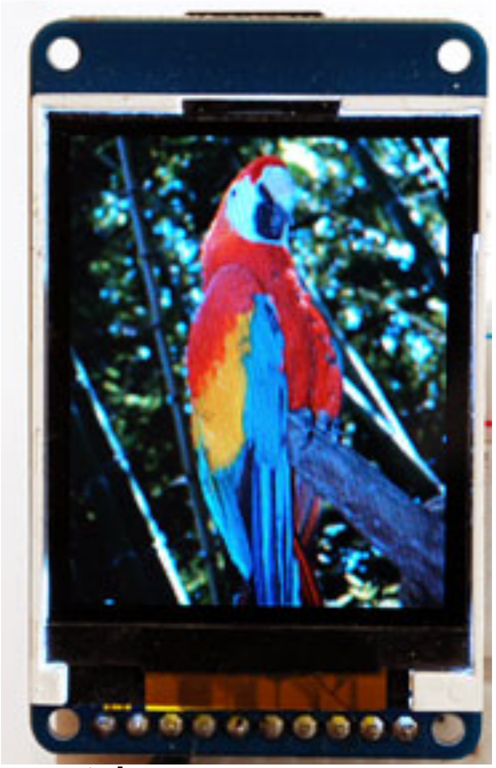
- How did this project come about? (or how to mix work and fun)
- SPI display controllers and the little bitty Adafruit display
- What's my obsession with Arduino and BeagleBone about?
- Linux, SPI, and display drivers
- Dissection of major organs in the driver
- Debugging: a tool for the masses, the OBLS
- Problems Problems Everywhere…
- Obligatory demo
- Q&A

- Customer:
  - "We don't understand how to use EDMA in our Linux SPI display driver"
- Field:
  - "There are no examples! It's too complex in Linux! There's no [fine] manual!"
- Manager:
  - "How can we help the customer?"
- Me:
  - Reviews customer driver that ignores all existing Linux driver frameworks
  - "Tell you what, it'll probably be easier to just write their driver for them as an example if the Linux FB and SPI docs are not sufficient."



- http://www.adafruit.com/products/358
- 128x160 resolution, 16-bit color
- ST7735 display controller
  - http://www.adafruit.com/datasheets/ST7735R_V0.2.pdf
- 3.3V/5.0V tolerant I/O
- LCD and and controller on a breakout board with header strip
  - Some assembly required
- Chip selects provided for both the ST7735 controller and for a uSD slot on the board
  - uSD isn’t very exciting for our purposes

- SPI or parallel connection
- Internal display data RAM contents drive display output
- In 4-wire serial mode, requires MOSI, CS, SCLK, RESET, and D/C
  - D/C (Data/Command mode) is an out-of-band signal driving SPI bus transfers to either the internal RAM or the internal register file, respectively
- SPI Mode 3
  - CPOL=1 (clock base high)
  - CPHA=1 (data setup on falling edge, latch on rising edge)
- Max clock frequency of 15MHz
  - More on this later…

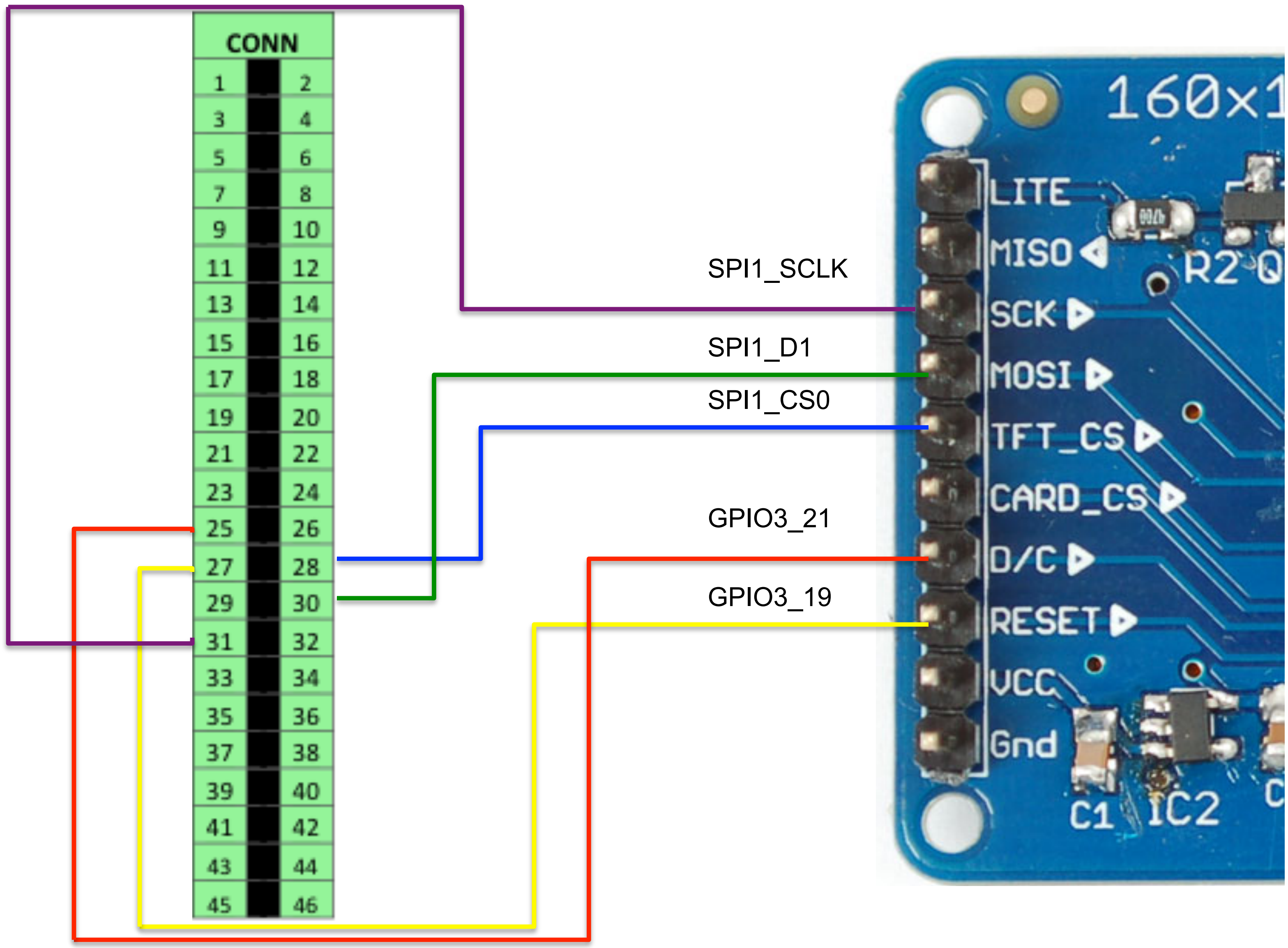
- Pixel formats
  - RGB444
  - RGB565
  - RGB666

- Basic operation
  - Send commands to init controller for display specific settings
  - Configure internal ram row/column window to write when data asserted
  - Assert data mode and perform SPI transfers to write pixel data

- The differences are quickly obvious
  - Arduino carries a lowly microcontroller and minimal peripheral support
  - Beaglebone carries a Cortex A8 core and loads of peripherals
- But what makes them similar?
  - Design choices…BeagleBone set out to fill in the higher end need for hobbyists to interface with an SoC that runs Linux has much more processing power.
    - Both provide standardized expansion headers for standardized shields or capes to be stacked.
    - 5V or 3.3V tolerant I/O (depends on Arduino model) for simple interfacing
  - Both have strong communities
    - Just about every part or breakout board you can buy at popular outlets like Sparkfun and Adafruit have Arduino libraries
    - Beagleboard.org has an active community for existing boards and many of those users are also using BeagleBone

- Two 48 pin expansion connectors P8 and P9
- P8 has pins with GPIO, GPMC, LCD, Timers, PWM/QEP, McASP, UART and MMC capabilities
- P9 has pins with GPIO, SPI, I2C, GPMC, MII/GMII/RGMII, UART, Timers, PWM, CAN, McASP, and MMC
- All expansion header I/O is 3.3V
  - Easy interfacing of current parts and breakout boards
- P9 has everything we need to interface the Adafruit 1.8” LCD

SIGNAL NAME	PIN	CONN		PIN	SIGNAL NAME
	GND	1	2	GND	
	VDD_3V3EXP	3	4	VDD_3V3EXP	
	VDD_5V	5	6	VDD_5V	
	SYS_5V	7	8	SYS_5V	
PWR_BUT*		9	10	A10	SYS_RESETn
UART4_RXD	T17	11	12	U18	GPIO1_28
UART4_TXD	U17	13	14	U14	EHRPWM1A
GPIO1_16	R13	15	16	T14	EHRPWM1B
I2C1_SCL	A16	17	18	B16	I2C1_SDA
I2C2_SCL	D17	19	20	D18	I2C2_SDA
UART2_TXD	B17	21	22	A17	UART2_RXD
GPIO1_17	V14	23	24	D15	UART1_TXD
GPIO3_21	A14	25	26	D16	UART1_RXD
GPIO3_19	C13	27	28	C12	SPI1_CS0
SPI1_D0	B13	29	30	D12	SPI1_D1
SPI1_SCLK	A13	31	32	VDD_ADC	
AIN4	C8	33	34	GNDA_ADC	
AIN6	A5	35	36	A5	AIN5
AIN2	B7	37	38	A7	AIN3
AIN0	B6	39	40	C7	AIN1
CLKOUT2	D14	41	42	C18	GPIO0_7
	GND	43	44	GND	
	GND	45	46	GND	


CONN
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
SPI1_SCLK
SPI1_D1
SPI1_CS0
GPIO3_21
GPIO3_19
LITE
MISO
SCK
MOSI
TFT_CS
CARD_CS
D/C
RESET
VCC
Gnd
R2
C1
IC2

- Ignore the Linux SPI framework
- Ignore the Linux framebuffer framework
- Ignore the Linux GPIO framework
- Ignore the platform pinmux (or generic pinctrl/pinmux) framework
- Write a misc driver
  - Implement your own pinmux routines, bang on hardware directly
  - Implement your own GPIO routines, bang on hardware directly
  - Implement your own SPI transfer routines, banging on the hardware directly
  - Implement a display driver by transferring a display buffer via write()

- When in doubt – assume everything you're about to do has been done before
- Linux SPI subsystem
  - http://www.kernel.org/doc/Documentation/spi/spi-summary
- Linux GPIO subsystem
  - http://kernel.org/doc/Documentation/gpio.txt
- Linux framebuffer subsystem
  - http://kernel.org/doc/Documentation/fb/framebuffer.txt
  - http://kernel.org/doc/Documentation/fb/deferred-io.txt
- Pinmuxing might be the only thing that's underdocumented and completely arch specific (today)…but there are examples.

```
static const struct st7735fb_platform_data bone_st7735fb_data = {
        .rst_gpio       = GPIO_TO_PIN(3, 19),
        .dc_gpio        = GPIO_TO_PIN(3, 21),
};
```

Convert the ST7735 reset signal on GPIO 3_19 to a unique Linux GPIO value.

Convert the ST7735 data/ command signal on GPIO 3_21 to a unique Linux GPIO value.

```
static struct spi_board_info bone_spi1_slave_info[] = {
	{
		.modalias		= "adafruit_tft18",
		.platform_data	= &bone_st7735fb_data,
		.irq			= -1,
		.max_speed_hz	= 8000000,
		.bus_num		= 2,
		.chip_select	= 0,
		.mode		= SPI_MODE_3,
	},
};
```

McSPI bus numbering starts at 1 so spi1 is bus 2.

McSPI bus numbering starts at 1 so spi1 is bus 2.

Mode 3 corresponds to CPOL/CPHA == 1.

```
/* setup spi1 */
static void spi1_init(int evm_id, int profile)
{
	setup_pin_mux(spi1_pin_mux);
	spi_register_board_info(am335x_spi1_slave_info,
			ARRAY_SIZE(am335x_spi1_slave_info));
	return;

}
```

DO NOT forget to set up your platform's pin muxes!!!

Finally! Register our SPI slave device(s) with the device model.

```
static struct spi_driver st7735fb_driver = {
	.driver = {
		.name		= "st7735fb",
		.owner		= THIS_MODULE,
	},
	.id_table	= st7735fb_ids,
	.probe		= st7735fb_probe,
	.remove		= __devexit_p(st7735fb_remove),
};
```

Our framebuffer driver entry point. Use the existing FB skeletonfb or another similar driver from here.

- Traditional framebuffer driver relies on video memory on the “graphics card” or in system memory which directly drives the display.
  – This framebuffer is what is exposed to userspace via mmap().
- For SPI and other indirect bus connections to a display controller, we can’t directly expose the internal display controller memory to userspace.
  – USB DisplayLink
- With deferred I/O and an indirect display connection, userspace can be presented with a kernel buffer that can be mmaped
  – Userspace writes to mmapped buffer
  – Deferred I/O framework records page faults and maintains a list of modified pages to pass to the device driver deferred i/o handler on a periodic basis
  – Driver handler then performs bus-specific transfers to move the data from the modified pages to the display controller

```
static void st7735fb_deferred_io(struct fb_info *info, struct list_head *pagelist)
{
	st7735fb_update_display(info->par);
}

static struct fb_deferred_io st7735fb_defio = {
	.delay		= HZ/20,
	.deferred_io	= st7735fb_deferred_io,
};

…
info->fbdefio = &st7735fb_defio;
fb_deferred_io_init(info);
…
```

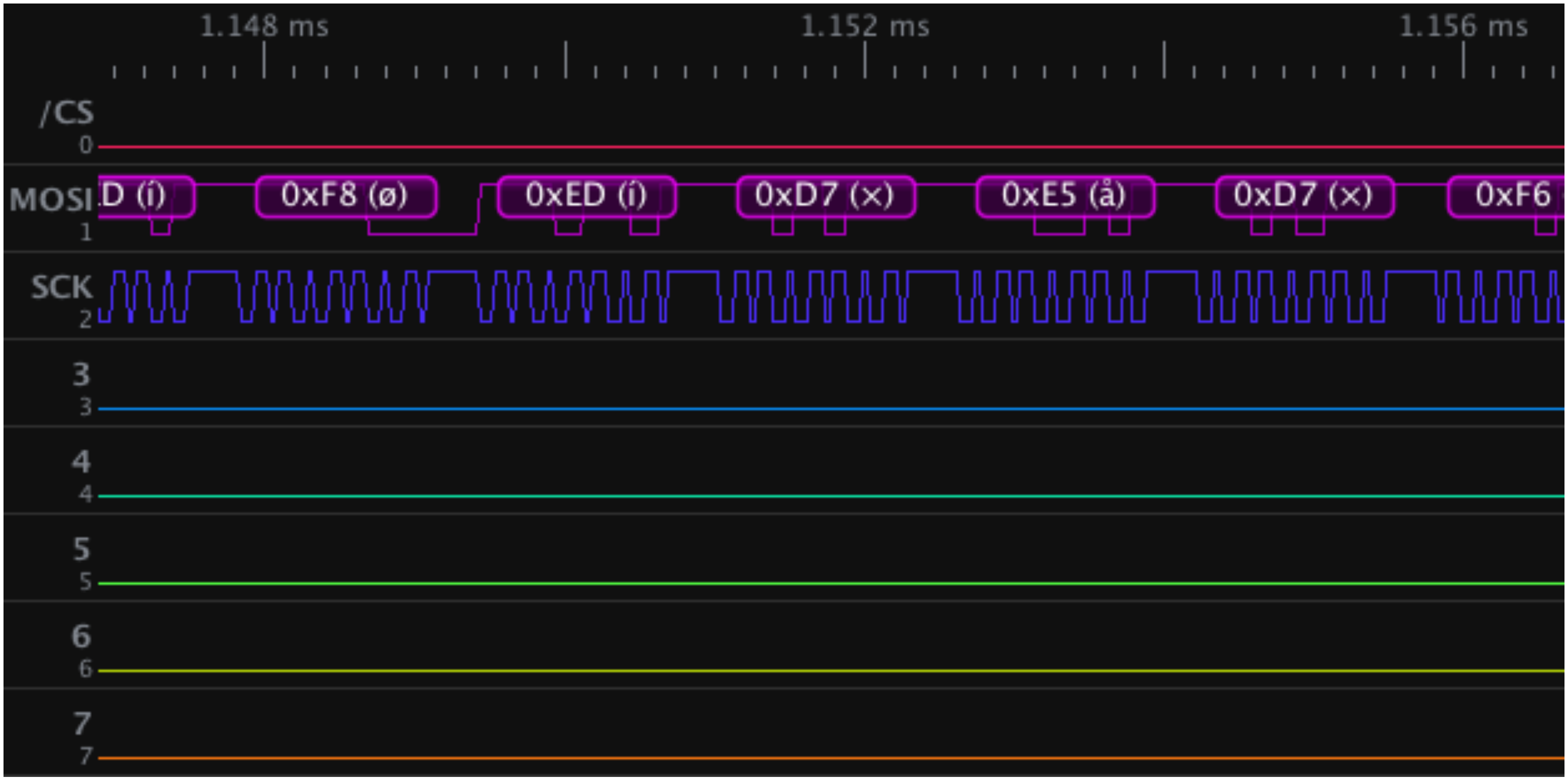
```
static void st7735fb_update_display(struct st7735fb_par *par)
{
    int ret = 0;
    u8 *vmem = par->info->screen_base;
    /* Set row/column data window */
    st7735_set_addr_win(par, 0, 0, WIDTH-1, HEIGHT-1);
    /* Internal RAM write command */
    st7735_write_cmd(par, ST7735_RAMWR);
    ret = st7735_write_data_buf(par, vmem, WIDTH*HEIGHT*2);
    if (ret < 0)
            pr_err("%s: spi_write failed to update display buffer\n", par->info->fix.id);
}
```

FB is ~40KiB, ignore the pagelist and write the entire thing every time

- JTAG
  - External (BDI2000/3000, Flyswatter, etc)
  - Onboard (BeagleBone has FTDI2232H)
  - OpenOCD (http://openocd.sourceforge.net/)
- Logic Analyzer
  - Salae ($149)
    - http://www.saleae.com
  - Open Bench Logic Sniffer ($50)
    - http://dangerousprototypes.com/docs/Open_Bench_Logic_Sniffer
    - http://ols.lxtreme.nl/
    - http://sigrok.org/wiki/Main_Page

- Logic Analyzer
- 16 buffered channels (-0.5V to 7V tolerant)
  - Additional 16 channels can be enabled by adding a buffered “wing”
- Up to 200MHz bandwidth depending on channel configuration
- USB powered
- USB connectivity (CDC ACM)
- Completely open hardware
- Many client choices

- Modified SUMP
  - Java
- OLS (alternative java client)
  - Java
  - Several protocol decoders
- Sigrok
  - Cross platform C
  - Extendable with Python-based protocol decoders
    - Some early ones in place


1.148 ms
1.152 ms
1.156 ms
/CS
0
MOSI
1
D (í)
0xF8 (ø)
0xED (í)
0xD7 (×)
0xE5 (å)
0xD7 (×)
0xF6
SCK
2
3
3
4
4
5
5
6
6
7
7

- Tried the display on an Arduino Uno first, gotta love how everything comes with an Arduino sketch library these days
  - Same sequence on BeagleBone, epic fail
- AM335x TRM shows SPI1_D0 being the MOSI output, it is not. MOSI is found on SPI1_D1
- Originally tried to drive at max 15MHz SPI clock rate, this was another fail.
  - The Adafruit breakout board adds a CD4050B level shifter to be 5V tolerant for Arduino. This chip is slow and limits the clock rate to <10MHz, driving my change to 8MHz for the spi device registration.
  - Some hardware hacks can get around this:
    - http://fabienroyer.wordpress.com/2011/05/29/driving-an-adafruit-st7735-tft-display-with-a-netduino/

- The 16-bit pixel format presented an issue with userspace compatibility
  - All userspace application assume that framebuffers are organized in a native endian format.
  - On our little endian ARM system, the mmaped shadow framebuffer is written in native little endian.
  - SPI buffer transfers in 8-bit data mode required by the ST7735 do a byte swap by nature of the byte-wise addressing of the PIO or DMA based memory access
    - Have to present the SPI adapter driver with a byte swapped shadow buffer
    - Driver has hack which byte swaps the buffer before doing a spi_write() on every deferred_io update. This allows unmodified use existing FB API applications

- fbv displaying a JPEG
- Capture and SPI protocol decode of display transferring framebuffer data during display update

- ST7735FB driver
  - https://github.com/ohporter/linux-am33x/tree/st7735fb
- ST7586FB driver
  - https://github.com/ohporter/linux/tree/st7586fb
- Enlightenment running on the ST7735FB driver
  - http://www.youtube.com/watch?v=MIb-1ZeVik0