```
#ifndef cbi
#define sbi(port,pin) port |= _BV(pin)
#define cbi(port,pin) port &= ~_BV(pin)
#endif

#// pins on portd
#define active_pin 5
#define clock_pin 2  // also INT0
#define D1_PIN 3  // flash data, also INT1
#define D2_PIN 4  // camera data

//*****************************************************************
#define BUFFER_SIZE 100

class ETTL_buf {
public:
  // the array contains 8 pairs of {D2,D1} bits
  // queue<15> = D2<7>, queue<14> = D1<7>, queue<13> = D2<6>, etc.
  volatile uint16_t queue[BUFFER_SIZE];

  volatile uint8_t write_index;
  volatile uint8_t read_index;

  inline void add(uint16_t word) {
    queue[write_index++] = word;
    if (write_index == BUFFER_SIZE) write_index = 0;
  }

  ETTL_buf() {read_index = 0; write_index = 0;}

  inline unsigned char isavail() {return read_index != write_index;}

  inline uint16_t get() {
    uint16_t d = queue[read_index++];
    if (read_index == BUFFER_SIZE) read_index = 0;
    return d;
  }
} ettl_buf;

//*****************************************************************
void  printnibble(unsigned char d) {
  if (d < 10) Serial.print(d+'0', BYTE);
  else Serial.print(d - 10 + 'A', BYTE);
}
void  printhex(unsigned char d) {
  printnibble(d >> 4);
  printnibble(d & 0xf);
}
void printhex16 (uint16_t d) {
  printhex((unsigned char) ((d >> 8) & 0xff));
  printhex((unsigned char) d & 0xff);
}

//*****************************************************************
// rising edge interrupts on ettl clock for first bit
ISR(INT0_vect) {
  uint16_t new_word;
  uint8_t bits = 0;

  // process the byte
  while(1) {
    // same both D2 & D1; normalize to bits <1:0> and mask
    uint8_t pind = (PIND >> D1_PIN) & 0x3;  // sample pins

    // accumulate bits from MSB to LSB
    new_word = ( new_word << 2) | pind;  // add into the sampled word
```

```
    // count the bits we've processed
    bits++;
    if (bits == 8) {
      // done with the byte
      ettl_buf.add(new_word);  // add to queue

      EIMSK = 0x02;  // reenable int1, disable int0
      EIFR = 0x03;  // neither interrupt pending
      return;
    }

    // need more bits
    // look for a new clock
    // 1st, look for clock to drop
    uint8_t count = 0xff;
    while((PIND & _BV(clock_pin)) != 0) {
      count--;
      if (count == 0) {
        // too long for a high
        // false start after preflash or flash clk edge
        EIMSK = 0x02;  // reenable int1, disable int0
        EIFR = 0x03;  // neither interrupt pending
        return;
      }
    }

    // wait for clock to rise
    while((PIND & _BV(clock_pin)) == 0);

    // process the next bit
  }
}

// D1 rising interrupt
// Flash raises D1 to indicate its ready for new data
ISR(INT1_vect) {
  EIMSK = 0x01;  // enable int0, disable int1
  EIFR = 0x03;  // neither interrupt pending
}

//*****************************************************************
void setup() {
  Serial.begin(115200);
  Serial.println("start");
  cbi(TIMSK0, TOIE0);  // disable timer0 interrupts

  // setup the pin interrupts
  EICRA = 0x0F;  // rising edge for INT0, rising edge for INT1
  EIFR = 0x03;  // neither interrupt
  EIMSK = 0x02;  // enable int1
}


//*****************************************************************
#define RESPONSE_SIZE 20

void loop() {
  // declare array to hold the flash response bytes
  uint8_t flash_response[RESPONSE_SIZE];
  uint8_t flash_response_count = 0;

  while(1) {
    // check if new communication between camera/flash
    if (ettl_buf.isavail()) {
      // get the pack 16-bit value
      uint16_t word = ettl_buf.get();
```

```
      // unpack to the separate D1, D2 pieces
      uint8_t d1;
      uint8_t d2;

      for (uint8_t i = 0; i < 8; i++) {
        d1 = (d1 << 1) | ((word & 0x4000) != 0);
        d2 = (d2 << 1) | ((word & 0x8000) != 0);
        word = (word << 2);
      }

      // save the flash response for later
      flash_response[flash_response_count++] = d1;
      if (flash_response_count >= RESPONSE_SIZE) {
        Serial.println("response overflow");
        while(1);  // lockup - something went wrong!!!
      }

      // check if a new camera command
      if ((d2 & 0xb0) == 0xb0 && (d2 != 0xff)) {
        // print accumulated responses from previous command
        Serial.print("- ");
        for (unsigned i = 0; i < flash_response_count; i++) {
          printhex(flash_response[i]);
          Serial.print(" ");
        }
        flash_response_count = 0;
        Serial.println();  // start new line
      }

      // print new command or subsequent commanddata byte from camera
      printhex(d2);
      Serial.print(" ");
    }
  }

}
```