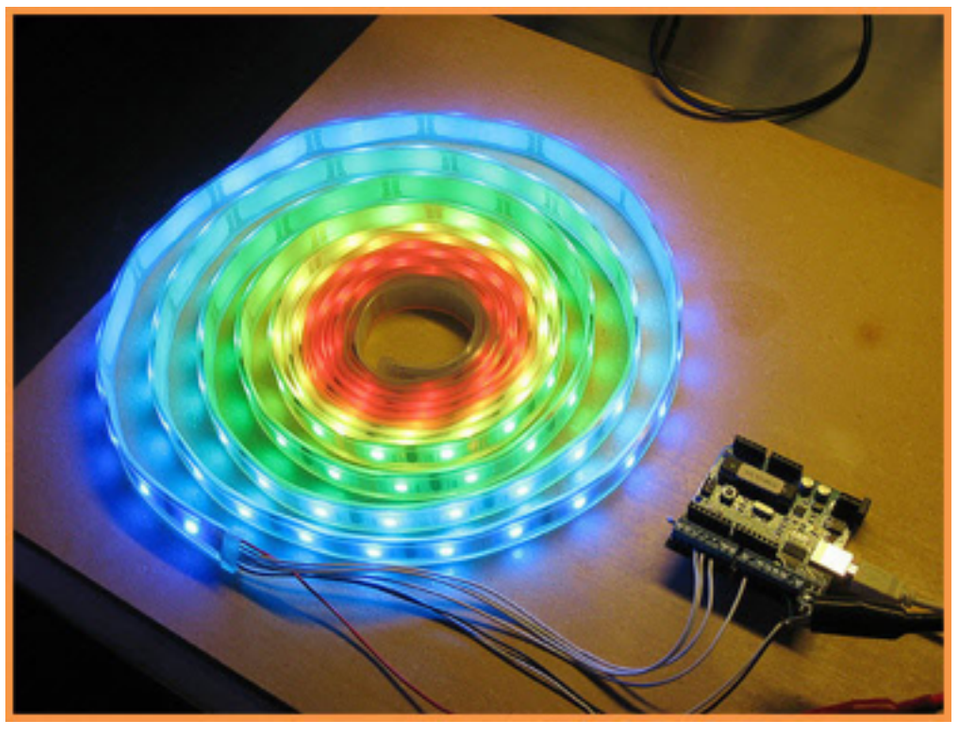
further project ideas

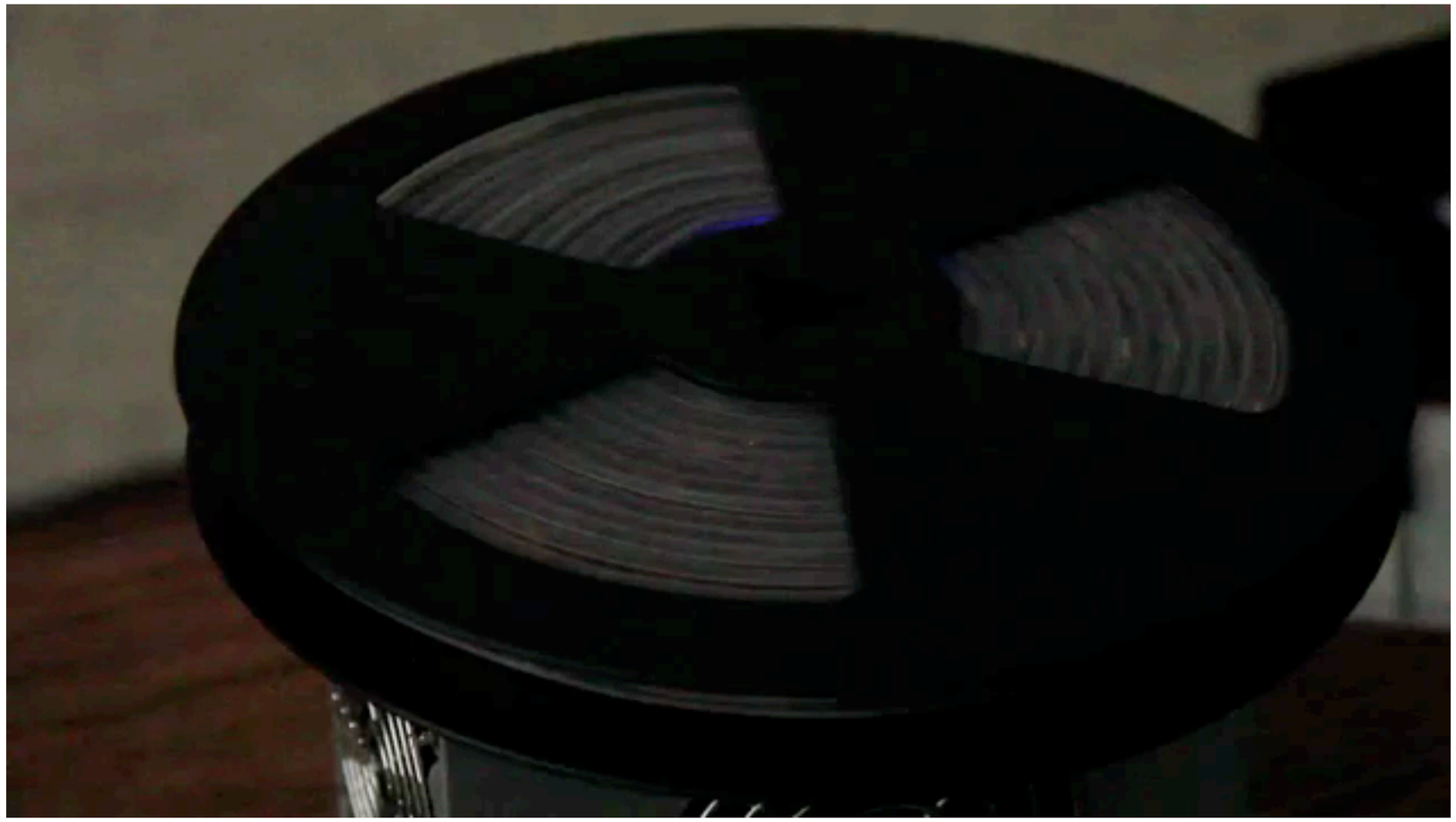
light and music

string of LEDs (RGB):

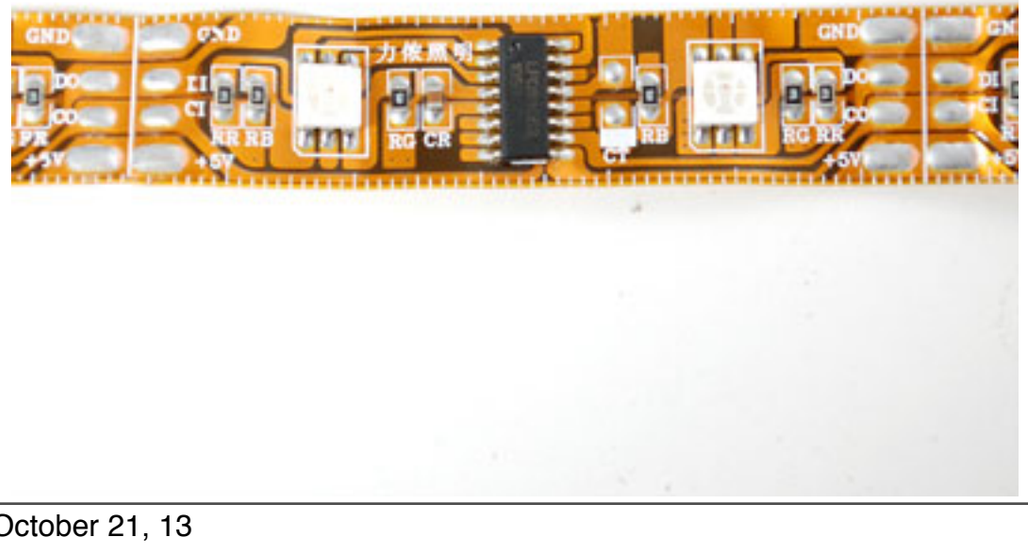
https://www.youtube.com/watch?v=8B3-OyukRb4



light and music, 2

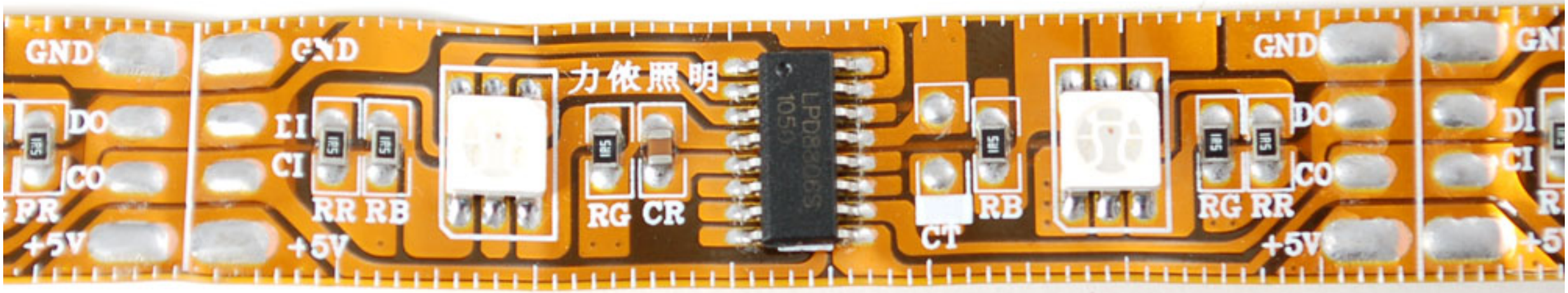


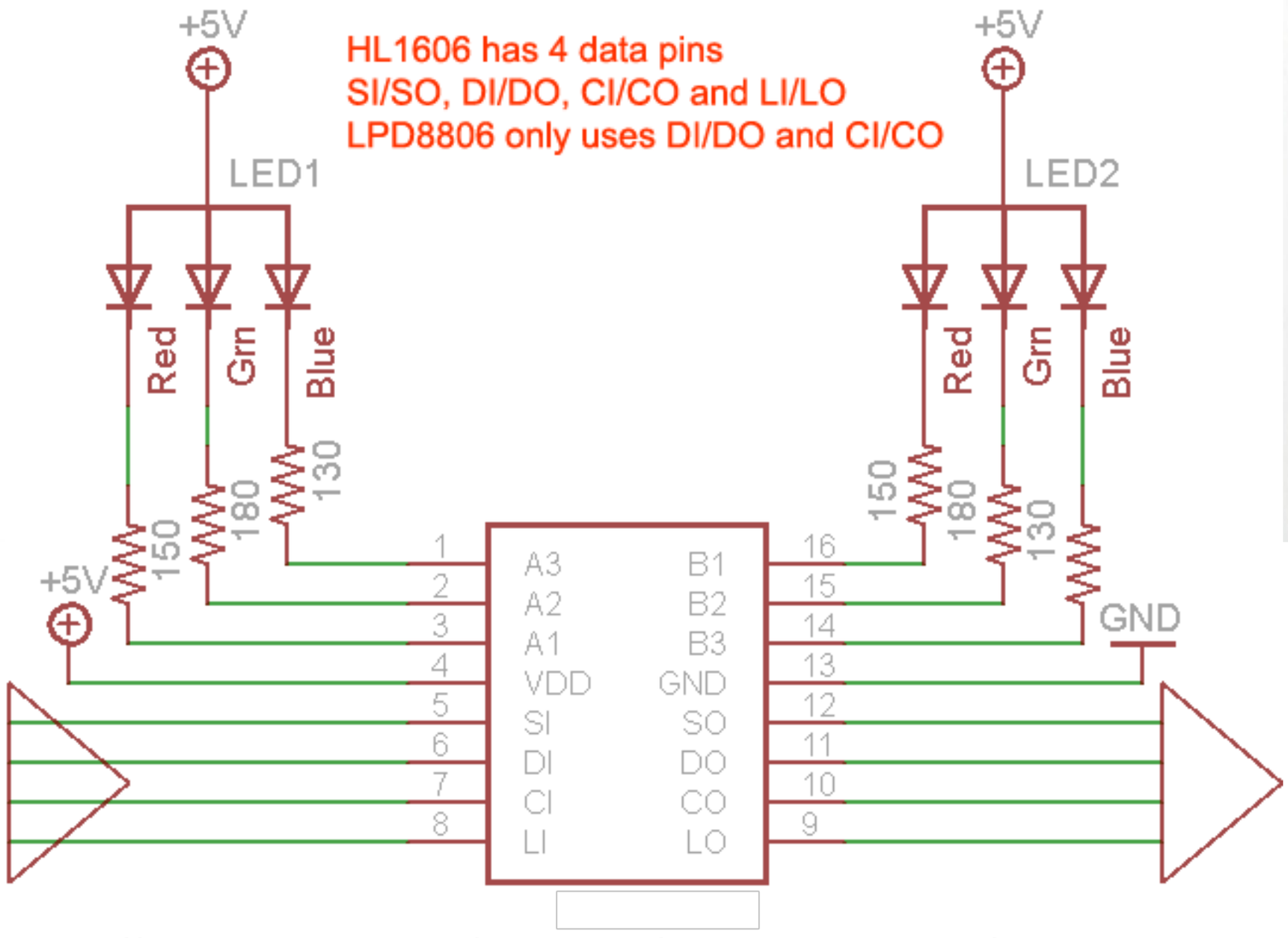
https://www.youtube.com/watch?v=2xbZTF4GZBc



← one section

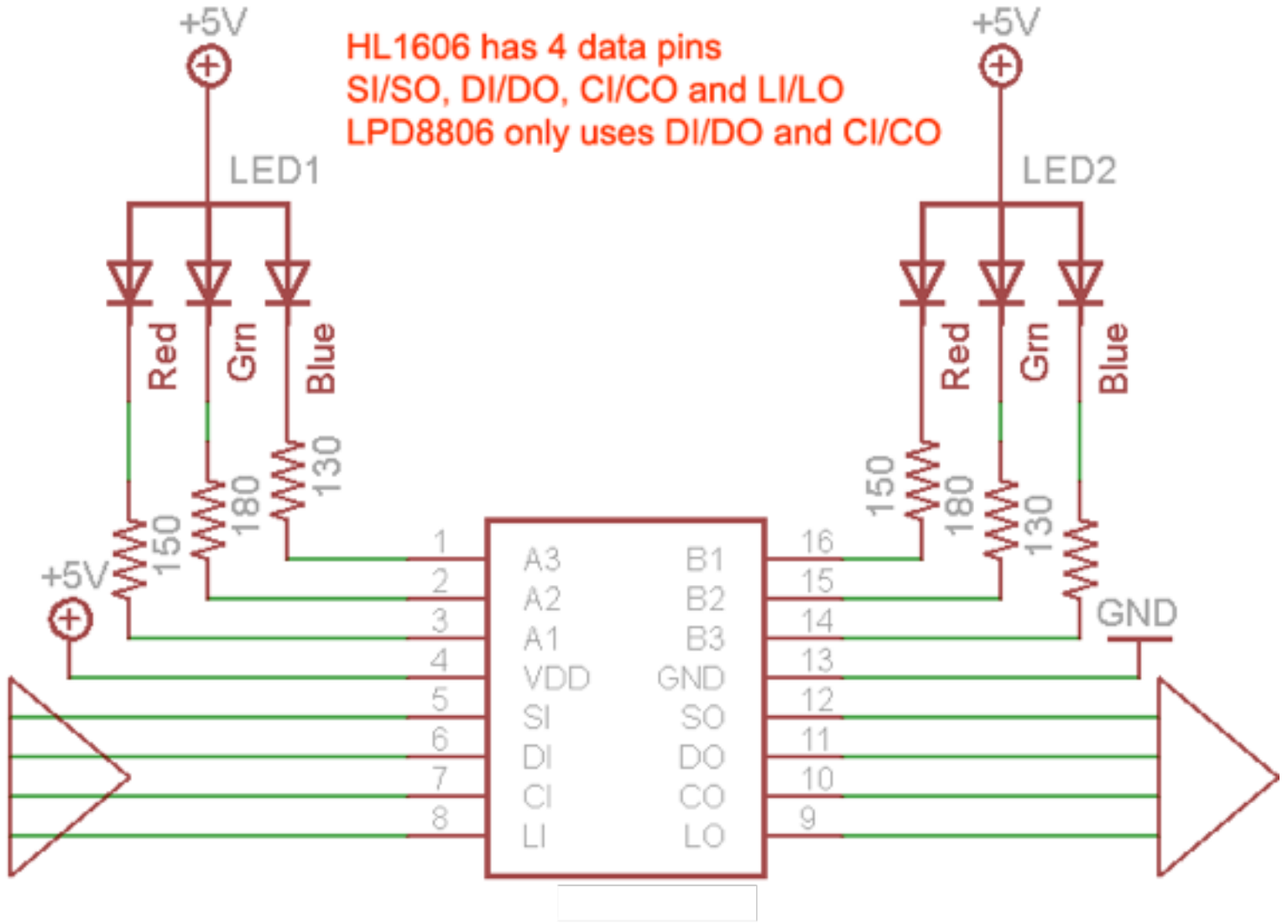
1. bit addressable
2. pwm for intensity (RGB values)





http://www.ladyada.net/products/digitalrgbledstrip/index.html

http://www.adafruit.com/products/306



bit ‘addressing’:

1. device receives RGB values
2. can keep or push down line
3. latch to display current value

to change one pixel, all must be updated

```
/* initalize the library */

LPD8806::LPD8806(uint16_t n) {
   numLEDs = n;

   // malloc per pixel so we dont have to hardcode the length
   pixels = (uint32_t *)malloc(numLEDs);
}
```

we send this serially to LED strip

```
/* setup a strip instance of n pixel length */

void LPD8806::begin(void) {
   // initialize the SPI bus
   // the strip uses default settings (mode=0, msbfirst), so we don't
need to override anything
   SPI.begin();
```

synchronous serial interface

```
   // run the SPI bus as fast as possible
   // this has been tested on a 16MHz Arduino Uno
   SPI.setClockDivider(SPI_CLOCK_DIV2);


   // clear the strip on startup
   // remember, even if we reset the controller, the LPD8806 chips hold
state until told otherwise
   clear();
}
```

https://github.com/cjbaar/LPD8806

```
/* recall the number of pixels */

uint16_t LPD8806::numPixels(void) {
   return numLEDs;
}


/* create a 3-byte color string from individual r,g,b values */

uint32_t LPD8806::Color(byte r, byte g, byte b) {
   //Take the lowest 7 bits of each value and append them end to
end
   // We have the top bit set high (its a 'parity-like' bit in
the protocol
   // and must be set!)

   // (the LPD8806 wants the order to be green, red, blue)

   uint32_t x;
   x = g | 0x80;
   x <<= 8;
   x |= r | 0x80;
   x <<= 8;
   x |= b | 0x80;

   return(x);
}
```

need 24-bits to represent RGB value

(have to set intensity for each LED at pixel)

```
void LPD8806::show(void) {
   uint16_t i;

   // get the strip's attention
   write8(0);
   write8(0);
   write8(0);
   write8(0);

   // write 24 bits per pixel
   // LPD8806 order is g,r,b
   for (i=0; i<numLEDs; i++ ) {
      write8(pixels[i]>>16 & 0xff);
      write8(pixels[i]>>8 & 0xff);
      write8(pixels[i] & 0xff);
   }

   // to 'latch' the data, we send just zeros
   write8(0);
   write8(0);
   write8(0);
   write8(0);

   delay(2);
}
```

write new values to string

(serial interface requires one byte per TX)

```
/* reset the strip (write all black) */

void LPD8806::clear() {
    for (uint16_t i=0; i < numLEDs; i++) {
        setPixelColor(i, 0, 0, 0);
    }
    show();
}


/* store an rgb component in our array */

void LPD8806::setPixelColor(uint16_t n, uint8_t r, uint8_t g, uint8_t b) {
    uint32_t data;
    if (n > numLEDs) return;

    data = (g | 0x80);
    data <<= 8;
    data |= (r | 0x80);
    data <<= 8;
    data |= (b | 0x80);

    pixels[n] = data;
}


/* store a 3-byte color component in our array */

void LPD8806::setPixelColor(uint16_t n, uint32_t c) {
    uint32_t data;
    if (n > numLEDs) return;

    data = ((c>>16) | 0x80);
    data <<= 8;
    data |= ((c>>8) | 0x80);
    data <<= 8;
    data |= ((c) | 0x80);

    pixels[n] = data;
}
```

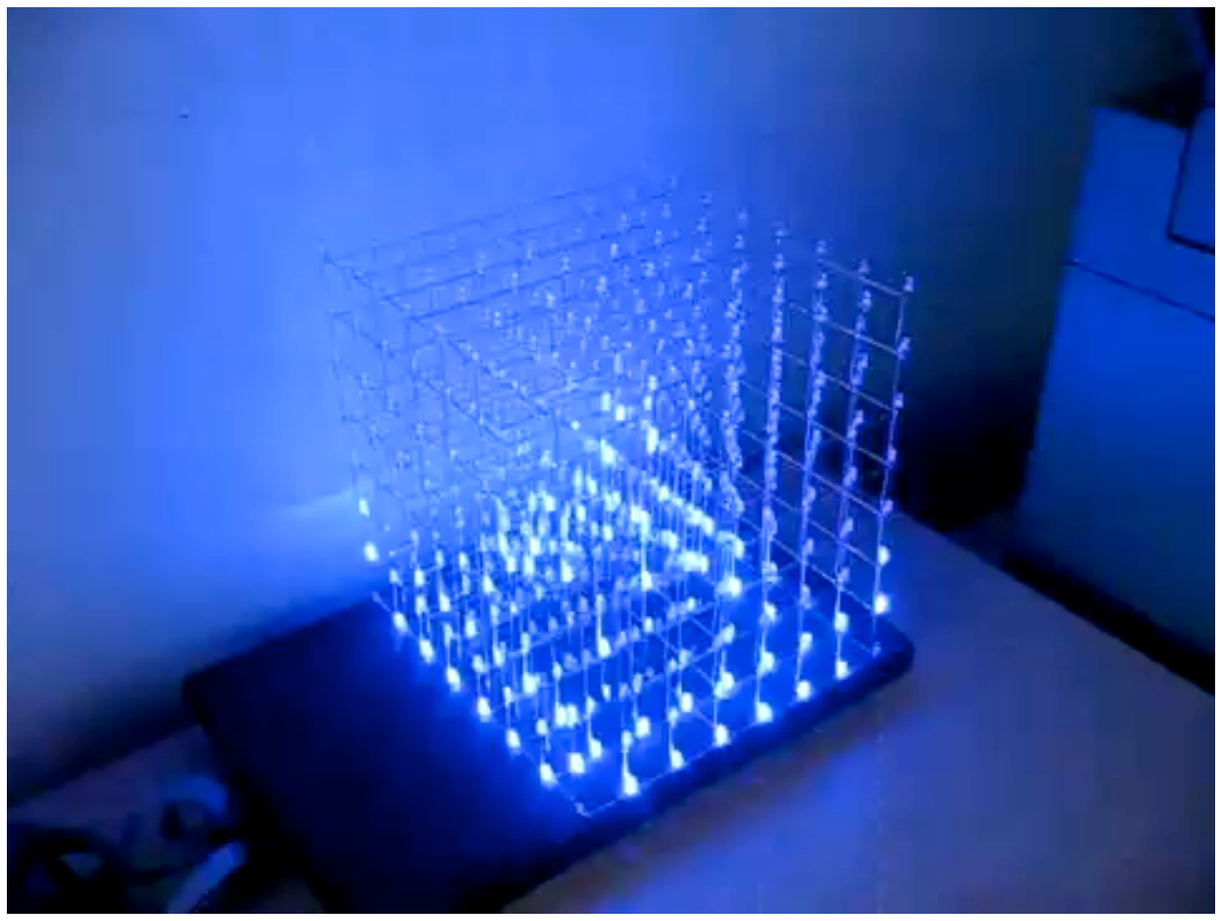
helper functions

have to write pixel array to strip for change (show())

issues:

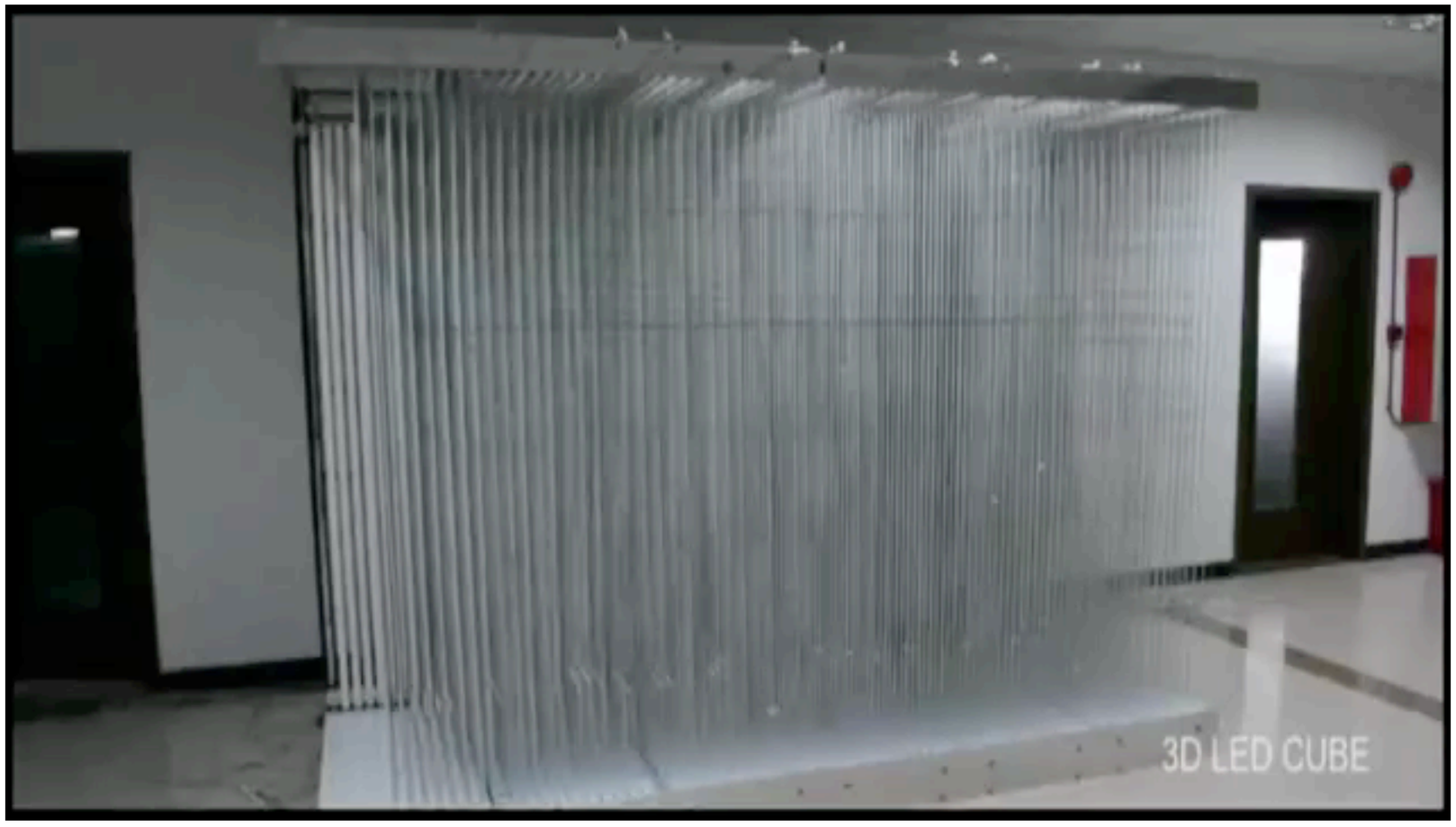
1. power (do LEDs need to be always on?)
2. throughput (how fast can we update pixels?)

make it 3D



https://www.youtube.com/watch?v=6mXM-oGggrM

more 3D



https://www.youtube.com/watch?v=dVHP7Nhsn4E

leave it to an engineer to create dancing women...sigh.

Interrupts III

ECE 3710

Sponges grow in the ocean ... that *kills* me. I wonder how much deeper the oceans would be if that didn't happen.

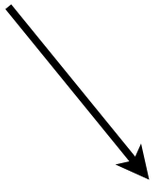
- Steven Wright

using interrupts:

1. set priorities
2. enable interrupts for peripheral

(@NVIC and @peripheral)

3. write ISR
4. wait for IRQ



5. ack IRQ in ISR

pending interrupts

(how uC manages interrupts)

assume:

1. two ISRs (*ISR_A* and *ISR_B*)
2. what if *ISR_B* is more important than *ISR_A*?

e.g.

ISR_A handles student tasks

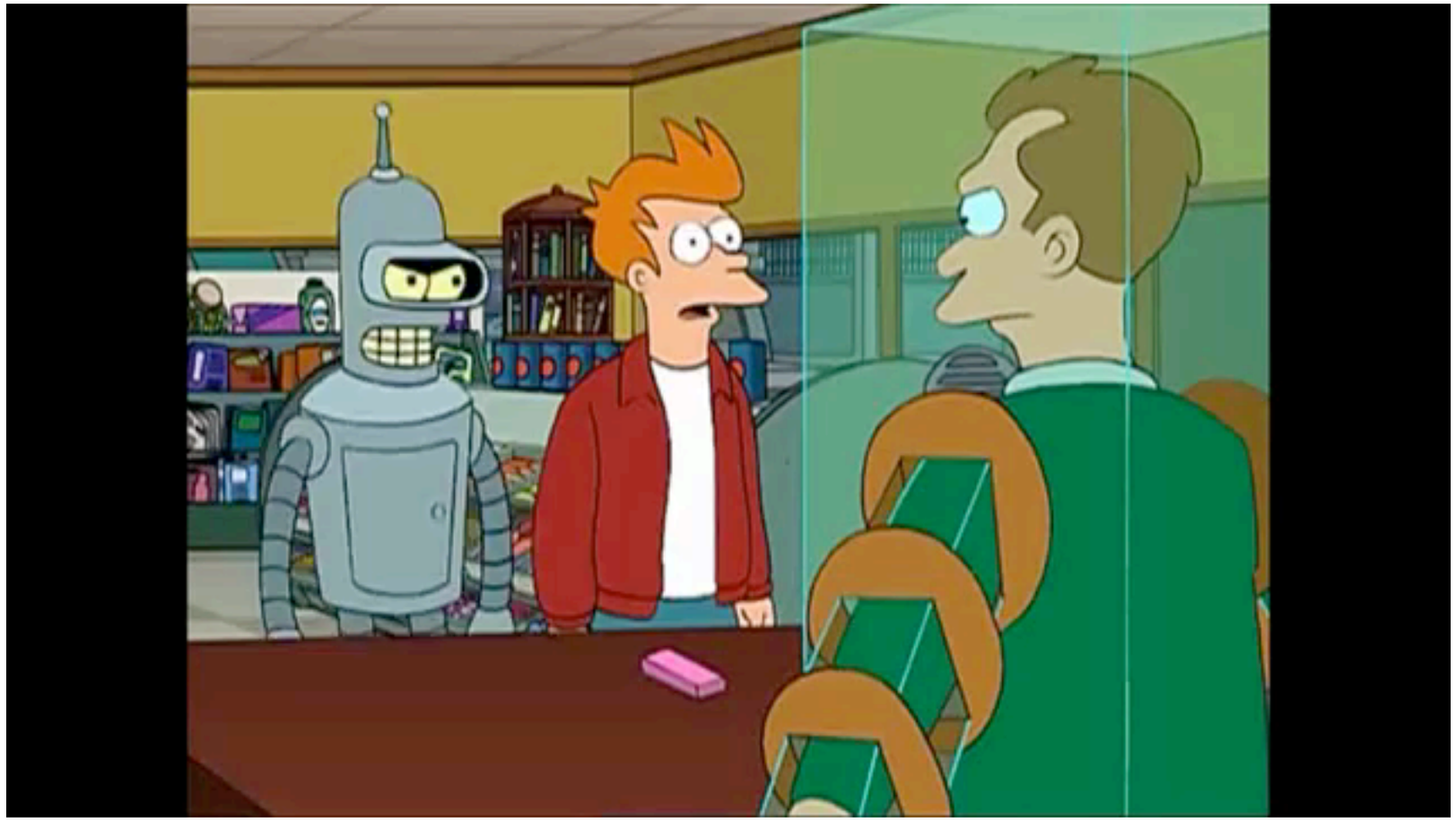
ISR_B handles professor tasks

i.e. task associated with ISR_B

if ISR_A is running while IRQ_B asserted, need to run ISR_B immediately

priority(students) < priority(professor)

student response:



pending interrupts

(how uC manages interrupts)

assume:

1. ISR_A running when
2. IRQ_B goes high

what happens depends on priorities:

1. if priority(ISR_A) >= priority(ISR_B)
2. if priority(ISR_A) < priority(ISR_B)

ISR_B begins when ISR_A ends

ISR_A halted and ISR_B begins

note: if lower priority ISR interrupt, same regs as before saved

pending interrupts

(how uC manages interrupts)

lowest

priority levels: 0--7

highest

need three bits to represent

to set:

1. set these bits to level

2. at this address

Address	31 – 29	23 – 21	15 – 13	7 – 5	Name
0xE000E400	GPIO Port D	GPIO Port C	GPIO Port B	GPIO Port A	NVIC_PRI0_R
0xE000E404	SSI0, Rx Tx	UART1, Rx Tx	UART0, Rx Tx	GPIO Port E	NVIC_PRI1_R
0xE000E408	PWM Gen 1	PWM Gen 0	PWM Fault	I2C0	NVIC_PRI2_R
0xE000E40C	ADC Seq 1	ADC Seq 0	Quad Encoder	PWM Gen 2	NVIC_PRI3_R
0xE000E410	Timer 0A	Watchdog	ADC Seq 3	ADC Seq 2	NVIC_PRI4_R
0xE000E414	Timer 2A	Timer 1B	Timer 1A	Timer 0B	NVIC_PRI5_R
0xE000E418	Comp 2	Comp 1	Comp 0	Timer 2B	NVIC_PRI6_R
0xE000E41C	GPIO Port G	GPIO Port F	Flash Control	System Control	NVIC_PRI7_R
0xE000E420	Timer 3A	SSI1, Rx Tx	UART2, Rx Tx	GPIO Port H	NVIC_PRI8_R
0xE000E424	CAN0	Quad Encoder 1	I2C1	Timer 3B	NVIC_PRI9_R
0xE000E428	Hibernate	Ethernet	CAN2	CAN1	NVIC_PRI10_R
0xE000E42C	uDMA Error	uDMA Soft Tfr	PWM Gen 3	USB0	NVIC_PRI11_R
0xE000ED20	SysTick	PendSV	--	Debug	NVIC_SYS_PRI3_R

3. for interrupt source

ex: timer0a ISR vs. timer0b ISR

priority(B) > priority(A)

p108

```
/* NVIC setup */
// 1. enable interrupts for timer0a/b: second byte of EN0 register
M3CP[0x102] = 0x18; //0x18 = 0b00011000
```

p118

```
// 2. set priorities: timer0a=2, timer0b=1 (timer0b has greater priority)
//timer0a priority is set in last byte of PRI4
M3CP[0x413] = 0x40; //0x40=010000000
//timer0b priority is set in first byte of PRI5
M3CP[0x414] = 0x20; //0x20=001000000
```

p353

```
/* timer setup */
// 5. enable interrupts
TM0[0x18] = 0x1;
TM0[0x19] = 0x1;
```

TM0A finishes before TM0B

```
// 4. set initial value
TM0A_INIT = 0xF;
TM0B_INIT = 0x15;
```

ex: timer0a ISR vs. timer0b ISR

priority(B) > priority(A)

```
// not as important stuff happening here
void Timer0A_Handler(void)
{
    unsigned int i,j;

    // let timer know interrupt has been handled
    TM0[0x24] = 0x1;

    // have to do something in loop, or compiler removes it...
    for(i=0;i<10;i++)
        j++;
}
```

p357

```
                    // important stuff happening here
                    void Timer0B_Handler(void)
                    {
                        unsigned int i,j;
                        // let timer know interrupt has been handled
                        TM0[0x25] = 0x1;

                        // have to do something in loop, or compiler remov
                        for(i=0;i<5;i++)
                            j++;
                    }
```

p353

interrupt latency:

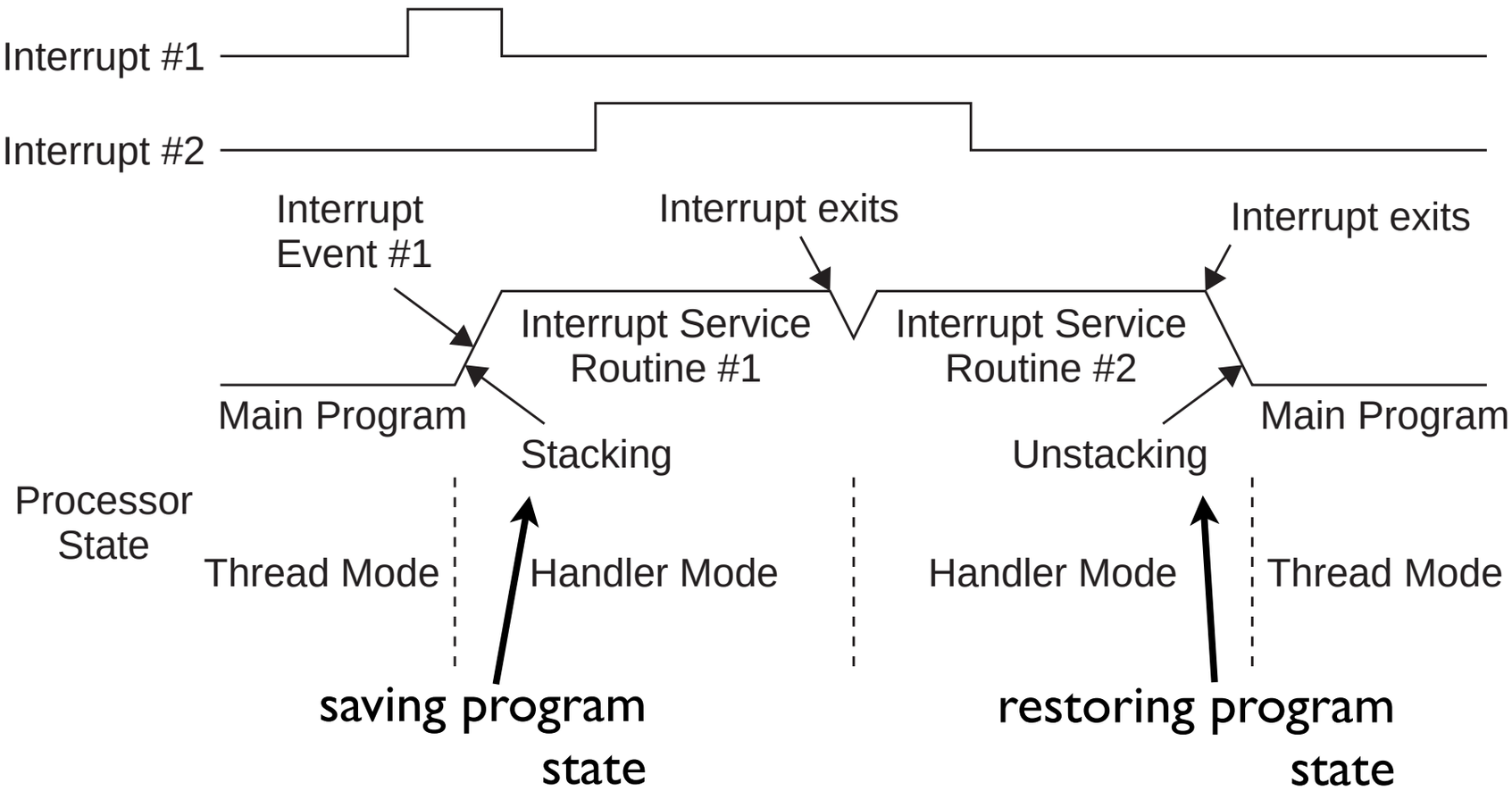
time between interrupt occurring and it being serviced

solutions
(for uC)

1. tail chaining
2. late arrivals

interrupt latency: tail chaining

scenario: IRQ2 raised during ISR1 → ISR2 will be run after ISR1 finishes



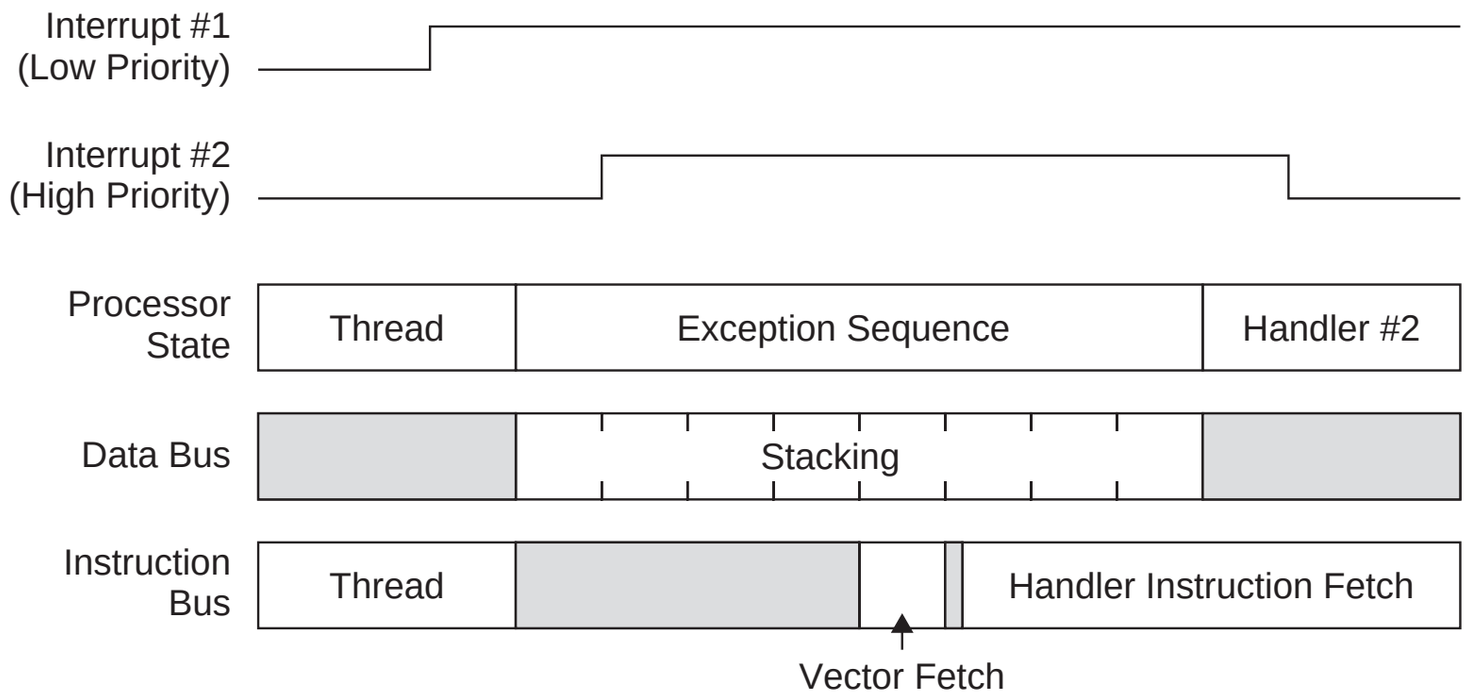
behaviour: don't restore program context (state) between ISR1 end and ISR2 begin

interrupt latency: late arrivals

scenario: IRQ1 raised; before entering ISR1, IRQ2 raised

priority(INT1)<priority(INT2)

ISR1 will be pre-empted by ISR2



behaviour: after context save, execute ISR2 and use tail chaining

what can trigger interrupt?

example

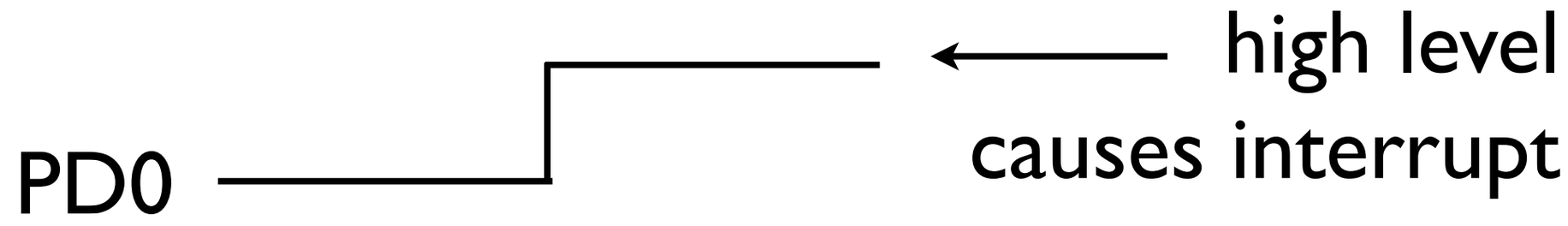
we'll consider:

1. timer expiration (SysTick &GPTM)
2. peripherals (UART)
3. external events (GPIO)

today

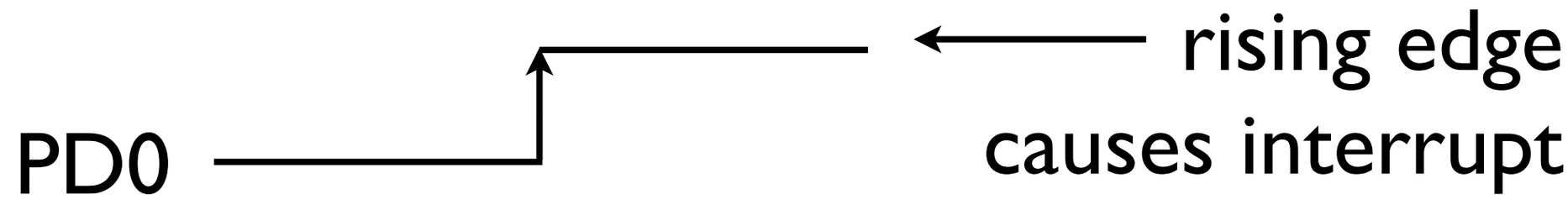
how external interrupts are recognised

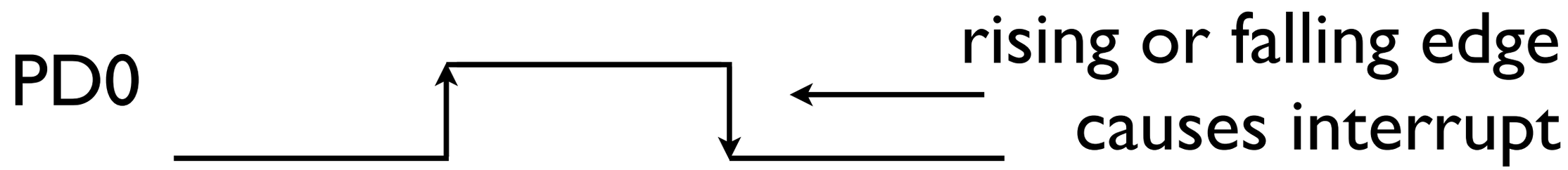
1. level change



how external interrupts are recognised

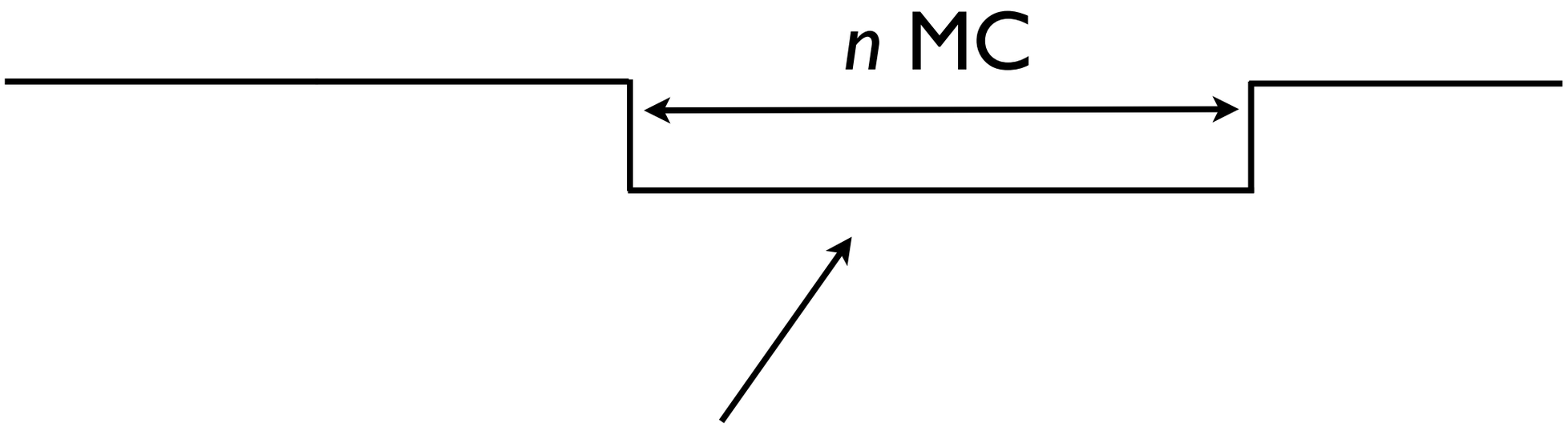
2. transitions





how external interrupts are recognised: timing

low level triggering:



input must remain low this long for IRQ to recognised

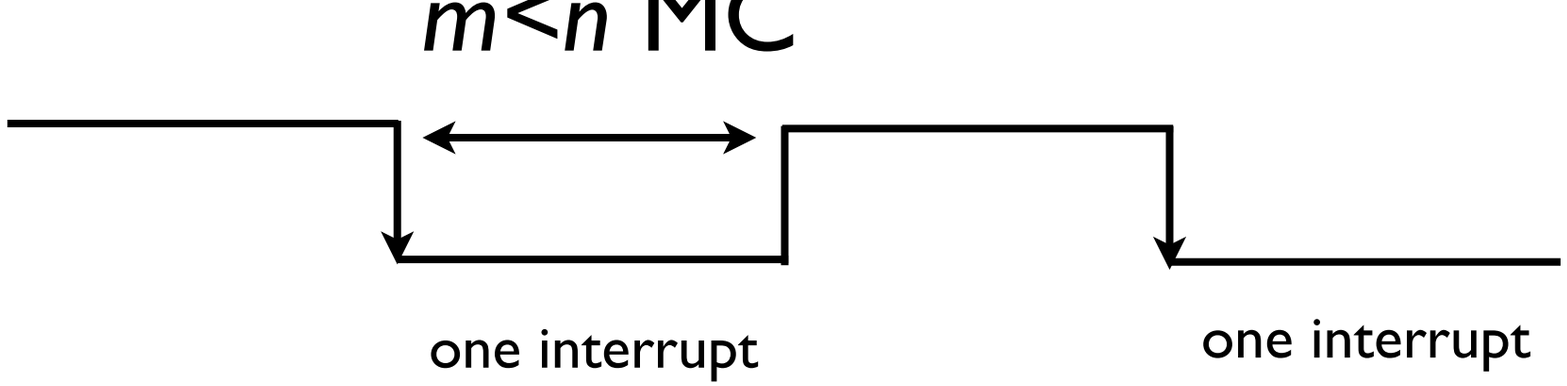
time between interrupts greater for level than edge

how external interrupts are recognised: timing

transitions are preferred for
reasons of timing:

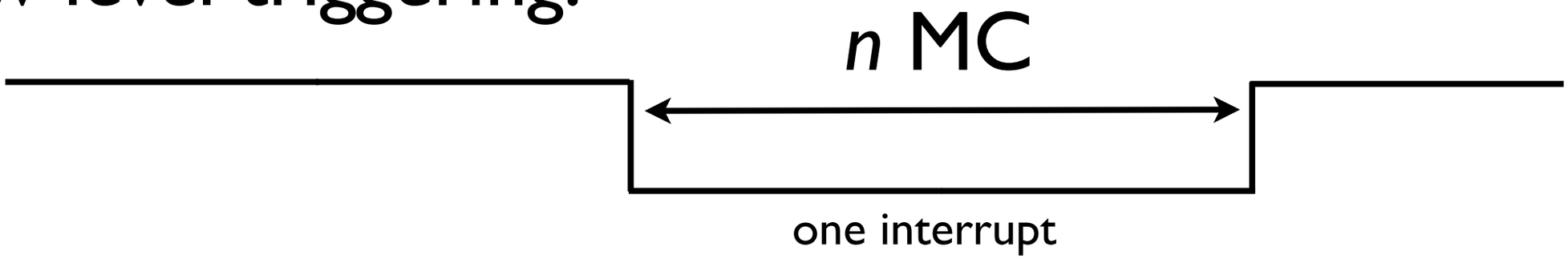
:negative edge triggering

m<*n* MC



low level triggering:

n MC



edge vs. level triggering

note:

1. edge triggering: ISR must clear/acknowledge interrupt for another to occur
2. level triggering: ISR needn't acknowledge
   (device must hold line level for long enough for recognition)

p292

external interrupts:

1. select port/pin and configure as input
2. choose level or edge
3. select level or edge type
(high/low; rising/falling/both)
4. enable interrupt for pin
5. enable interrupt for port

GPIOIS
0x404,p300

GPIOIEV
0x40C,p302

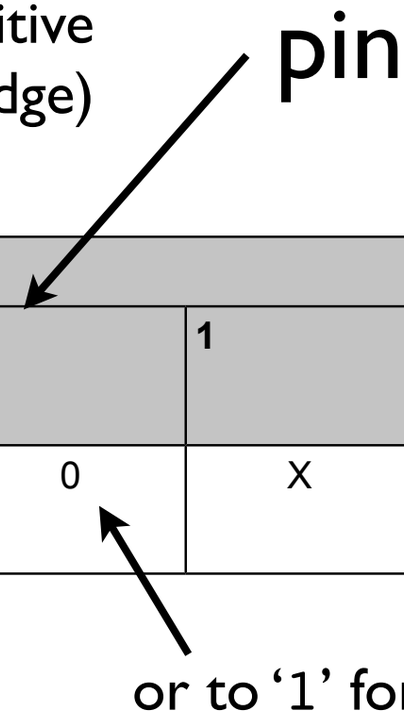
GPIOIBE
0x408,p301

GPIOIM
0x410,p303

to tell GPIO interrupt has been serviced —set bit→ GPIOICR
0x41C,p306

example:Port X, pin 2

(enable interrupts with edge triggering, positive edge)



Register	Desired Interrupt Event Trigger	Pin 2 Bit Value[a]							
		7	6	5	4	3	2	1	0
GPIOIS	0=edge 1=level	X	X	X	X	X	0	X	X
GPIOIBE	0=single edge 1=both edges	X	X	X	X	X	0	X	X
GPIOIEV	0=Low level, or negative edge 1=High level, or positive edge	X	X	X	X	X	1	X	X
GPIOIM	0=masked 1=not masked	0	0	0	0	0	1	0	0

a. X=Ignored (don't care bit)

ex: PD0, positive, edge-triggering

```
void PD0Init()
{
    ...
    // 2. edge triggering
    PD[0x404] = 0;

    // 3. rising edge
    PD[0x408] = 0; //don't want both edges
    PD[0x40C] = 1; //rising edge

    // 4. enable pin interrupt
    PD[0x410] = 1;
    ...
}
```

GPIOIS
0x404,p300

GPIOIBE
0x408,p301

GPIOIEV
0x40C,p302

GPIOIM
0x410,p303

GPIOICR
0x41C,p306

```
/* one interrupt handler per port */
void GPIOPortD_Handler(void)
{
    // acknowledge interrupt
    PD[0x41C] = 1;

    CNT++;
}
```

ex: alarm system monitor

if sensors trip they output 0, do:

1. trigger alarm
2. lock doors
3. execute notification routine

connected to `PD0`

(many sensors share same pin: open drain)

someone else writes these

(`intruder(unsigned char intPresent)`)

1: intruder present
(do 1--3)
0: no intruder
(reverse 1--3)