# Besturingssystemen 2

Bert De Saffel

16 mei 2017

# Inhoudsopgave

# Deel I

# Datasheet

# Hoofdstuk 1

# Belangrijke registers

Dit hoofdstuk bevat de belangrijkste registers met hun pagina in de datasheet. Als ze bitadresseerbaar zijn bevat het ook nog eens de relevante bits tussen haakjes.

## 1.1 Timers

Volgende registers bevinden zich in SFR pagina 0.

- **TMOD**: 290
- **CKCON**: 291

## 1.2 ADC

Volgende registers bevinden zich in SFR pagina 0.

- **ADC0CN**: 57 (AD0EN, AD0BUSY, AD0INT)
- **AMX0CF**: 72
- **AMX0SL**: 73
- **REF0CN**: 108

## 1.3 Seriële communicatie

Volgende registers bevinden zich in SFR pagina 0.

- **SCON0**: 271 (TI0, RI0)
- **SSTA0**: 272 (SMOD0)

# Deel II

# Configuratie

# Hoofdstuk 2

# Gebruik van de 7-segment display

## 2.1 Initialisatie

De 7-segment display kan tot maximum 4 poorten ingesteld worden waarbij elke poort een 7-segment display voorstelt.

```
mov P0MDOUT, #0FFh
mov P1MDOUT, #0FFh
mov P2MDOUT, #0FFh
mov P3MDOUT, #0FFh
```

Uiteraard is het mogelijk dat andere poorten gebruikt worden.
De 7-segment display bevat, zoals de naam het doet vermoeden, 7 segmenten die elk onafhankelijk aan of uit kunnen staan. De getallen 0 tot en met 9 zijn dus een bepaald bitpatroon. In het volgende voorbeeld schrijven we deze bitpatronen weg van 20h tot en met 29h. De underscores zijn enkel bedoeld voor de leesbaarheid en dienen niet in het codesegment te komen.

```
mov 20h, #0011_1111b //0
mov 21h, #0000_0110b //1
mov 22h, #0101_0011b //2
mov 23h, #0100_1011b //3
mov 24h, #0110_0110b //4
mov 25h, #0110_1101b //5
mov 26h, #0111_1101b //6
mov 27h, #0000_0111b //7
mov 28h, #0111_1111b //8
mov 29h, #0110_1111b //9
```

Merk op dat het 8ste bit nooit gebruikt wordt. Logisch aangezien het een 7-segment display is.

## 2.2 Afbeelden

Om een getal af te beelden wordt het volgende gedaan. In R2 moet een waarde van 0 tot en met 9 zitten.

```
mov A, #20h
add A, R2
mov R0, R2
mov P0, @R0
```

Als in R2 het getal 4 zit, zal in de accumulator #24h zitten. Wat overeenkomt met het bitpatroon om een 4 af te beelden. Deze wordt in R0 gestopt aangezien je naar een uitvoerpin een pointer moet wegschrijven, A heeft die functionaliteit niet. Deze code zal de segmenten in een vorm van een vier doen laten branden.

# Hoofdstuk 3

# Timers

## 3.1 Timer 0 en Timer 1

Volgende instructies zijn van toepassing voor zowel timer 0 als timer 1.

1. **setb ET0 / setb ET1 (p. 147, p. 149)**
   Laat interrupts voor timer 0 / 1 toe. Zit in alle SFR pagina's.

2. **mov TMOD (p. 290)**
   Configureert zowel timer 0 als timer 1. De meest beduidende nibble configureert timer 1 en de minst beduidende nibble configureert timer 0. Bij elke nibble zijn de minst beduidende 2 bits het meest belangrijk. Als je maar één timer gebruikt laat je vanzelfsprekend de andere nibble op nul. Voor het instellen van een nibble kies je ofwel voor 1 (mode 1), wat de timer in 16-bit mode zet, of 2 (mode 2), wat de auto-reload functionaliteit geeft aan de timer, ten kosten van 8 bits. Een voorbeeld: *mov TMOD, #21h* zal timer 1 op mode 2 zetten en timer 0 op mode 1.

3. **mov CKCON (p. 291)**
   Bij deze configuratie moet er alleen gekeken worden naar de minst beduidende 2 bits, aangezien bit 4 en bit 3 altijd nul zullen zijn, aangezien die afhankelijk zijn van een instelling die we in TMOD niet gebruiken. Deze minst 2 beduidende bits zullen bepalen door hoeveel de systeemklok gedeeld moet worden. Er bestaan dus vier mogelijkheden:

   - mov CKCON, #00h → Systeemklok delen door 12
   - mov CKCON, #01h → Systeemklok delen door 4
   - mov CKCON, #02h → Systeemklok delen door 48
   - mov CKCON, #03h → Externe klok delen door 8

   In principe zal *#03h* nooit gebruikt worden, dus blijven er drie mogelijkheden over.

4. **mov TH0 / mov TH1 & mov TL0 / TH1**
   Een timer heeft een bepaalde startwaarde nodig. Dit is een 16-bit getal waarvan de 8 meest beduidende bits in TH0 of TH1 komen en de 8 minst beduidende bits in TL0 of TH1. Om te weten wat er in deze registers moeten komen zijn er eerst een aantal gegevens nodig.

   (a) De duur in seconden (t)

   (b) De klokfrequentie (klokfreq) $\rightarrow$ Bij de c8051f120 controller is dit altijd gelijk aan $\frac{(24,5*10^6)}{8} = 3062500$

   Eens we deze gegevens hebben kunnen we het aantal klokpulsen achterhalen via de volgende formule: $klokfreq * t$. Stel dat we om de seconde het lampje op P1.6 willen laten pinken. Initieël wordt de formule als volgt ingevuld: $3062500 * 1 = 3062500$. 3062500 is uiteraard te groot voor een 16 bit getal, daarom gaan we de systeemklok delen zoals aangegeven in CKCON. Als we delen door 4 of 12 is het getal nog altijd grooter dan $2^{16}$, daarom delen we door 48 wat uitkomt op ongeveer 63802. Aangezien een 16-bit timer overflowt op 65535 moeten we hiervaan 63802 aftrekken. Je komt uit op 1733 wat in hexadecimale notatie overeenkomt met F93A. Deze hexadecimale notatie hebben we zo meteen nodig. Aangezien een timer overloopt van FFFFh naar 0000h moet er van FFFFh de waarde F93Ah afgetrokken worden. $FFFFh - F93Ah = 06C5h$. De twee meest beduidende bits komen in TH0 of TH1 terecht en de twee minst beduidende bits komen in TL0 of TH1 terecht. In het geval dat timer 1 gebruikt wordt krijgen we:

   ```
   mov TH1, #06h
   mov TL1, #0C5h
   ```

   In het geval dat je de timer met auto-reload ingesteld hebt, is de waarde in TH1 kleiner dan 256 en zit deze waarde ook in TL0.

5. **setb TR0 / setb TR1**
   Tot slot moet de timer nog gestart worden. Deze instructie zorgt daarvoor.

6. **clr TF0 / clr TF1**
   In het geval dat de timer geen auto-reload gebruikt moet de timer, wanneer hij overloopt, gecleard worden en moeten de startwaarden opnieuw ingesteld worden met THx en TLx. Deze bevatten dezelfde waarden als bij het initieël instellen van de timer. Bij auto-reload moet je THx en TLx dus niet opnieuw invullen

## 3.2 Timer 2

Timer 2 heeft nog veel gelijkenissen met timer 0 of timer 1. We overlopen ze.

1. **setb ET2**
   Zoals timer 0 en timer 1 laat dit interrupts toe voor timer 2. Deze zitten trouwens allemaal in het register IE (pagina 149).

2. **mov TMR2CF**
   Bij dit controle register zijn we enkel geïnteresserd in het instellen van bit 4 en bit 3. Deze bepalen de deling van de systeemklok. Merk op dat deze timer niet kan delen door 48, het maximum is 12. Hier zijn er ook weer vier mogelijkheden:

   - mov TMR2CF, #00h → Systeemklok delen door 12
   - mov TMR2CF, #08h → De systeemklok als bron gebruiken
   - mov TMR2CF, #10h → Externe klok delen door 8
   - mov TMR2CF, #18h → Systeemklok delen door 2

   Ook zal hier *#10h* nooit gebruikt worden. Deze controleregister kan je terugvinden op pagina 298.

3. **mov TMR2H / mov TMR2L**
   Dit gebruikt hetzelfde principe als TH0 en TL0. Deze twee registers ondersteunen echter geen auto-reload, dus moet je deze waarden opnieuw instellen bij elke overflow.

4. **mov RCAP2H / mov RCAP2L**
   Dit wordt gebruikt in plaats van TMR2H en TMR2L wanneer auto-reload wel gewenst is. De auto-reload van timer 2 kan wel 16-bit waarden aan dus gaat de voorkeur naar RCAP en niet naar TMR.

5. **setb TR2** Timer 2 starten.

## 3.3 Timer 3

# Hoofdstuk 4

# Analoog naar digitaal converter

## 4.1 Meten van de chiptemperatuur

Om de chiptemperatuur te meten worden volgende commando's gebruikt.

1. **mov REF0CN**
   Dit register selecteer de referentie generator
2. **mov AMX0SL (p. 55)**
   Altijd op #0Fh zetten als je de chiptemperatuur wil meten.
3. **clr AD0INT**
   Dit voer je best vooraleer je een conversie start aangezien dit bit aantoont dat een data conversie voltooid is.
4. **setb AD0EN**
   De analoog naar digitale converter actief zetten en staat klaar om te converteren.
5. **setb AD0BUSY**
   De conversie starten.
6. **jnb AD0INT, $**
   Hier wachten we tot dat AD0INT op 1 staat, wat betekent dat de conversie klaar is.
7. **clr AD0INT**
   Deze bit terug op 0 zetten in het geval je nog een conversie wil doen. Bijvoorbeeld bij loops.

Een voorbeeldprogramma ziet er zo uit:

```
$include (c8051f120.inc)
cseg at 0000h
        jmp main
cseg at 0080h

main:
        clr EA
        mov WDTCN, #0DEh
        mov WDTCN, #0ADh
        setb EA
        mov REF0CN, #07h
        mov AMX0SL, #0Fh
        clr AD0INT
        setb AD0EN
        setb AD0BUSY
        jnb AD0INT, $
        clr AD0INT
        jmp $
```

Om deze temperatuur nu op een 7-segment display te tonen moet de digitale waarde berekent worden. Nadat een conversie wordt uitgevoerd zal het resultaat verschijnen in *ADC0H* en *ADC0L*. Aangezien het een 12-bit converter is zullen de 8 minst beduidende bits in ADC0L zitten en de 8 meest beduidende, waarvan 4 ongebruikt, in ADC0H.

# Hoofdstuk 5

# Seriële communicatie

In de oefeningen verloopt de communicatie tussen de microcontroller en een computer. De toestellen zijn verbonden met een RS-232 connector. De c8051f120 implementeerd de RS-232 standaard en voorziet 9 aansluitingen. De receive(RX) en transmit(TX) lijn zijn de twee belangrijkste hiervan terwijl de andere 7 handshakelijnen zijn die gebruikt worden om ontvanger en afzender te synchroniseren. Vooraleer communicatie mogelijk is moet je UART0 naar buiten brengen via `mov XBR0, #04h` (zie pagina 217).