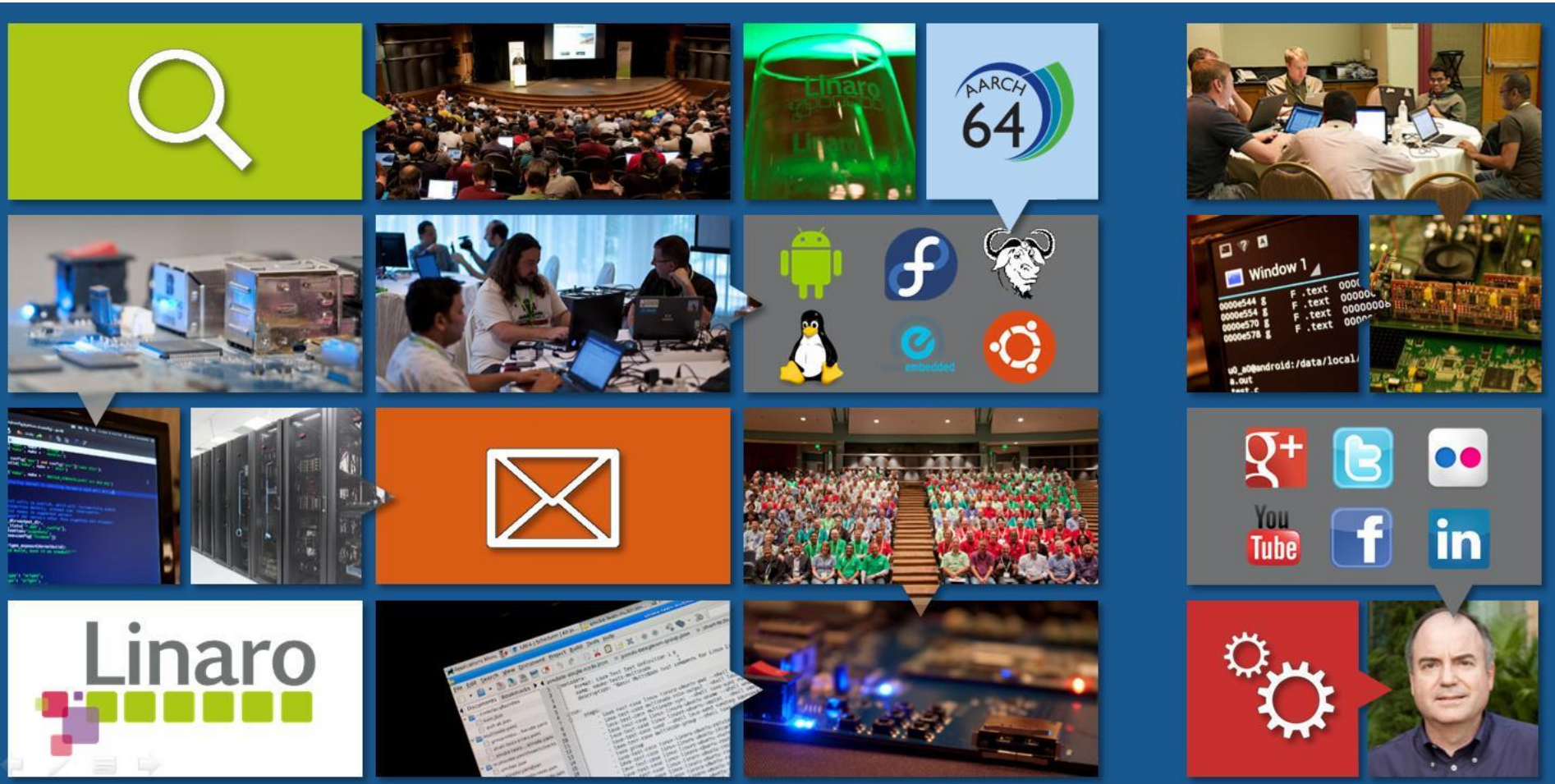
What's going on with SPI?

ELC, May 2014



Overview

- Hardware overview
- Framework overview
- Recent enhancements
- Future plans

What is SPI?

Simple bidirectional serial bus with four signals:

- Master Out Slave In (MOSI)
- Master In Slave Out (MISO)
- Clock
- Chip select

- Little endian byte ordering for words

What is SPI?

Comparable with I2C:

- Four wires instead of two
- Typically 1-2 orders of magnitude faster
- Full duplex
- Very simple implementation

Applications

- Flash
- Mixed signal ICs
- DSPs

Controller hardware

- No support at all, using GPIOs
  - Very slow and inefficient
  - Commonly used for chip select
- PIO based FIFOs
  - Less slow
  - Requires CPU access every word
- DMA based FIFOs
  - Less work for CPU
  - Higher setup overhead
  - Faster for large blocks of data
- Dual and quad mode
  - Extra data lines, mainly used with flash (v3.12)
- Specialised flash controllers

Controller hardware

- No support at all, using GPIOs
  - Very slow and inefficient
  - Commonly used for chip select
- PIO based FIFOs
  - Less slow
  - Requires CPU access every word
- DMA based FIFOs
  - Less work for CPU
  - Higher setup overhead
  - Faster for large blocks of data
- Dual and quad mode
  - Extra data lines, mainly used with flash (v3.12)
- Specialised flash controllers

Basic software stack

Originally contributed by David Brownell

- Merged in 2.6.16 (released March 2006)
- Largely unchanged until recently

Standard device model bus:

- Controllers and devices
- Device registration via machine driver/firmware

Device interface

Simple message based interface for devices

- List of transfers, for scatter/gather and mixed read/write
- Some settings can change per transfer/message
- Optionally asynchronous

Device interface

```
struct spi_transfer {
    const void*tx_buf;
    void       *rx_buf;
    unsigned     len;
};

void spi_message_init(struct spi_message *m);
void spi_message_add_tail(struct spi_transfer *t,
                          struct spi_message *m);

int spi_async(struct spi_device *spi, struct
              spi_message *message);
int spi_sync(struct spi_device *spi,
             struct spi_message *message);
```

Basic driver interface

Very basic:

```
int (*transfer)(struct spi_device *spi,
                struct spi_message *mesg);
```

Executes in atomic context!

“Bitbang” driver framework

Not just for bitbanging:

```
int (*setup_transfer)(struct spi_device *spi,
                      struct spi_transfer *t);
void   (*chipselect)(struct spi_device *spi, int is_on);
int (*txrx_bufs)(struct spi_device *spi,
                 struct spi_transfer *t);
```

- Factors out logic to do with transfer list
- Can even support DMA

What’s missing?

- No code reuse outside of bitbang
- Lots of wheels of varying shapes
- Good ideas need to be copied

Standard parameter checking and handling

Many ways of specifying/validation same information

- Selecting a transfer speed
- Bits per word settings
- Overriding these per transfer
- Validating buffer sizes

Message queue

```
int (*prepare_transfer_hardware)(struct spi_master *m);
int (*transfer_one_message)(struct spi_master *m,
                struct spi_message *m);
int (*unprepare_transfer_hardware)(struct spi_master *m);
```

- Factors out code
- Standard synchronisation with suspend
- Standard runtime PM implementation
- Standard support for managing priority of pump

Contributed by Linus Walleij, merged in v3.4 (May 2012)

Standard message parsing

Moves more logic from spi_bitbang into core:

```
int (*prepare_message)(struct spi_master *master,
                       struct spi_message *message);
int (*unprepare_message)(struct spi_master *master,
                       struct spi_message *message);

void (*set_cs)(struct spi_device *spi, bool enable);
int (*transfer_one)(struct spi_master *master,
                          struct spi_device *spi,
                          struct spi_transfer *transfer);
```

Merged in v3.13

Standard DMA mapping

Most drivers only handled some cases:

- Buffers need to be mapped before DMA
- Buffers may not be physically contiguous
- vmalloc()ed addresses need different mapping

Drivers provide a callback to check for DMA:

```
bool (*can_dma)(struct spi_master *master,
                struct spi_device *spi,
                struct spi_transfer *xfer);
```

If true passed sg_lists instead of buffers

Dual and quad modes

- Extra data lines for higher speed
- Capability set when registering device
- Enabled per-transfer by device drivers

Contributed by Wang Yuhang, merged in v3.12

What’s next?

Standard GPIO chip select

- Handling controller chip select
- Standard way to set in DT

Latency - spi_sync()

Device driver	SPI thread	Hardware/IRQ
Queue transfer		
Wait...	Schedule	
	Start transfer	Start transfer
	Wait...	Wait...
		Wake SPI
	Schedule	
	Wake driver	
Schedule		
Return		

Latency - spi_async()

Device driver	SPI thread	Hardware/IRQ
Queue transfer		
Wait...	Schedule	
	Start transfer	Start transfer
	Wait...	Wait...
		Wake SPI
	Schedule	
	Wake driver	
Schedule	Start transfer	Start transfer
Return		

Latency - complete in IRQ

Device driver	SPI thread	Hardware/IRQ
Queue transfer		
Wait...	Schedule	
	Start transfer	Start transfer
	Wait...	Wait...
		Wake driver
Schedule	Schedule	
Return		

Latency - start immediately

Device driver	SPI thread	Hardware/IRQ
Queue transfer		Start transfer
Wait...		Wait...
		Wake SPI
		Wake driver
Schedule		Start transfer
Return		

Latency

- Do DMA mapping while prior transfer runs
- Coalesce transfers and use hardware scatter/gather

Pre-validated messages

- Messages validated once and used several times
- Saves iterating and checking
- Allows drivers to keep buffers DMA mapped
- Mainly for very high bandwidth applications

Work being done by Martin Sperl

Fully DMA driven queues

- Use DMA transfers to set chip select and parameters
- Requires dmaengine and gpiolib enhancements
- Extremely low CPU overhead, runs from interrupt

Work being done by Martin Sperl

Summary

- Simple bus, not so simple software
- Much more active development recently
  - New hardware
  - More demanding performance requirements

More about Linaro: http://www.linaro.org/about/
More about Linaro engineering: http://www.linaro.org/engineering/
How to join: http://www.linaro.org/about/how-to-join
Linaro members: www.linaro.org/members