Uitwerkingen Practicum

Microcontrollers

Mauro de Lyon & Arthur Brink

22-02-2015

Inhoudsopgave

Track 1

BEWIJS
https://www.youtube.com/watch?v=7i-rq-SOiLg

B.1 Uit de literatuur

a) *Hoe groot is het program memory van de ATmega128?*
128kB
4Kbytes + 4kB EEPROM

b) *Wat is het adres van Data direction register van PORTE (DDRE)?*
Het adres van PORTE is $03($23) en DDRE is $02($22)
$03 / $23

c) *Uit hoeveel byte bestaat de instructie 'IN R3, PORTA' ?*
Op pagina 80 van de instruction set wordt beschreven dat de IN instructie 2 bytes is.
2 bytes

d) *Hoeveel RS232 poorten zitten er op het BIGAVR6 development board?*
Op het bord zijn 2 RS232 poorten aanwezig
2

e) *Op welke pin van de microcontroller zit de ingang voor Analog digitaalconverter, channel 1?*
De ADC zit op pin 60
Pin 60

f) *Hoe groot is het data geheugen van de microcontroller maximaal?*
64 Kb
64 kB

g) *Hoeveel I/O-registers zijn er op de ATmega128?*
Op pagina 20 staat dat het bord 64 ports hebben
64

h) *De pinnen van PORTA kunnen met een weerstand naar 0 V (pull-down) of met een weerstand naar de +5V verbonden worden (pull-up). Hoe is dat standaard ingesteld op het BIGAVR6 development board?*
De standaard is Pull downs
Pull down

B.2

Maak een nieuwe applicatie die beurtelings de LED op PORTD, pin 7 (PORTD.7) en de LED op PORTD, pint 6 (PORTD.6) om de 500ms laat oplichten. Ontwikkel de applicatie in de simulator en programmeer daarna het board (gaat veel sneller!)

Zoals de opdracht al zegt moet je applicatie eerst PORTD instellen op OUTPUT en vervolges om de beurt twee LED's op aan en uit zetten met een interval van 500 ms. Dit gebeurt met de volgende code (zie het bijbehorende commentaar):

```
DDRD = 0b11111111;   //Set PORTD to OUTPUT

    while (1)
    {


            PORTD = 0x80; //Turn on only PORTD7
            wait(500);           //Wait for 500ms

            PORTD = 0x40; //Turn on only PORTD7

            wait(500);           //Wait for 500ms

    }
```

Link naar repository:
https://github.com/MaurodeLyon/Microcontrollers/tree/master/Track1/B2

B.3

Maak een applicatie die de led op PORTD.7 laat knipperen als drukknop PORTC.0 laag (0) is (ingedrukt) en stopt bij het loslaten van de drukknop.

Zoals de opdracht al noemde moet je applicatie bij deze opdracht een LED laten knipperen als reactie op het indrukken van een drukknop. Dit doen wij door eerst PORTC op INPUT te zetten en PORTD op output te zetten. Vervolgens checkt de applicatie met een interval van 250 ms wat de status is van PINC en of het eerste bit aan staat (wat betekend dat de benodigde drukknop zou zijn ingedrukt). Zo ja, dan zal de LED(s) op PORTD getoggled worden. Mocht het eerste bit niet aanstaan zal PORTD uitgezet worden (zie commentaar). Hieronder de essentiele code van deze opdracht:

```
DDRD = 0b11111111;              // zet D naar output
DDRC = 0b00000000;              // zet C naar input
PORTD = 0x0;                    // zet port D op 0
while (1)                       //
{                               //
    wait(250);                  // wacht 250 ms
    if(PINC & 0x1)PORTD ^= 0x40;// als pinc bit 1 aan is toggle portD bit7
    else PORTD = 0x0;           // anders zet je port D bit 1 uit

}
return 1;
```

Link naar repository:

https://github.com/MaurodeLyon/Microcontrollers/tree/master/Track1/B3

B.4

Implementeer een looplicht applicatie op de LED's van PORTD. Tussen elke verandering van output zit 50ms (milliseconden). Hoe zou je dit kunnen meten?. Om een eenvoudig looplicht te maken kun je gebruik maken van de shift operatoren in C (de >> en de < <). Dit heb je ook al gedaan in periode TI-1.1 op het GUI board.

Zoals hierboven al is uitgelegd moet de applicatie van deze opdracht een "Looplicht" maken met de LED's van PORTD. Dit gebeurd door eerst PORTD op OUTPUT te zetten en deze een standaard waarde te geven (om deze te kunnen bitshiften verderop in de code). Vervolgens checkt de applicatie om de 50 ms eerst of de animatie al ten einde is. Zo ja, dan zal de applicatie PORTD resetten naar 0x1 en weer 50 ms wachten. Is dit echter niet het geval dan zal de applicatie de waarde/status van PORTD gaan bitshiften. Dit zorgt ervoor dat het eerstvolgende LED op de strip die is aangesloten op PORTD zal aangaan (de rest van de LED's is dan uit) waarna er weer 50 ms zal worden gewacht (zie ook het commentaar bij de onderstaande code).

De code die bij deze opdracht hoort:

```
DDRD = 0b11111111;      //Set PORTD to OUTPUT
PORTD= 0x1;             //Give PORTD a default value to bit-shift
while (1)
{

    if(PORTD >= 0x80)   //If statement checks if the current value of PORTD has reached the end of the strip of LEDs
    {
        PORTD=0x1;      //Reset to the begin of the strip of LEDs
        wait(50);       //Wait for 50 ms
    }
    PORTD= PORTD << 1;  //Change the value of PORTD with bit-shifting so that the LED thats next in line goes on
     wait(50);      //wait for 50 ms

}
```

Link naar repository:
https://github.com/MaurodeLyon/Microcontrollers/tree/master/Track1/B4

B.5

Een looplicht kun je implementeren met een schuifoperatie. Als het gewenste patroon niet zo eenvoudig is kun je e.a. met een grote if-then-else of switch-case constructie implementeren. Dit levert, in het algemeen, slecht onderhoudbare en starre implementaties op. Beter is om een lichtpatroon te sturen vanuit een datastructuur, bijvoorbeeld een C array.
Enig idee hoe dit moet? Zie ook het voorbeeld in de code repository. Implementeer een lichteffect met behulp van deze techniek.

Zoals in de opdracht is vermeld moet je een animatie aansturen op een strip met LED's. Zoals in de opdracht is voorgesteld hebben wij de animatie opgeslagen in een array van structs om grote switches en if-then-else statements te voorkomen. Hierdoor kan de animatie in kwestie ook snel worden aangepast. De desbetreffende struct (genaamd 'step') bevat een code waarin staat welke LED's aan of uit moeten en een value die aangeeft wat de pause moet zijn tussen deze en de volgende stap. Voor de uitwerking van deze array en deze struct verwijs ik naar de repository.

Als de applicatie draait zal PORTD eerst op OUTPUT worden ingesteld. Verder word ook een value(index) geinitialiseerd die bij gaat houden bij welke stap in de array we zijn. Hierna checkt de applicatie om de zoveel tijd (aangegeven bij de huidge step) of we aan het einde van de animatie zijn gekomen. Zo nee, dan zal PORTD de waarde van de huidige step krijgen en zal de index waarde met 1 worden verhoogd (in andere woorden de volgende step wordt geselecteerd). Vervolgens zal er gewacht worden voor een bepaalde tijd die door de huidige step wordt gedefinieerd. Mocht de animatie ten einde zijn dan zal de index weer op 0 gezet worden zodat de animatie opnieuw begint (zie het commentaar bij de onderstaande code).
De essentiele code bij deze opdracht:

```
int main(void)
{

    int index=0;                                        //initiates index value which is used to navigate through the earlier mentioned array
    DDRD = 0b11111111;                                  //Set PORTD to OUTPUT
    while (1)
    {
        if((sizeof(steps)/sizeof(steps[0])) >= index)       //This if-statement checks if we have reached the end of the array
        {
            PORTD= steps[index].data;                       //Take the next value for PORTD out of the array

            index++;                                        //increment the index with 1
            wait(steps[index].delay);                       //"freeze" the program for a given amount of time which is taken from the array
        }
        else{
            index=0;
        }                                   //Set the index value on zero, therefore restarting the loop}

    }
}
```

Link naar repository:
https://github.com/MaurodeLyon/Microcontrollers/tree/master/Track1/B5

B.6

Toestanden. Maak een applicatie die de led op PORTD.7 laat knipperen met een frequentie van circa 1Hz (1 keer per seconde). Als nu PORTC.0 kort wordt ingedrukt gaat (en blijft) de led sneller knipperen (bijvoorbeeld 4H). Bij nogmaals kort drukken gaat (en blijft) de led weer knipperen met een frequentie van 1Hz.

Zoals in de opdracht al uitgelegd moet deze applicatie de toestand van een LED (de LED van PORTD.7 om precies te zijn) veranderen op basis van het indrukken van een drukknop die is aangesloten op PORTC (PORTC.0 op precies te zijn).

In onze applicatie gebeurt dat door eerst PORTD op OUTPUT te zetten en PORTC op INPUT en word PORTD op 0x0 gezet. Ook worden er eerst twee values geinitialiseerd (de toggle value die zal bijhouden of het LED op 1 of 4 Hz moet knipperen en de "Time to wait" value T2W. Hierna zal de applicatie om de zoveel tijd (aangegeven door T2W) controleren of de drukknop die is aangesloten op PORTC.0 is ingedrukt. Zo ja, dan zal gecontroleerd worden wat de huidige instelling is van de LED en zal deze worden aangepast (zie if-then-else statement van de onderstaande code). Op het einde van de while loop zal de waarde van de LED aangesloten op PORTD (PORTD.7) worden omgedraaid (zie commentaar bij de onderstaande code).

De essentiële code van deze opdracht:

```
DDRD = 0b11111111;              // zet D naar output
DDRC = 0b00000000;              // zet C naar input
PORTD = 0x0;                    // zet port D op 0
int toggle = 0;                 // maak toggle
int T2W = 1000;                 // time to wait
while (1)
{
    if (PINC & 0x1)     {           // registreer button press
        if(toggle == 0)
        {
            toggle = 1;
            T2W = 500;              // 1 keer per seconden knipperen
        }
        else
        {
            toggle = 0;
            T2W = 250;              // 2 keer per seconden knipperen
        }
    }
    wait(T2W);
    PORTD ^= 0x40;                      // toggle portD bit7

}
```

Link naar repository:
https://github.com/MaurodeLyon/Microcontrollers/tree/master/Track1/B6

B.7A

Implementeer onderstaande 'eindige toestandsmachine' (eng: finite state machine of fsm). Een fsm is de basis van bijna elke embedded applicatie. Koffiemachines, televisies, pacemakers, ABS computers, alarmsystemen enz. zijn voorbeelden van applicaties waar de main-loop vaak bestaat uit een (ingewikkelde) eindige toestandsmachine.

Bij het maken van de opgave is gekozen voor een switch case met daarin verscheidene if-statements. Er is ook een enumerator aangemaakt die ervoor moet zorgen dat er duidelijke states zijn waar de switch doorheen gaat.

```
DDRC = 0b11111111;              // zet C naar output
DDRD = 0b00000000;              // zet D naar input
typedef enum {START,S1,S2,S3,END} STATE;
STATE state = START;
```

```
while (1)
{
    switch (state)
    {
        case START:
            if(PIND & BIT(6)) state = S1;
            PORTC = 0x1;
        break;
        case S1:
            if(PIND & BIT(7)) state = START;
            if(PIND & BIT(5)) state = S2;
            PORTC = 0x2;
        break;
        case S2:
            if(PIND & BIT(7)) state = START;
            if(PIND & BIT(6)) state = S1;
            if(PIND & BIT(5)) state = S3;
            PORTC = 0x4;
        break;
        case S3:
            if(PIND & BIT(7)) state = START;
            if(PIND & BIT(5) || PIND & BIT(6)) state = END;
            PORTC = 0x8;
        break;
        case END:
            if(PIND & BIT(7)) state = START;
            PORTC = 0x10;
        break;
    }
    PIND = 0;
}
```

Ondanks deze code werkte het programma niet op de bedoelde manier en ging hij regelmatig naar de end state.

Link naar repository:
https://github.com/MaurodeLyon/Microcontrollers/tree/master/Track1/B7a

Track 2

BEWIJS
https://www.youtube.com/watch?v=P8XSCOYnRaw

A.

Good

Er zijn drie soorten interrupts: blocking, polling en interrupting.

Blocking: snel en ideaal voor processen waarbij timing van belang is, maar waarbij er verder geen taken voltooid moeten worden.

Polling: langzamer de microcontroller controleert regelmatig of er al voldaan is aan een bepaalde waarde. Dit is langzamer, maar de controller kan ondertussen andere taken voltooien.

Interrupts: snel en ze zorgen ervoor dat de controller andere taken kan voltooien interrupts combineerd hierbij het beste van beiden.

BAD

Je moet een interrupt niet te lang maken anders komt je main-loop nooit aan de beurt.

Interrupts hebben verschillende prioriteiten en het is dan ook van belang dat je ze goed indeeld.

Het is belangrijk dat je processor intensieve taken in de main-loop houdt en dat je de interrupts zo kort mogelijk houdt.

The ugly

In dit artikel gaat Elliot Williams in op de minder mooie aspecten van het gebruiken van bijvoorbeeld interrupts.

The “obvious” pitfall when sharing variables with ISRs is the race condition: your code sets the variable’s value here and then uses it again there. The “race” in question is whether your code can get fast enough from point A to point B without an interrupt occurring in the middle.

Als je variabelen toegankelijk maakt voor je main én interrupts kan je nooit weten wanneer een interrupt zal worden geactiveerd (murphy’s law) waardoor je variabele kan veranderen naar iets ongewenst na het checken op deze ongewenste waardes.

Een mogelijke oplossing is het gebruiken van een shadow copy. Dit brengt dan wel enige problemen met zich mee wanneer de code op een 8 of 16 bit ARM machine wordt gebruikt.

Een variabele zal namelijk hoogstwaarschijnlijk (op een 8/16bit machine) in meerdere regels machine code worden opgeslagen (en is dus niet “Atomic”). Een interrupt tijdens het uitvoeren van deze code kan foutieve waardes opleveren.

Als laatste komt men met een goed werkende oplossing. Men zet gewoon de interrupts tijdelijk uit terwijl de main methode met de meest huidige variable werkt.

B.1

Download ioisr.c uit de repository en executeer deze op het BIGAVR board. Verklaar de werking

Tijdens uitvoeren van deze code is te zien dat PORTD.7 om de 500 ms aan/uit wordt gezet.
Als er op de knop PORTD.0 wordt gedrukt gaat PORTD.5 aan.
Als er op de knop PORTD.1 wordt gedrukt gaat PORTD.5 uit.

We gaan nu in de code kijken hoe dit precies in elkaar zit. Als eerste zal de main methode worden uitgevoerd. Er gebeuren verschillende dingen in de main methode.

Zoals de documentatie al uitlegt wordt PORTD ingesteld. Hierbij worden PORTD.4 t/m PORTD.7 ingesteld voor output en PORTD.0 t/m PORTD.3 ingesteld voor input (hier komen ook de interrupts vandaan, maar meer hierover later).

EICRA |= 0x0B;
EIMSK |= 0x03;

Hier worden de verschillende interrupts ingesteld.
Bij EICRA |= 0x0B; wordt ingesteld op wat voor soort interrupts INT0 en INT1 moeten reageren.

De binaire code van 0x0B is: 0000 0000 0000 1011.

Hierbij vertegenwoordigt 11 INT0 en 10 INT1. 11 houdt in dat de interrupt zal worden geactiveerd bij een rising edge en 10 houd in dat de interrupt zal worden geactiveerd bij een falling edge.
Vervolgens zal bij EIMSK |= 0x03 INT0 en INT1 worden geactiveerd. Een 1 staat voor geactiveerd en een 0 staat voor gedeactiveerd.

De binaire code van 0x03 is 0000 0000 0000 0011.

Hierbij vertegenwoordigt de meest rechtse 1 INT0 en die ernaast INT1 (en zo gaat deze rij door t/m INT7).

sei();
Hiermee zal het interrupt systeem van het board geactiveerd worden.

Vervolgens gaat een infinite while loop PORTD.5 om de 500 ms aan of uitzetten (togglen).

Buiten de main functie zijn er ook twee interrupt methodes voor INT0 en INT1. Hieronder een code fragment van de interrupt methode voor INT0:

```
ISR( INT0_vect ){
    PORTD |= (1<<5);
}
```

De interrupt methode voor INT0 zet PORTD.5 aan terwijl de interrupt methode voor INT1 deze weer uitzet. Als de interrupt methodes worden aangeroepen zal de while loop in de main methode tijdelijk stilstaan.

B.2

Implementeer een looplicht applicatie waarbij de main() bestaat uit een initialisatie gedeelte met daarna een lege while(true) loop. In ISR_INT1 en ISR_INT2 maakt het looplicht steeds 1 stap. Het looplicht wordt dus gestuurd vanuit de ISR's, niet vanuit de main().

Zoals in de opdracht is beschreven moet je applicatie een looplicht animatie doorlopen via interrupts. Deze interrupts,wanneer geactiveerd, moeten de volgende stap van de animatie tonen op PORTD.4 t/m PORTD.7. Het looplicht zelf is een aangepaste versie van onze uitwerking van B5 uit week 1.

Als eerste gaat de applicatie door de main methode heen. Deze initialiseerd het interrupt systeem en PORTD. Eerst wordt PORTD.4 t/m PORTD.7 op output en PORTD.0 t/m PORTD.3 op input gezet. Vervolgens wordt er gespecificeerd waarop de interrupts moeten reageren (rising of falling edge). In ons geval zetten we INT0 t/m INT3 op falling edge. Vervolgens activeren wij deze interrupts bij EIMSK. Hierna wordt het interrupt systeem geactiveerd met sei().

Hierna gaat de main methode een infinite while loop in waar niks gebeurd. De rest van de applicatie-logica speelt zich ergens anders af. Net zoals bij B5 uit week 1 heb je een struct die gegevens van een stap bezit en een array met daarin verschillende stappen.Boven aan de code staan 4 Interrupt methodes (zie code hieronder). Deze roepen alle 4 de methode nextstep() aan. In deze methode zit de logica voor het functioneren van het looplicht. Als eerste checkt nextstep of het einde van het looplicht is bereikt. Zo niet, dan toont hij de volgende stap en verhoogt hij de index waarde met 1 (zodat de methode bij een volgende aanroep de volgende stap kan laten zien) waarna er voor een in de struct gespecificeerde tijd wacht. Als de loopanimatie ten einde is wordt de indexwaarde weer op 0 gezet en begint het looplicht opnieuw.

Een probleem dat wij tegen kwamen bij onze uitwerking was dat wij INT0 en INT1 wel geactiveerd/werkend kregen maar dit niet lukte bij INT2 of INT3 terwijl uit de binaire codes te halen is dat wij deze toch wel activeren.

Essentiele code:

Interrupt methode:

```
ISR( INT1_vect )
{
	nextStep();	//Show the next step of the animation
}
```

looplicht methode (zie ook B5 week 1):

```
void nextStep()
{
    if((sizeof(steps)/sizeof(steps[0])) >= index)       //This if-statement checks if we have reached the end of the array
    {
        PORTD= steps[index].data;                       //Take the next value for PORTD out of the array

        index++;                                        //increment the index with 1
        wait(steps[index].delay);                       //"freeze" the program for a given amount of time which is taken from the array
    }
    else{
        index=0;
    }                                   //Set the index value on zero, therefore restarting the loop}

}
```

Initialisatie PORTD en Interrupts:

```
// Init I/O
DDRD = 0xF0;            // PORTD(7:4) output, PORTD(3:0) input
// Init Interrupt hardware
EICRA |= 0xAA;          // INT0 through INT3 set to falling edge
EIMSK |= 0xF;           // Enable INT1 through INT3


sei();              // Enable global interrupt system

while (1)
{

}
```

Link naar repository:
https://github.com/MaurodeLyon/Microcontrollers/tree/master/Track2/B2

B.3

Implementeer een functie waarmee een 7 segment display kan worden aangestuurd.

Bijvoorbeeld: void display(int digit){…}.

er is een display functie gemaakt die met behulp van een lookup table alle waardes op een hexadecimaal display kan weergeven.

als het getal groter is dan 15 dan wordt er een E met een punt weergegeven om een error voor te stellen.

er zijn ook functies gemaakt voor het ophogen en verlagen van de waarde die wordt weergeven.

```
static int displayDigit;
static unsigned char HEXNUMBERS[17] =
{
    0b00111111, //0
    0b00000110, //1
    0b01011011, //2
    0b01001111, //3
    0b01100110, //4
    0b01101101, //5
    0b01111101, //6
    0b00000111, //7
    0b01111111, //8
    0b01101111, //9
    0b01110111, //A (10)
    0b01111100, //B (11)
    0b01011000, //C (12)
    0b01011110, //D (13)
    0b01111001, //E (14)
    0b01110001, //F (15)
    0b11111001  //ER(16)
};
```

```
void increase(void)
{
    displayDigit++;
    display(displayDigit);
}

void decrease(void)
{
    displayDigit--;
    display(displayDigit);
}
```

```
void display(int digit)
{
    displayDigit = digit;                       //veranderd globale variabele
    if (0 <= displayDigit && displayDigit <= 15)//als het 0 < getal < 15
    {
        PORTD = HEXNUMBERS[displayDigit];       //display getal
    }
    else if(displayDigit > 15)                  //als het getal > 15
    {
        PORTD = HEXNUMBERS[16];                 //display error = E.
    }
}
```

```
int main(void)
{
    /* Replace with your application code */
    displayDigit = 0;
    DDRD = 0b11111111;      // PORTD all output

    while (1)
    {
        increase(); // verhoogt de waarde van de globale variabele met 1 en update het 7-segment display
        wait(250);  // wacht 250 ms
    }
}
```

Link naar repository:
https://github.com/MaurodeLyon/Microcontrollers/tree/master/Track2/B3

B.4

Bestudeer de code uit de repository uit lesweek 1 (lookup.c) waarin met behulp van een lookup-table een patronen op leds worden afgebeeld. Implementeer een '7 segment display lichteffect' met behulp van deze techniek. Zie presentatie voor de aansluiting van het 7 segment display.

Met behulp van de lookup techniek heb ik een lichteffect gemaakt die van boven naar beneden het 7-segment display activeert.

```
PATTERN_STRUCT pattern[] = {
    {0b00000001, 150},
    {0b00100010, 150},
    {0b01000000, 150},
    {0b00010100, 150},
    {0b00001000, 150},
    {0xFF, 0}
};
```

Link naar repository:
https://github.com/MaurodeLyon/Microcontrollers/tree/master/Track2/B4

Track 3

BEWIJS
https://www.youtube.com/watch?v=tTreXhOzpCw

A.1

De initialisatiecode om het LCD in 4-bits mode te initialiseren (dia pagina 10). Welke data, commando's en instellingen moet je maken om het LCD te initialiseren? Maak een tabel en vertaal deze handelingen in een C functie.

Simulator handeling tabel

Stap	D7	D6	D5	D4	D3	D2	D1	Do	E	R/W	RS	uitleg
1	0	0	0	0	0	0	1	0	0	0	0	
2	0	0	0	0	0	0	1	0	1	0	0	Cursor home.
3	0	0	0	0	0	0	1	0	0	0	0	
4	0	0	1	0	0	0	0	0	0	0	0	4-bit mode ON
5	0	0	1	0	0	0	0	0	1	0	0	1 line
6	0	0	1	0	0	0	0	0	0	0	0	5*8 font

C functie code:

```
int main(void)
{
    DDRC = 0b11111111;  //Set PORTD to OUTPUT
    lcd_command(0x02);
    lcd_command(0x20);
    while (1)
    {
    }
}
```

```
void lcd_command ( unsigned char dat )
{
    PORTC = dat & 0xF0;             // hoge nibble
    PORTC = PORTC | 0x08;           // data (RS=0),
                                    // start (EN=1)
    _delay_ms(1);                   // wait 1 ms
    PORTC = 0x04;                   // stop (EN = 0)

    PORTC = (dat & 0x0F) << 4;      // lage nibble
    PORTC = PORTC | 0x08;           // data (RS=0),
                                    // start (EN=1)
    _delay_ms(1);                   // wait 1 ms
    PORTC = 0x00;                   // stop
                                    // (EN=0 RS=0)
}
```

Link naar repository:
https://github.com/MaurodeLyon/Microcontrollers/tree/master/Track3/A1

A.2

Verklaar de functie van de Do-D7 pinnen, de E pin en de RS pin.

De pinnen Do-D7 zijn de data bussen die worden gebruikt om data te lezen/schrijven.
De E pin bepaald of het lcd scherm de data verwerkt.
De RS pin wordt gebruikt om data te schrijven naar de instruction register of naar het data register

A.3

Het afbeelden van enkele karakters op het display. Welke data, commando's en/of instellingen moet je maken? Vertaal deze handelingen in een C functie.

Je moet een aantal functies hebben:

- commando
- initialisatie
- schrijven van een karakter

Main

```
init_lcd();
lcd_writeChar('H');
lcd_writeChar('e');
lcd_writeChar('l');
lcd_writeChar('l');
lcd_writeChar('o');
```

Initialisatie

```
void init_lcd(void)
{
    // return home
    lcd_command( 0x02 );
    // mode: 4 bits interface data, 2 lines, 5x8 dots
    lcd_command( 0x28 );
    // display: on, cursor off, blinking off
    lcd_command( 0x0C );
    // entry mode: cursor to right, no shift
    lcd_command( 0x06 );
    // RAM adress: 0, first position, line 1
    lcd_command( 0x80 );
}
```

Commando

```
void lcd_command ( unsigned char dat )
{
    PORTC = dat & 0xF0;             // hoge nibble
    PORTC = PORTC | 0x08;           // data (RS=0),
    // start (EN=1)
    _delay_ms(1);                   // wait 1 ms
    PORTC = 0x04;                   // stop (EN = 0)

    PORTC = (dat & 0x0F) << 4;      // lage nibble
    PORTC = PORTC | 0x08;           // data (RS=0),
    // start (EN=1)
    _delay_ms(1);                   // wait 1 ms
    PORTC = 0x00;                   // stop
    // (EN=0 RS=0)
}
```

Schrijven karakter

```
void lcd_writeChar( unsigned char dat )
{
    PORTC = dat & 0xF0;             // hoge nibble
    PORTC = PORTC | 0x0C;           // data (RS=1),
    // start (EN=1)
    _delay_ms(1);                   // wait 1 ms
    PORTC = 0x04;                   // stop (EN = 0)

    PORTC = (dat & 0x0F) << 4;      // lage nibble
    PORTC = PORTC | 0x0C;           // data (RS=1),
    // start (EN=1)
    _delay_ms(1);                   // wait 1 ms
    PORTC = 0x00;                   // stop
    // (EN=0 RS=0)
}
```

Link naar repository:
https://github.com/MaurodeLyon/Microcontrollers/tree/master/Track3/A3

A.4

Link naar repository:
https://github.com/MaurodeLyon/Microcontrollers/tree/master/Track3/A4

Het schuiven van alle tekst naar rechts. Welke data, commando's en/of instellingen moet je maken? Vertaal deze handelingen in een C functie.

Je moet een aantal functies hebben:

- commando
- initialisatie
- schrijven van een karakter of tekst
- scrollen van de karakter of tekst

```
void display_text(char *str)
{
    for (int i=0; i< strlen(str); i++)
    {
        lcd_writeChar( str[i] );
    }
}
```

```
int main(void)
{
    /* Replace with your application code */
    init();
    display_text("Hello, world");
    while (1)
    {
        _delay_ms(250);
        scroll_rechts();
        _delay_ms(250);
        scroll_rechts();
        _delay_ms(250);
        scroll_rechts();
        _delay_ms(250);
        scroll_links();
        _delay_ms(250);
        scroll_links();
        _delay_ms(250);
        scroll_links();
    }
}

void scroll_links()
{
    lcd_command(0x18);
}

void scroll_rechts()
{
    lcd_command(0x1c);
}
```

B.1

*Ontwerp een LCD 'C module' (dus een *.c en een *.h file, bijvoorbeeld lcd.c en lcd.h).*

Het idee van deze opdracht is om generieke LCD functies te schrijven waarmee verschillende dingen gedaan kunnen worden met het LCD scherm. Men moest 3 functies (init, display_text en setcursor) uitwerken.

Bij onze oplossing hebben we gebruik gemaakt van de al gegeven methodes lcd_writeChar en lcd _command. De init methode maakt het LCD klaar voor gebruik. Deze methode haalt onder andere het scherm leeg en zet de cursor op positie 0,0.

Bij display_text maken we gebruik van de lcd_writeChar methode. Ook staat er (in comments) een mogelijkheid om eerst het scherm leeg te maken voor het schrijven. De functie roept per character in de array van characters de desbetreffende methode aan. Bij setcursor maken we gebruik van de lcd_command methode. We gebruiken deze methode om de cursor positie op te schuiven naar een gewenste positie. De code om de cursor een plek naar rechts te doen bewegen is 0x14 (0001 0100 in binary). Deze methode met deze code wordt aangeroepen tot de aangegeven position is bereikt.
Hieronder samples van de code:

```
void init()
{
    //Initialise LCD
    DDRC = 0b11111111;  //Set PORTD to OUTPUT

    // return home
    lcd_command( 0x02 );
    // mode: 4 bits interface data, 2 lines, 5x8 dots
    lcd_command( 0x28 );
    // display: on, cursor off, blinking off
    lcd_command( 0x0C );
    // entry mode: cursor to right, no shift
    lcd_command( 0x06 );
    // RAM adress: 0, first position, line 1
    lcd_command( 0x80 );

void display_text(char *str)
{
    //Display an array of chars on the LCD

    //lcd_command(0x80);  <-- would clear the screen before writing again
    for (int i=0; i< strlen(str); i++)
    {
        lcd_writeChar( str[i] );
    }

void set_cursor(int position)
{
    int i;
    for(i=0; i<position ; i++)
    {
        lcd_command(0x14);      //This command tells the LCD to move its to the right by one.
                                //It will be run until the preferred position is reached
```

Link naar de repository:
https://github.com/MaurodeLyon/Microcontrollers/tree/master/Track3/B1

B.2

Realiseer met AVR T/C-2 (counter mode) een 'teller' die het aantal keren 'toets indrukken' telt en weergeeft op het LCD (met de code uit B.1).

De opdracht is hier om een teller te maken toets indrukken moet bijhouden. Deze moet gebruik maken van de AVR T/C-2. Ook moet dezecode werken met de IR-sensor).

Bij onze oplossing was er een klein probleem bij het het gebruiken van de code van B2 en de LCD code van B1.Beide onderdelen werkte n gescheiden van elkaar maar in combinatie kregen we niks te zien op het scherm. Daarom hebben we er op moment van schrijven voor gekozen om de reacties te tonen op de LED's van PortA en PORTB. De code initialiseerd als eerste PORTD.7 voor INPUT en PORTA en PORTB voor OUTPUT en wordt de counter value op 0 gezet. Vervolgens gaan we de teller initaliseren. Eerst geven we de teller een start value van -10 waarna we de teller instellen om te reageren op een overflow interrupt (zie code documentatie). Vervolgens zetten we interrupts aan voor PORTD en zetten we de teller aan. In de while methode die hierop volgt laten we de values zien op PORTA en PORTB. Hier zou je ook de LCD code kunnen aanroepen. Wat de applicatie moet doen bij deze interrupt is ook gedefinieerd. Deze moet de teller resetten en bijhouden hoe vaak de interrupt is aangeroepen. Deze code werkt ook met een IR-sensor als deze is aangesloten op PORTD.

Hieronder wat sample code:

```
ISR(TIMER2_OVF_vect )
{
     TCNT2 = -10;    // Preset value
     counter++;          // Increment counter


}
```

```
int main(void)
{
    counter=0;
    DDRD=   ~BIT(7); //Put PORTD.7 on INPUT
    DDRA = 0xFF;    //Set PORTA on OUTPUT.
    DDRB = 0xFF;    //Set PORTB on OUTPUT.


    TCNT2 = -10;               // Initialise the value the counter should hold
    TIMSK |= BIT(6);           // Enable the T2 overflow interrupt. This interrupt will launch when the counter reaches 0.
    SREG|= BIT(7);             //Turn on the interrupt for PORTD
    TCCR2 = 0x07;              // Enable the counter in normal mode and start it

    while (1)
    {
        PORTA = TCNT2;         // show value counter 2 on PORTA LED's
        PORTB = counter;       // show value tenth counter on PORTB LED's
        wait(10);              // every 10 ms

    }
}
```

Link naar de repository:
https://github.com/MaurodeLyon/Microcontrollers/tree/master/Track3/B2

B.3

Realiseer met de AVR T/C-2 (timer mode) pulsen op PORTD.7 van 15ms (milliseconden) 'hoog' en daarna 25ms 'laag'. Realiseer dit door gebruik te maken van interrupts en meet met een oscilloscoop op PORTD.7 om het gewenste gedrag te controleren.

Vragen die je jezelf kunt stellen:

- *Zet de prescaler, in deze opdracht, op 1024. Wat betekent dit?*
  - *De snelheid van de klok wordt wat trager gemaakt.*
- *Met welke frequentie wordt Timer2 dan aangestuurd?*
  - *7812.5 Hz.*
- *Met welk tijdsinterval telt Timer2?*
  - *128 microseconde.*
- *Wat is de waarde in TCNT2 die je moet instellen om 15ms af te tellen?*
  - *15000/128 = 117.*
- *Kun je dan een interrupt genereren en daar een I/O pin toggelen?*
  - *Ja, zie de code.*

```
int flip;
// Interrupt routine timer2 overflow
//
ISR( TIMER2_OVF_vect )
{
    PORTD ^= 0x80;          // toggle portd.7
    if(flip)
    {
        TCNT2 = 255-195;    // 15 ms
        flip = 0;
    }else
    {
        TCNT2 = 255-117;    // 25 ms
        flip = 1;
    }
}
// Initialize timer2
//
void timer2Init( void )
{
    flip = 0;
    TCNT2 = 0;              // Preset value of counter 2
    TIMSK |= (1<<6);        // T2 overflow interrupt enable
    TCCR2 = 0b00000101;     // Initialize T2: ext.counting, rising edge, run
}


// Main program: Counting on T2
int main( void )
{
    DDRD = 0x80;                // set PORTD.7 for input
    DDRA = 0xFF;                // set PORTA for output (shows countregister)
    DDRB = 0xFF;                // set PORTB for output (shows tenthvalue)
    timer2Init();
    SREG |= (1<<7);             // turn_on intr all
    while (1)
```

Link naar repository:
https://github.com/MaurodeLyon/Microcontrollers/tree/master/Track3/B3