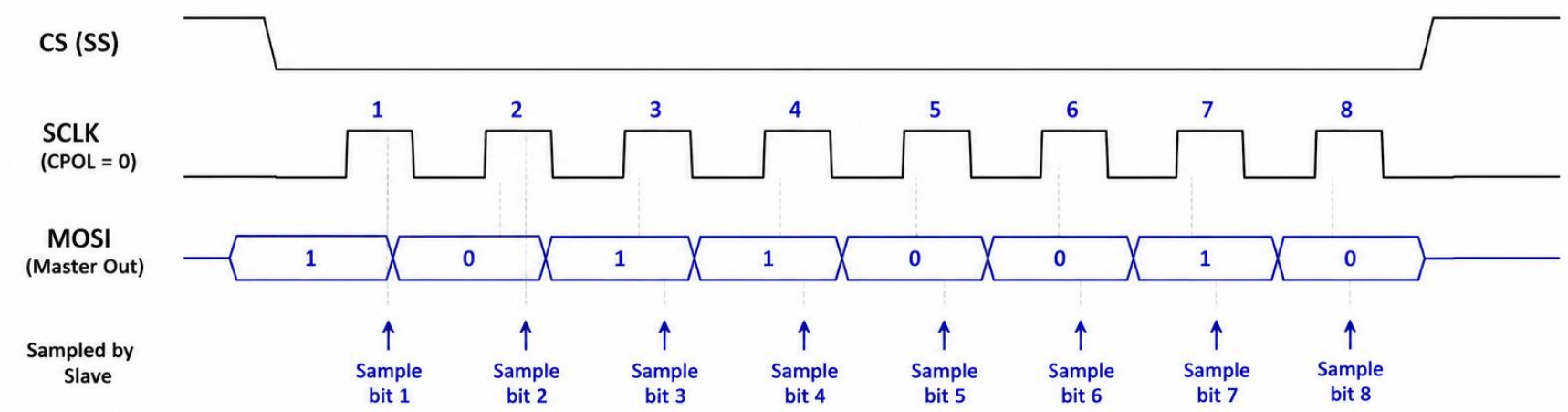
SPI configuration parameters

- Clock Frequency (SCLK) - generated by master
- Clock Polarity (CPOL)
  - Defines whether the clock is idle (rest state) **Low (0)** or **High (1)**.
- Clock Phase (CPHA)
  - Determines whether data is sampled on the **first** (leading) or **second** (trailing) clock edge.
  - **CPHA 0** : Data is sampled ("read") on the first edge and shifted "written" on the second edge.
  - **CPHA 1** : Data is sampled ("read") on the second edge and shifted "written" on the first edge.
- Bit Order
  - LSB or MSB first
- Frame size
  - Typically 8 or 16 bits

SPI Modes

- Default mode 0

CPOL	CPHA	Mode	Behaviour
0	0	Mode 0	SCKL idles **low**, shifted on falling edge, sampled on **rising** edge.
0	1	Mode 1	SCKL idles **low**, shifted on rising edge, sampled on **falling** edge.
1	0	Mode 2	SCKL idles **high**, shifted on falling edge, sampled on **falling** edge
1	1	Mode 3	SCKL idles **high**, shifted on rising edge, sampled on **rising** edge.



Simple SPI driver, spi_write()

```c
/* Example payload written to the device at probe time */
static const u8 lkss_spi_init_cmd[] = { 0x01, 0x02, 0x03 };

static int lkss_spi_probe(struct spi_device *spi)
{
	int ret;

	/* Apply any mode/speed/bits_per_word set in the DT node or spi_board_info */
	ret = spi_setup(spi);
	if (ret) {
		dev_err(&spi->dev, "spi_setup failed: %d\n", ret);
		return ret;
	}

	ret = spi_write(spi, lkss_spi_init_cmd, sizeof(lkss_spi_init_cmd));
	if (ret) {
		dev_err(&spi->dev, "spi_write failed: %d\n", ret);
		return ret;
	}

	dev_info(&spi->dev, "lkss_spi_simple: probe write succeeded\n");

	return 0;
}
```

int **spi_write**(struct spi_device * *spi*, const void * *buf*, size_t *len*)

SPI synchronous write

ST7789 display

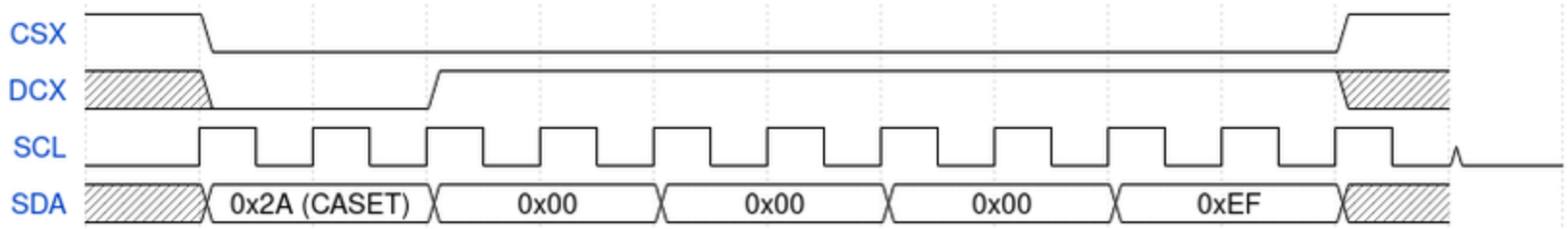
- Maximum resolution: **240 (H) × 320 (V)** RGB pixels
- On-chip frame buffer: 240 × 320 × 18 bits
- Pixel format used: **16 bpp (RGB565)**
- 4-wire SPI interface (SCL, SDA, CS, DCX)
  - **no MISO needed for display use**
- Operating supply: 3.3 V
- The 240×240 module physically crops the 240×320 controller

ST7789 pinout

Pin Name	Direction	Description
GND	Power	Ground reference.
VCC	Power	Connect to 3.3 V only.
SCL (SCK)	Input	Serial clock. Maximum write frequency **62.5 MHz**
SDA (MOSI)	Input	Serial data in. Clocked on rising edge of SCL (SPI Mode 0).
RES	Input	Hardware reset, **active-low**.
DC	Input	Data / Command select. **LOW** = command **HIGH** = data
BLK	Input	Backlight enable. Connect to 3.3 V for always-on backlight.
CS	Input	Chip select, **active-low**. Tied to GND on the module (always selected).

4-wire SPI protocol

- DC(Data / Command) pin




```
static int st7789_write_cmd(struct st7789_priv *priv, u8 cmd)
{
»       gpiod_set_value(priv->dc, 0);
»       return spi_write(priv->spi, &cmd, 1);
}
```

```
static int st7789_write_data(struct st7789_priv *priv,
»       »       »               const u8 *buf, size_t len)
{
»       gpiod_set_value(priv->dc, 1);
»       return spi_write(priv->spi, buf, len);
}
```

Display (minimal) initialization

```c
static int st7789_init_display(struct st7789_priv *priv)
{

  st7789_write_cmd(priv, ST7789_SLPOUT);    /* exit sleep; wait 500ms */
  msleep(500);
  st7789_write_cmd(priv, ST7789_COLMOD);    /* pixel format           */
  st7789_write_data_byte(priv, 0x55);       /* 0x55 = RGB565          */
  st7789_write_cmd(priv, ST7789_MADCTL);    /* memory access control  */
  st7789_write_data_byte(priv, 0x00);       /* normal orientation     */
  st7789_write_cmd(priv, ST7789_INVON);     /* inversion on           */
  st7789_write_cmd(priv, ST7789_NORON);     /* normal display mode    */
  st7789_write_cmd(priv, ST7789_DISPON);    /* display on; wait 100ms */
  msleep(100);
  return 0;
}
```

```c
st7789_hw_reset(priv);

ret = st7789_init_display(priv);
if (ret) {
»       dev_err(&spi->dev, "init failed: %d\n", ret);
»       return ret;
}
```

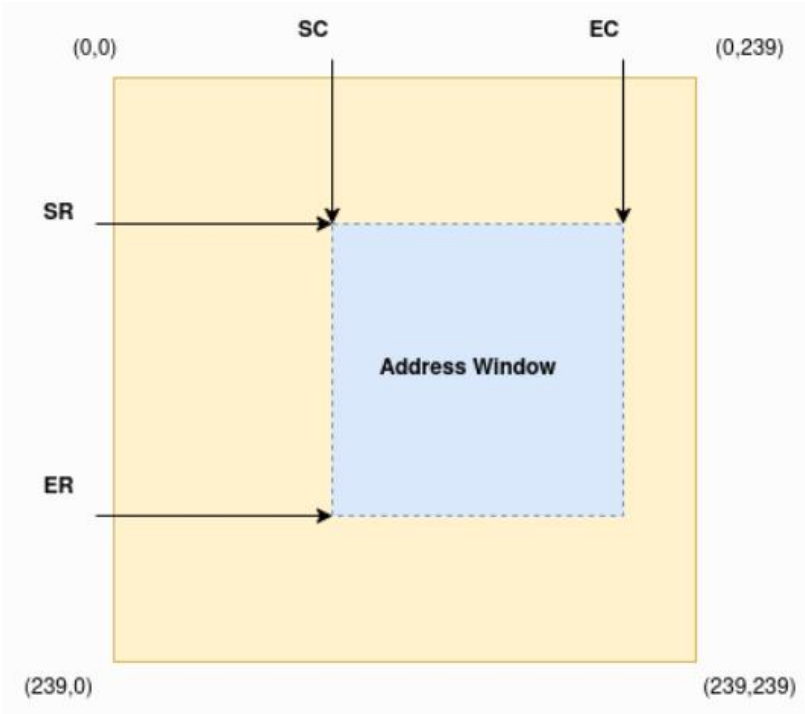
Pixel format

- RGB565 (5 bits red, 6 bits green, 5 bits blue)

Color	RGB565 value	Components (R, G, B)
Red	`0xF800`	R=31 (max), G=0, B=0
Green	`0x07E0`	R=0, G=63 (max), B=0
Blue	`0x001F`	R=0, G=0, B=31 (max)
White	`0xFFFF`	R=31 (max), G=63 (max), B=31 (max)
Black	`0x0000`	R=0, G=0, B=0
Yellow	`0xFFE0`	R=31 (max), G=63 (max), B=0
Cyan	`0x07FF`	R=0, G=63 (max), B=31(max)
Magenta	`0xF81F`	R=31 (max), G=0, B=31(max)

Theory of operation

- Display Data RAM (GRAM) 240 x 240 x 16
- Address Window
  - Use CASET command to set horizontal boundary
  - Use RASET command to set vertical boundary
  - Use RAMWR command to open streaming window




```c
static int st7789_set_addr_win(struct st7789_priv *priv,
                               u16 x0, u16 y0, u16 x1, u16 y1)
{
    u8 col[4] = { x0 >> 8, x0 & 0xff, x1 >> 8, x1 & 0xff };
    u8 row[4] = { y0 >> 8, y0 & 0xff, y1 >> 8, y1 & 0xff };
    int ret;

    ret = st7789_write_cmd(priv, ST7789_CASET);
    if (ret) return ret;
    ret = st7789_write_data(priv, col, 4);
    if (ret) return ret;

    ret = st7789_write_cmd(priv, ST7789_RASET);
    if (ret) return ret;
    ret = st7789_write_data(priv, row, 4);
    if (ret) return ret;

    return st7789_write_cmd(priv, ST7789_RAMWR);
}
```

Device tree (use SPI4 node)

```
&lpspi4 {
    #address-cells = <1>;
    #size-cells    = <0>;
    status         = "okay";

    pinctrl-0    = <&pinctrl_lpspi4_st7789>;
    pinctrl-names = "default";

    lkss_st7789: st7789@0 {
        compatible        = "lkss,st7789";
        reg               = <0>;            /* chip-select index 0   */
        spi-max-frequency = <62500000>;

        /* D/C:   GPIO2_IO13 (J601 pin 22), active-high = data */
        dc-gpios    = <&gpio2 13 GPIO_ACTIVE_HIGH>;

        /* Reset: GPIO2_IO19 (J601 pin 15), active-low */
        reset-gpios = <&gpio2 19 GPIO_ACTIVE_LOW>;
    };
};
```