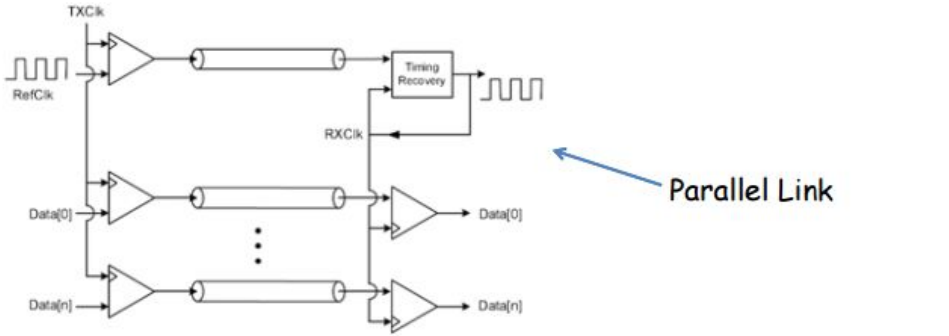
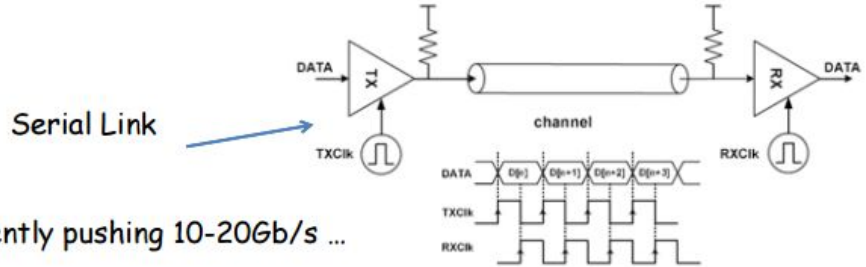
6.111 Serial Data Demo

Miren Bamforth - Fall 2015

Serial and Parallel Links





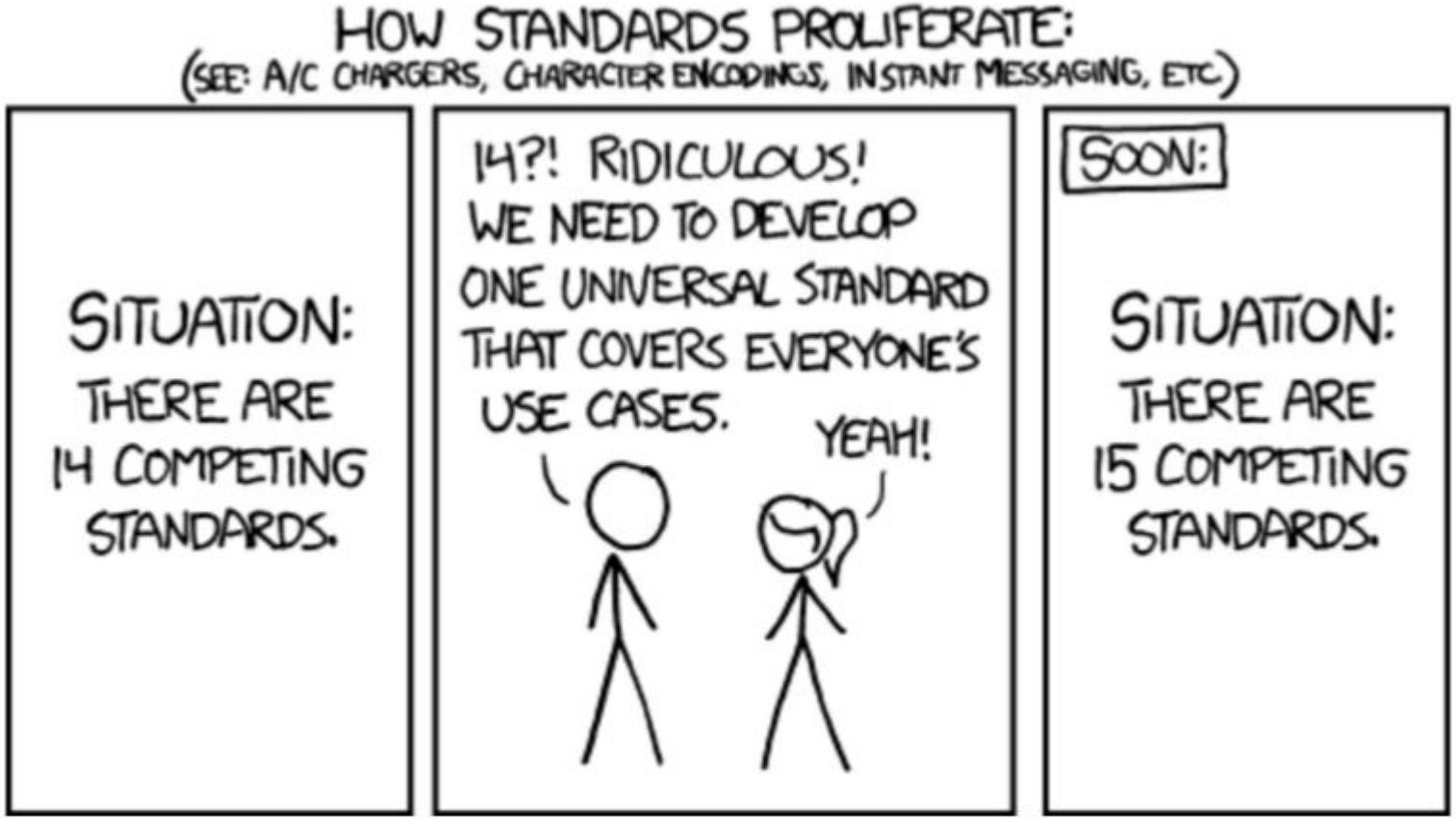
Currently pushing 10-20Gb/s ...

Intro to Serial Data

Serial Communications

- Sending information one bit at a time vs. many bits in parallel
  - Serial: good for long distance (save on cable, pin and connector cost, easy synchronization). Requires "serializer" at sender, "deserializer" at receiver
  - Parallel: issues with clock skew, crosstalk, interconnect density, pin count. Used to dominate for short-distances (eg, between chips).
  - BUT modern preference is for parallel, but independent serial links (eg, PCI-Express x1,x2,x4,x8,x16) as a hedge against link failures.
- A zillion standards
  - Asynchronous (no explicit clock) vs. Synchronous (CLK line in addition to DATA line).
  - Recent trend to reduce signaling voltages: save power, reduce transition times
  - Control/low-bandwidth Interfaces: SPI, I^2C, 1-Wire, PS/2, AC97
  - Networking: RS232, Ethernet, T1, Sonet
  - Computer Peripherals: USB, FireWire, Fiber Channel, Infiniband, SATA, Serial Attached SCSI

Intro to Serial Data



Intro to Serial Data

RS232 (aka "serial port")

- Labkit: simple bidirectional data connection with computer.
- Characteristics
  - Large voltages => special interface chips
    (1/mark: -12V to -3V, 0/space: 3V to 12V)
  - Separate xmit and rcv wires: full duplex
  - Slow transmission rates (1 bit time = 1 baud); most interfaces support standardized baud rates: 1200, 2400, 4800, 9600, 19.2K, 38.4K, 57.6K, 115.2K
  - Format
    - Wire is held at 1/mark when idle
    - Start bit (1 bit of "0" at start of transmission)
    - Data bits (LSB first, can be 5 to 8 bits of data)
    - Parity bit (none, even, odd)
    - Stop bits (1, 1.5 or 2 bits of 1/mark at end of symbol)
    - Most common 8-N-1: eight data bits, no parity, one stop bit

Intro to Serial Data

RS232 interface

- Transmit: easy, just build FSM to generate desired waveform with correct bit timing
- Receive:
  - Want to sample value in middle of each bit time
  - Oversample, eg, at 16x baud rate
  - Look for 1->0 transition at beginning of start bit
  - Count to 8 to sample start bit, then repeatedly count to 16 to sample other bits
  - Check format (start, data, parity, stop) before accepting data.

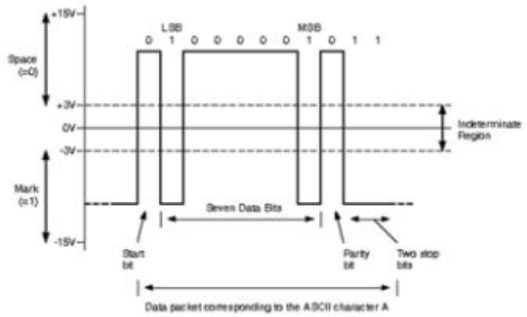
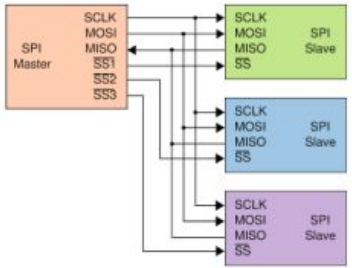


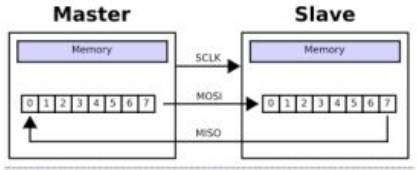
Figure from
http://www.arcelect.com/rs232.htm

Intro to Serial Data

SPI (Serial Peripheral Interface)

- Simple, 3-wire interface + devices selects
  - SCLK generated by master (1-70MHz). Assert data on one edge, sample data on the other. Default state of SCLK and assignment of edges is often programmable.
  - Master Out Slave In (MOSI) data shifted out of master register into slave register
  - Master In Slave Out (MISO) data shifted out of slave register and into master register
  - Selects (usually active low) determine which device is active. Assertion often triggers an action in the slave, so master waits some predetermined time then shifts data.



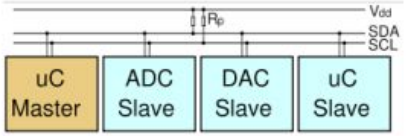


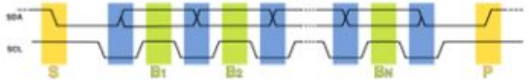
Figures from Wikipedia

Intro to Serial Data

I²C (Inter-Integrated Circuit)

- 2 open-drain wires (SCL = clock, SDA = data)
- Multiple-master, each transmission addresses a particular device, many devices have many different sub-addresses (internal registers)
- Format (all addresses/data send MSB first):
  - Sender: Start [S] bit (SDA↓ while SCL high)
  - Sender: One or more 8-bit data packets, each followed by 1-bit ACK
    - Data changed when SCL low, sampled at SCL↑
    - Receiver: Active-low ACK generated after each data packet
  - Sender: Stop [P] bit (SDA↑ while SCL high)
- SCL and SDA have pullup resistors, senders only drive low, go high-impedance to let pullups make line high (so multiple drivers okay!)
  - Receiver can hold SCL low to stretch clock timing, sender must wait until SCL goes high before moving to next bit.
  - Multiple senders can contend using SDA for arbitration



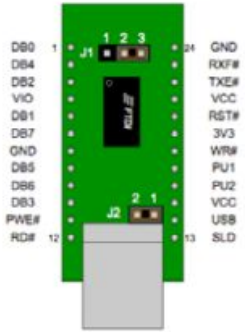


Figures from Wikipedia

Intro to Serial Data

USB (Universal Serial Bus)

- 2-wire (D+,D-) for high-speed, bidirectional polled transmission between master and addressable endpoints in multiple devices. Full speed (12Mbps) and High speed (480Mbps) data rates.
- Multi-level tiered-star topology (127 devices, including hubs)
- FTDI UM245R USB-to-FIFO module for bidirectional data transfer using a handshake protocol, also asynchronous "bit-bang" mode with selectable baud rates.
  - 24-pin DIP module, wire to user pins
  - Drivers for Windows workstations in lab

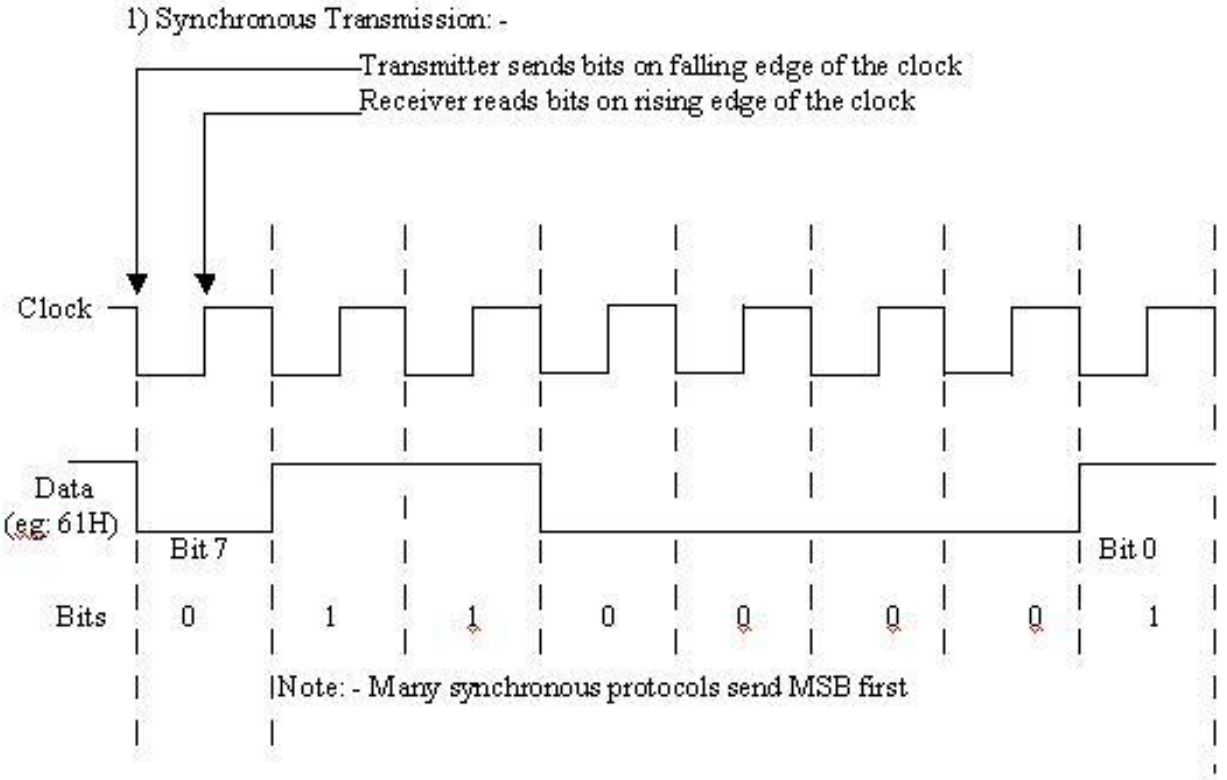


Figures from ftdi.com

Serial data in summary:

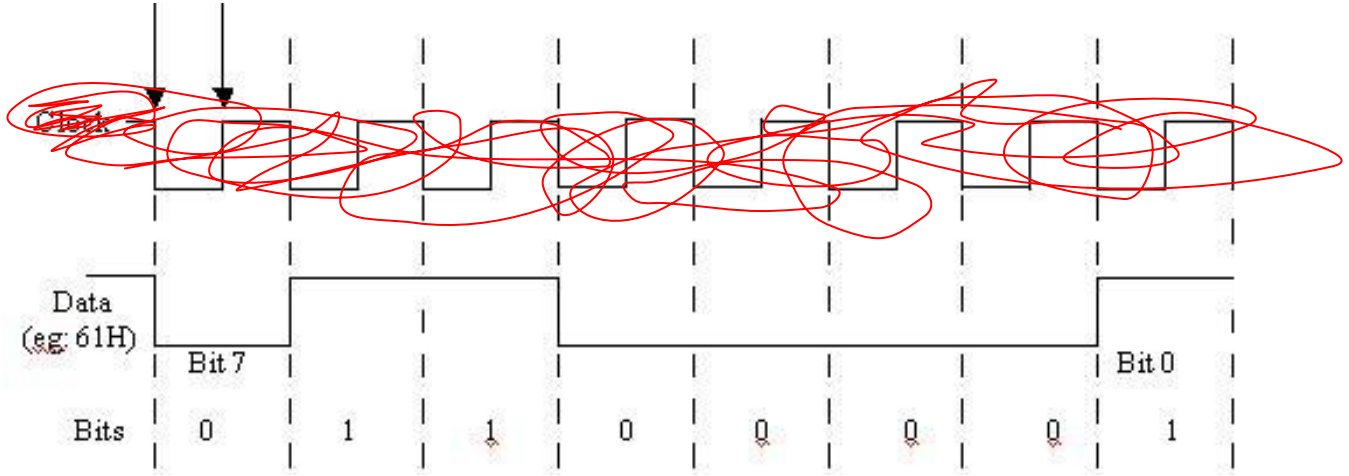
- Serial allows communication with few wires between devices
- Common protocols: UART, SPI, I2C, etc
- Asynchronous vs Synchronous

http://www.eeherald.com/section/design-guide/esmod7.html

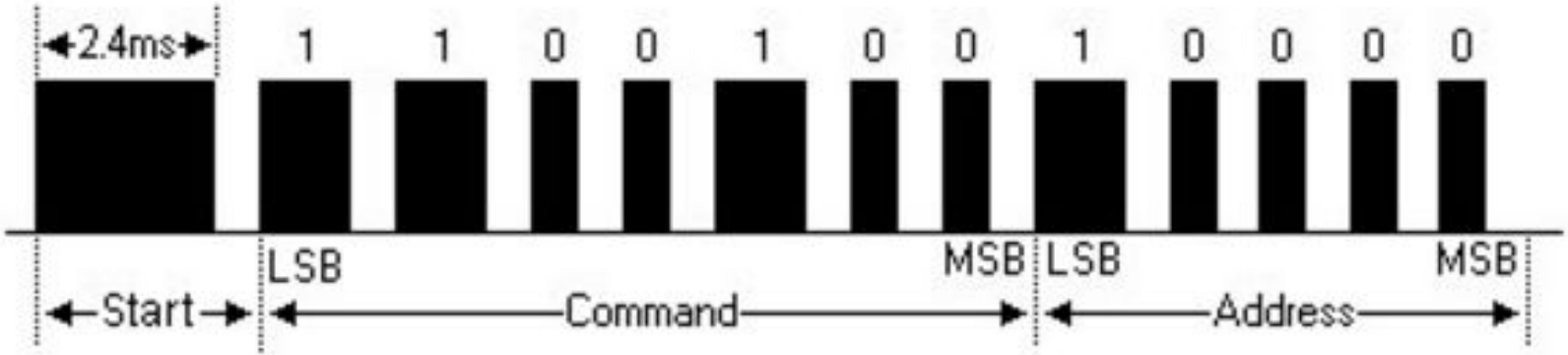


Synchronous Serial Data

Asynchronous means no clock...so how do we know when to look for data bits?



Asynchronous Serial Data



Predetermined timing specifications are the key to asynchronous serial data!

Lab 5b:

- 2.4ms start bit
- 1.2ms 1 bit
- 0.6ms 0 bit

Asynchronous Serial Data: Lab 5b

```
STARTING: begin
        // sample whenever expired is true to read the start bit
        if (expired) begin
                if (ir_clean) begin
                        if (start_counter < 5'b11111) begin
                                start_counter <= start_counter + 1;
                        end
                end
                else if (!ir_clean && ir_prev) begin
                        if (start_counter > 5'b11100) begin
                                state <= READING;
                                bit_counter <= 4'd0;
                                start <= 1'b1;
                        end
                        else begin
                                start_counter <= 5'b00000;
                                state <= IDLE;
                        end
                end
                ir_prev <= ir_clean;
        end
end
```

Asynchronous Serial Data: Lab 5b

The DMX512 protocol is characterized as a asynchronous serial data stream that runs at 250 kHz. As a result, each “bit” will be 4 µs long. DMX512 has one start bit (low), eight bits of data, 2 stop bits and no parity.

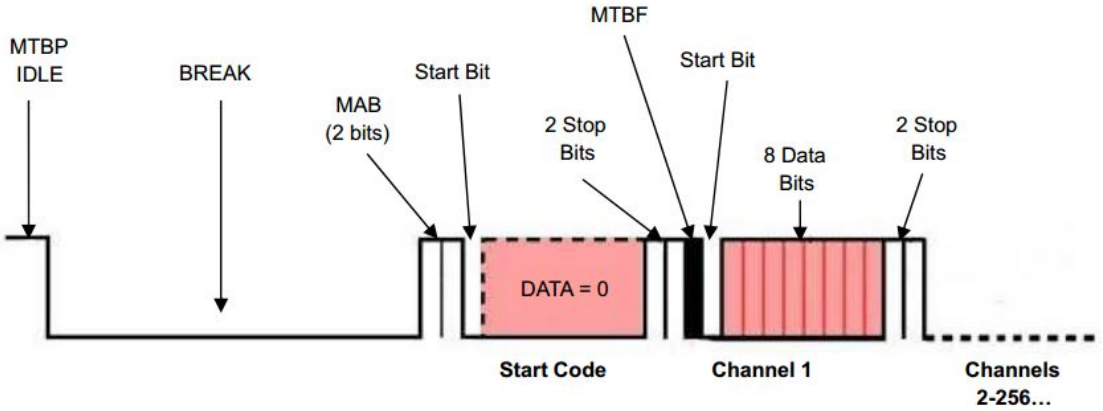


Figure 4. DMX512 Timing Diagram

Table 1. DMX512 Timing Chart

	Min	Typ	Max	Unit
Break	88	88	1000000	us
Mark After Break (MAB)		8		us
Frame Width		44		us
Start/Data/Stop bits		4		us
Mark Time Between Frames (MTBF)	0	N/A	1000000	us
Mark Time Between Packets (MTBP)	0	N/A	1000000	us

Asynchronous Serial Data: DMX512

http://www.ti.com/lit/an/sprabr9/sprabr9.pdf

```
// always block controls dmx output
always @(posedge clk) begin
        if (reset) begin
                state <= BREAK;
        end
        else begin
                case(state)
                        BREAK: begin
                                // hold low for 88 us to 1 sec
                                // choose 100 us which is approx. 2700 cycles at 27MHz
                                if (break_counter < BREAK_COUNTER_MAX) begin
                                        // dmx_out is low during break
                                        dmx_out <= LOW;
                                        break_counter <= break_counter + 1;
                                end
                                else begin
                                        // once we have waited for long enough, go to next state
                                        state <= MAB;
                                        break_counter <= BREAK_COUNTER_MIN;
                                        dmx_out <= HIGH;
                                end
                        end
                        MAB: begin
                                // hold high for 8us to 1 sec
                                // choose 10 us which is approx. 270 cycles at 27MHz
                                if (mab_counter < MAB_COUNTER_MAX) begin
                                        // dmx is high during mark after break
                                        dmx_out <= HIGH;
                                        mab_counter <= mab_counter + 1;
                                end
                                else begin
                                        // once we have waited long enough, go to next state
                                        state <= START_CODE;
                                        mab_counter <= MAB_COUNTER_MIN;
                                        // keep it high for now; the next state will change it
                                        dmx_out <= HIGH;
                                        // reset 4us counter too
                                        four_us_counter <= FOUR_US_COUNTER_MIN;
                                end
                        end
                        START_CODE: begin
                                // The start code is formatted like the channel data with a
                                //      value of zero. In order, its bits are: 0_0000_0000_11.
                                //      The first bit is the start bit (low) and the last two
                                //      bits are stop bits (high).
                                // The total start code is 44us, so each bit is held for 4us
                                if (four_us_counter == FOUR_US_COUNTER_MAX) begin
                                        sc_counter <= sc_counter + 1;
                                        four_us_counter <= FOUR_US_COUNTER_MIN;
                                end
                                else four_us_counter <= four_us_counter + 1;

                                // dmx_out control
                                if (sc_counter < 4'd9) begin
                                        // bits 0 to 8 are low
                                        dmx_out <= LOW;
                                end
                                else if (sc_counter == 4'd9) begin
                                        // bit 9 is high
                                        dmx_out <= HIGH;
                                end
                                else if (sc_counter == 4'd10) begin
                                        // bit 10 is high
                                        // go to the next state
                                        dmx_out <= HIGH;
                                        sc_counter <= SC_COUNTER_MIN;
                                        state <= MTB_FRAMES;
                                end
                        end
                        MTB_FRAMES: begin
                                // The mark between frames is up to 1 second long
                                // choose 10 us which is approx. 270 cycles at 27MHz
                                if (mtbframes_counter < MTBFRAMES_COUNTER_MAX) begin
                                        // dmx is high during mark time between frames
                                        dmx_out <= HIGH;

                                        // request the data for the next channel data output now
                                        if (mtbframes_counter == MTBFRAMES_COUNTER_MIN) begin
                                                // if we have sent all of the frames already, go to
                                                //      the mtb_packets state
                                                if (addr_count == ADDR_COUNT_MAX) begin
                                                        mtbframes_counter <= MTBFRAMES_COUNTER_MIN;
                                                        addr_count <= ADDR_COUNT_MIN;
                                                        state <= MTB_PACKETS;
                                                end
                                                else begin
                                                        request_pulse <= HIGH;
                                                        request_addr <= addr_count;
                                                        addr_count <= addr_count + 1;
                                                        mtbframes_counter <= mtbframes_counter + 1;
                                                end
                                        end
                                        else begin
                                                request_pulse <= LOW;
```

Asynchronous Serial Data: DMX512

http://www.ti.com/lit/an/sprabr9/sprabr9.pdf

Serial Data Resources

Some useful links:

RS-232: http://www.arcelect.com/rs232.htm

UART, I2C, SPI guide (ignore the device-specific info): https://tessel.io/blog/108840925797/a-web-developers-guide-to-communication-protocols

USB: http://www.beyondlogic.org/usbnutshell/usb1.shtml

DMX512: http://www.elationlighting.com/pdffiles/dmx-101-handbook.pdf

http://www.ti.com/lit/an/sprabr9/sprabr9.pdf