

Keskeytykset


Keskeytykset

- Miksi käyttää keskeytyksiä?
 - Typistetysti: niiden ansiosta esim. Painonapin painallusta ei tarvitse odottaa missään loopissa tai if rakenteessa (eli ei tarvitse pollata), vaan ohjelma voidaan pitää ajossa ja kun nappia on painettu siirrytään keskeytys käsittelijään, harjoitustyössä ButtonHandleriin.

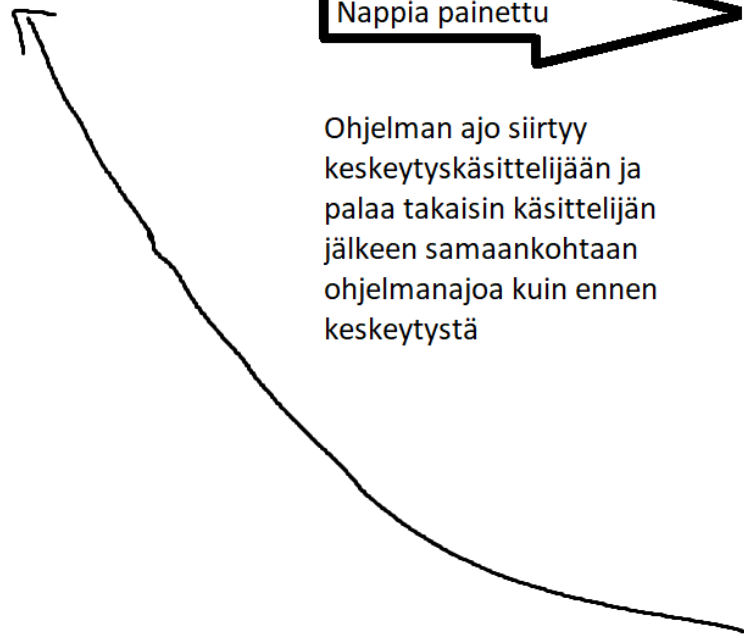
Painonappi keskeytykset

```
//Main loop
while(1){
```

Nappia painettu



Ohjelman ajo siirtyy keskeytyskäsittelijään ja palaa takaisin käsittelijän jälkeen samaankohtaan ohjelmanajoa kuin ennen keskeytystä



```
//Interrupt handler for switches and buttons. Connected buttons and switches are at bank2. Reading Status will tell which button or switch was used
void ButtonHandler(void *CallBackRef, u32 Bank, u32 Status)
{
    //****Write code here ****

    //BTN0 is clicked
    if(Status==0x01) {

    }

    //BTN1 is clicked
    else if(Status==0x02){

    }
    //BTN2 is clicked
    else if(Status==0x04){

    }
    //BTN3 is clicked
    else if(Status==0x08){

    }

    //SW0 position is changed. 0xE000A068 address needs to be read if you want to know in which position slider is
    else if(Status==0x10){

    }

    //SW1 position is changed. 0xE000A068 address needs to be read if you want to know in which position slider is
    else if (Status==0x20){

    }

    //****END OF OWN CODE*****
}
```

Ajastinkeskeyty

```
//Main loop
while(1){
```

1.25ms välein

Ohjelman ajo siirtyy keskeytyskäsittelijään ja palaa takaisin käsittelijän jälkeen samaankohtaan ohjelmanaajoa kuin ennen keskeytystä

```
//Timer interrupt handler for led matrix update. Frequency is 800Hz
void TickHandler(void *CallBackRef){
    //Don't remove this
    uint32_t StatusEvent;

    //exceptions must be disabled when updating screen
    Xil_ExceptionDisable();

    /**Write code here **/
}
```

```
/**END OF OWN CODE**/

//*****clear timer interrupt status. DO NOT REMOVE*****
StatusEvent = XTtcPs_GetInterruptStatus((XTtcPs *)CallBackRef);
XTtcPs_ClearInterruptStatus((XTtcPs *)CallBackRef, StatusEvent);
//*****

//enable exceptions
Xil_ExceptionEnable();
}
```

Ajastinkeskeyty

```
//Main loop
while(1){
```

0.1s välein

Ohjelman ajo siirtyy keskeytyskäsittelijään ja palaa takaisin käsittelijän jälkeen samaankohtaan ohjelmanajoa kuin ennen keskeytystä

```
//Timer interrupt for moving alien, shooting... Frequency is 10Hz by default
void TickHandler1(void *CallBackRef){
```

```
//Don't remove this
uint32_t StatusEvent;
```

```
****Write code here ****
```

```
*****END OF OWN CODE*****
```

```
//clear timer interrupt status. DO NOT REMOVE
```

```
StatusEvent = XTtcPs_GetInterruptStatus((XTtcPs *)CallBackRef);
XTtcPs_ClearInterruptStatus((XTtcPs *)CallBackRef, StatusEvent);
```

Keskeytykset

```
int main()
{
    /**INITS, DO NOT TOUCH***/
    init_platform();
    *leds=0;
    *rgbleds=0;
    init_interrupts();

    //setup screen
    setup();

    //initial pixels to screen
    draw_init();

    Xil_ExceptionEnable();
    /**End of init*****

    //empty loop
    while(1)
    {

    }

    cleanup_platform();
    return 0;
}
```

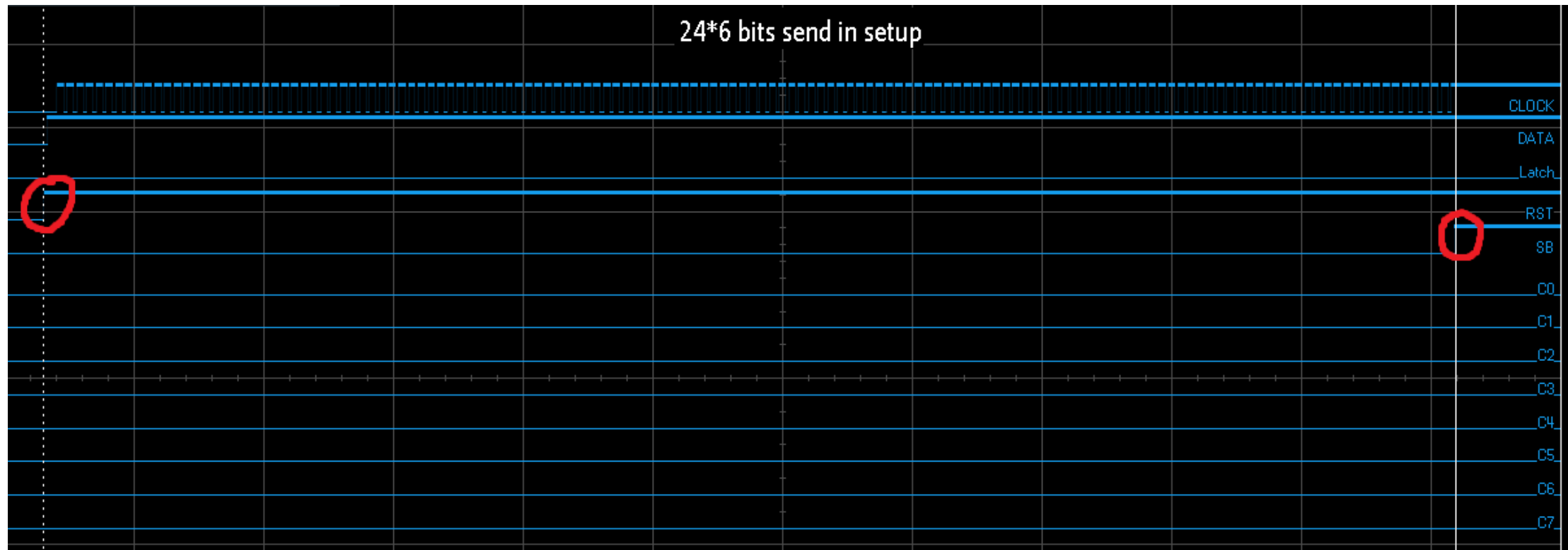
- Keskeytysten ansiosta referenssikoodin main näyttää suunnilleen vasemman kuvan mukaiselta (helppo ja vaikea)
 - leds ja rgbleds asettaminen nolleen on sinänsä turhaa koska niiden rekisterien arvot on jo 0 kun ohjelma ladataan pynq:lle

Led-matriisi

Perus termistöä harjoitustyössä

- CLK(Clock), SCK(Serial clock)=Sarjakello
- SDA(Serial Data), DATA, SIN(serial in)=datalinja
- Rst=reset
- Lat=latch

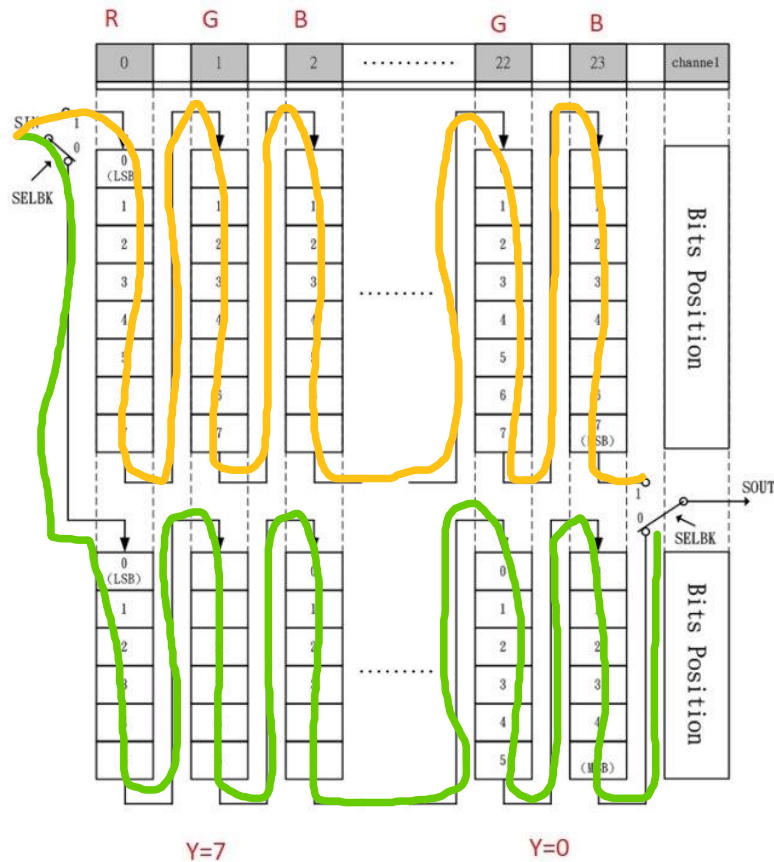
Setup



Setup

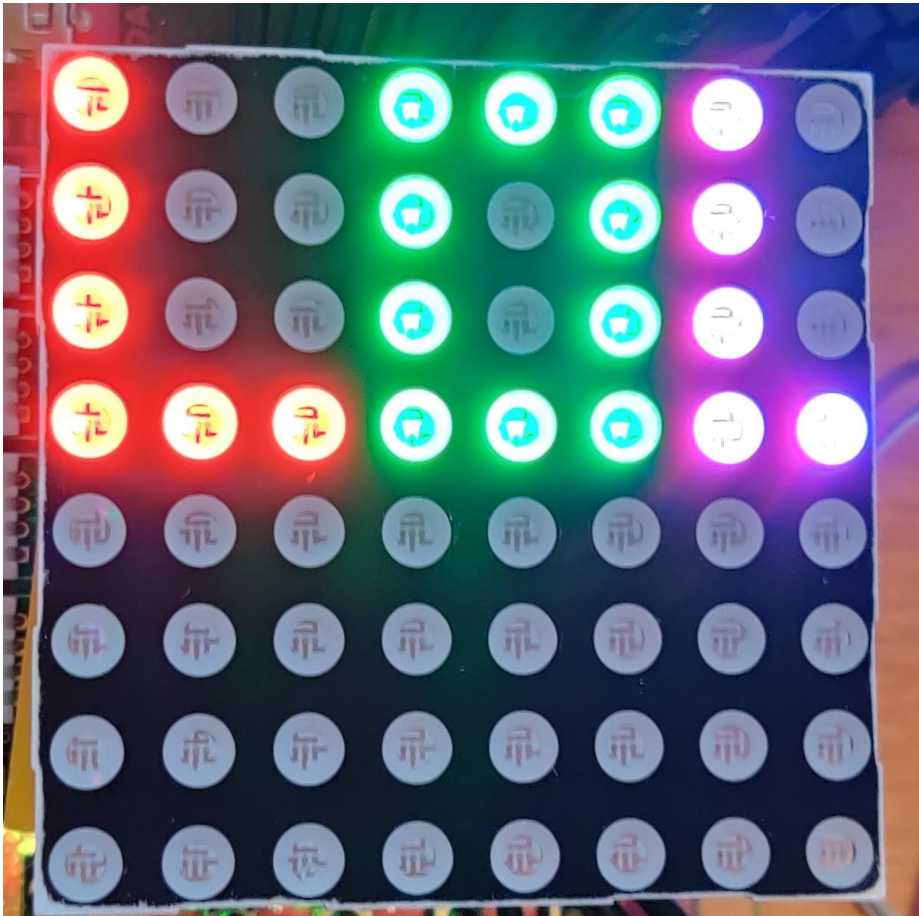
- Kuvassa miltä referenssi setup() näyttää mitattuna.
- DM163 piirille lähetetään 144kpl 1-tilassa olevia bittejä
- Saadaan aikaiseksi kun SDA linja on 1-tilassa kellon nousevalla reunalla
- Kello voidaan muodostaa for loopissa jonka ainut tehtävä on pistää kellolinja (CLK) ylös ja alas
 - Koodissa clk bit ylös ja seuraavalla rivilla alas

DM163 SB-bit

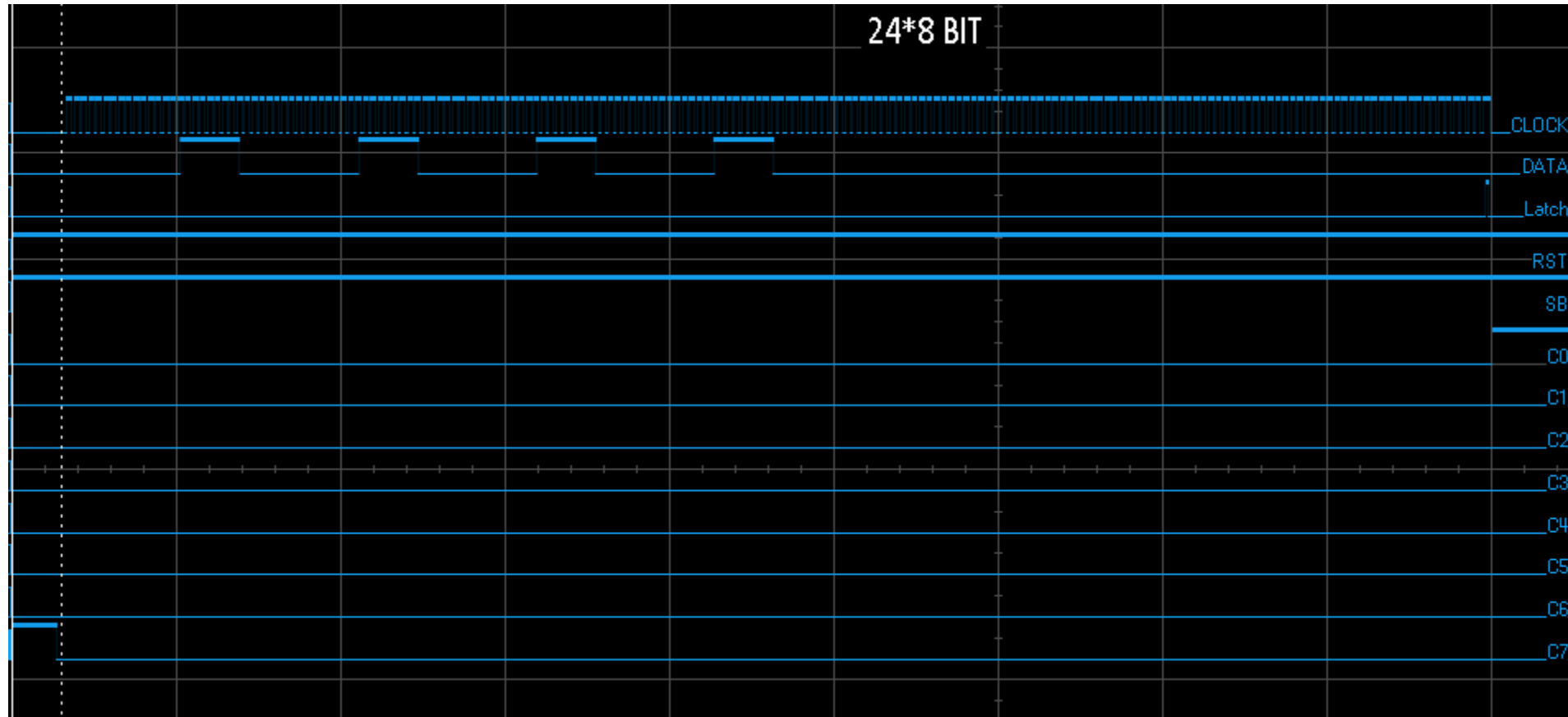


- SB=0
- SB=1
- Miksi halutaan 6-bittiseen rekisteriin kaikki bitit 1-tilaan
 - Koska ledien kirkkaus on typistetysti 6-bit reg*8-bit-reg
- Eli jos vaikka 0,1,2 kohdassa 6-bit reg kaikki bitit olisi 0 ja 8-bittisessä kaikki 1, niin Y=7 "pikseli" ei palaisi
 - Toki mikään ei pala jos yksikään channel biteistä ei ole 1

run



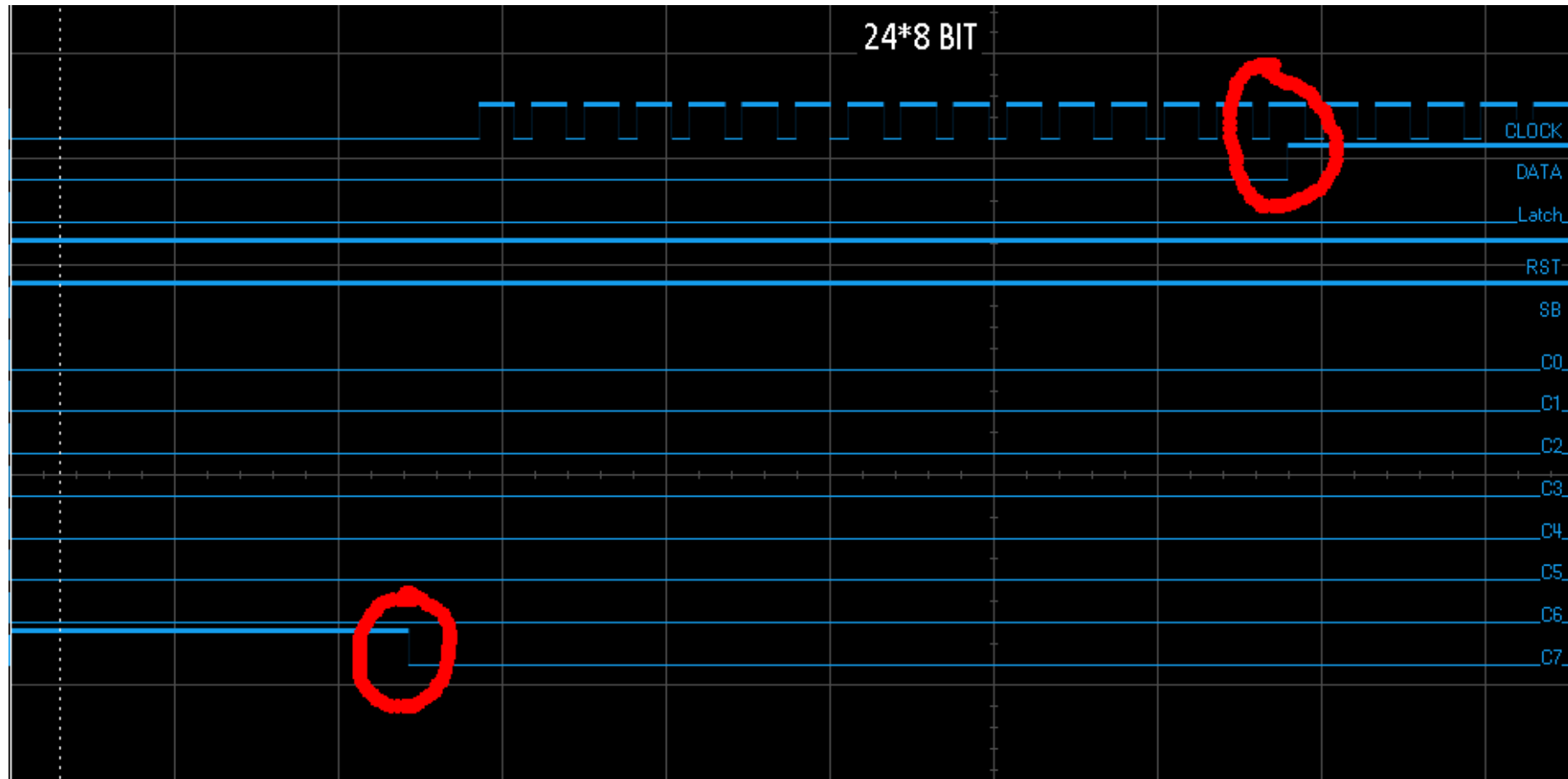
Run



run

- Kuvassa miltä referenssi `run()` näyttää mitattuna ja LOL kuvio matriisin ruudulla. Run mittaus on otettu vaiheessa jossa vasemman reunan eli x-coordinaatti 0:n data lähetetään.

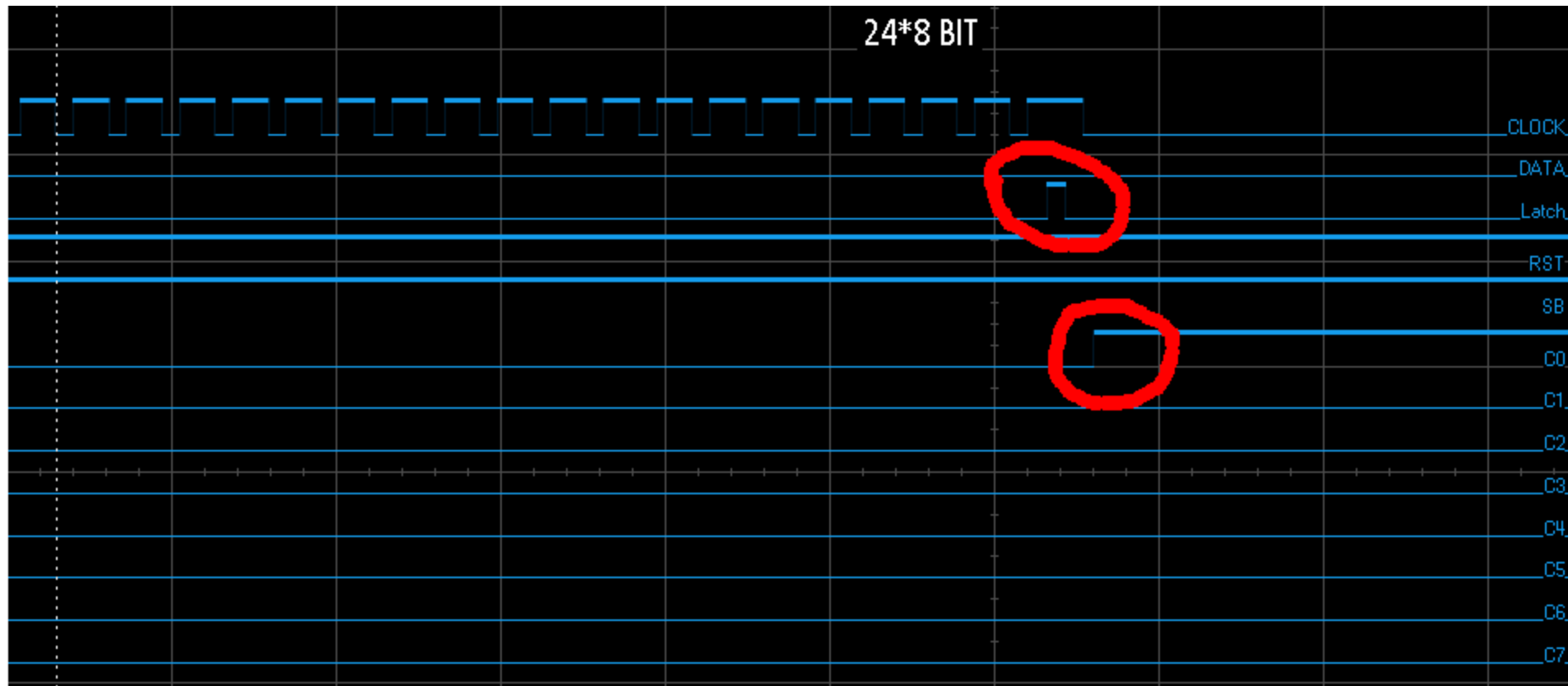
run



run

- C0-C7 ovat channel:ta. Huomatkaa missä kohtiin se nollataan.

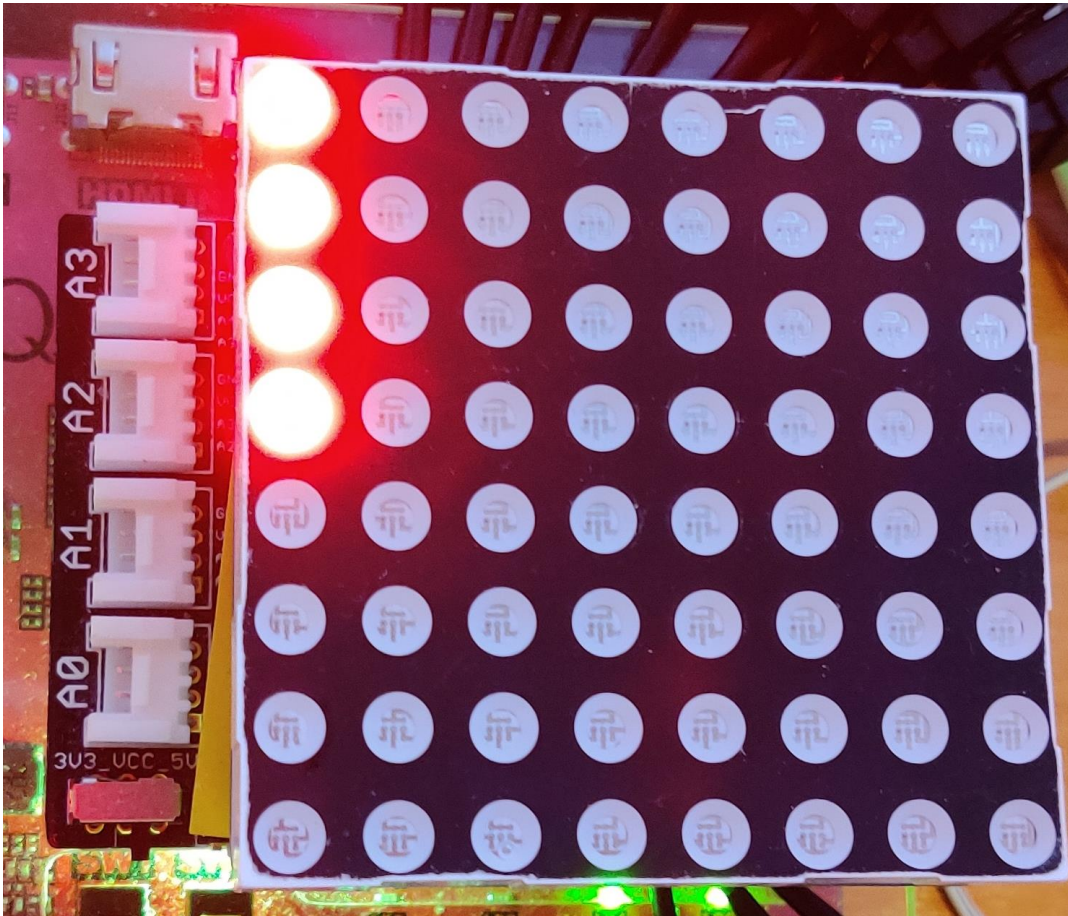
run



run

- Latch runnin lopuksi ja channel 0 päälle.
- Seuraavassa vielä esimerkkikuva miltä led-matriisi näyttää kun ajo on pysäytetty **debug breakpointilla** heti channel0 asettamisen jälkeen
- Tämänjälkeen millainen looppirakenne voisi olla hyvä

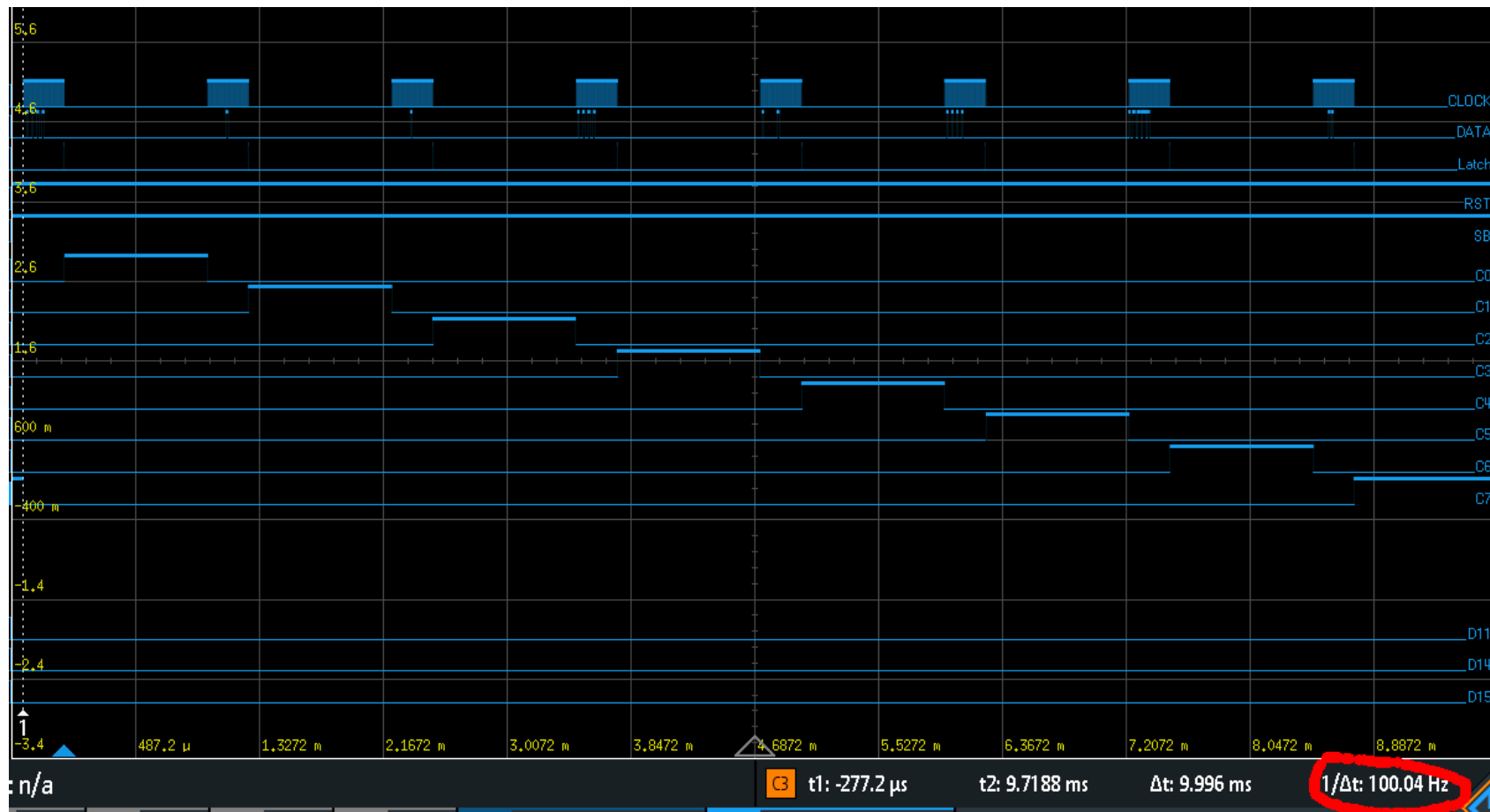
run



run

- Esimerkkinä myös mittauskuva miltä yksi koko näytön päivitystapahtuma näyttää
 - LOL kuvio edelleenkiin matriisin näytöllä

run



Run pohjaa

```
//Put new data to led matrix. Hint: This function is supposed to send 24-bytes and parameter x is for x-coordinate.  
void run(uint8_t x){  
  
    //Write code that writes data to led matrix driver (8-bit data). Use values from dots array  
  
    for(uint8_t y=0; y<8;y++){  
        for(uint8_t color=0;color<3;color++){  
            //Read dots array to some temporally variable. this temporally variable is used in sending data  
            for(uint8_t byte_count=0;byte_count<8;byte_count++){  
  
            }  
        }  
    }  
  
}
```

Setpixel (normaali versio)

- Vain kolme riviä koodia tarvitaan ja tämän tarkoitus on vain muokata dots-taulua funktionparametrien mukaan