FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Labo hardware

Computerhardware

W. Van den Breen

Opleiding
Academiejaar 2016–2017

UNIVERSITEIT
GENT

Inhoudsopgave

1 Inleiding

1.1 Embedded systemen

Embedded systemen zijn tegenwoordig nog maar moeilijk uit het dagelijks leven weg te denken. Mooie voorbeelden hiervan zijn GSM-toestellen, GPS-systemen, draagbare MP3-spelers en de in 2008 in Europa uitgebrachte iPhone van Apple.

Een embedded systeem is meestal opgebouwd rond een microcontroller. Aangezien een dergelijk systeem voornamelijk ontwikkeld wordt voor één welbepaalde toepassing dient het programma, dat door de microcontroller uitgevoerd wordt, dan ook weinig of nooit aangepast te worden. Het programma is bij wijze van spreken ingebed in de elektronica.

Bij embedded systemen kiest men ervoor om de productiekosten zo laag mogelijk te houden. Dit betekent dat de controller qua mogelijkheden volledig afgestemd wordt op het programma dat hij zal moeten uitvoeren. Aangezien de hardwaremogelijkheden van de controller, waaronder bijvoorbeeld de hoeveelheid geheugen, eerder beperkt zijn, dient men bij het programmeren van de controller uiterst zuinig om te springen met de beschikbare systeembronnen. Hoe minder geheugen het programma nodig heeft, hoe goedkoper het toestel kan gemaakt worden!

Gewone computersystemen, in tegenstelling tot embedded systemen, worden uitgerust met een microprocessor waar het uit te voeren programma constant wisselt. Om tegemoet te komen aan de veeleisendheid van de sofware die erop uitgevoerd wordt , worden computersystemen dan ook uitgerust met een enorm gamma aan systeembronnen. De programmeur moet dan ook bij het ontwikkelen van sofware niet gaan omzien om een kilobyte geheugen meer of minder te gebruiken.

1.2 Opbouw van een microcontroller

De basiscomponenten van een gewoon computersysteem zijn:

- Een centrale verwerkingseenheid oftewel CPU[1]
- Een zekere hoeveelheid extern geheugen. Een CPU is namelijk gebaseerd op de Van Neumann stored program architectuur waardoor elk programma dat door de CPU uitgevoerd wordt zich in het geheugen moet bevinden.
- In- en uitvoer of kortweg I/O. Omdat de gebruiker de computer interactief zou kunnen gebruiken dient men diverse randapparaten met het computersysteem te verbinden. Deze randapparaten stellen de gebruiker en de computer in staat om gegevens met elkaar uit te wisselen.

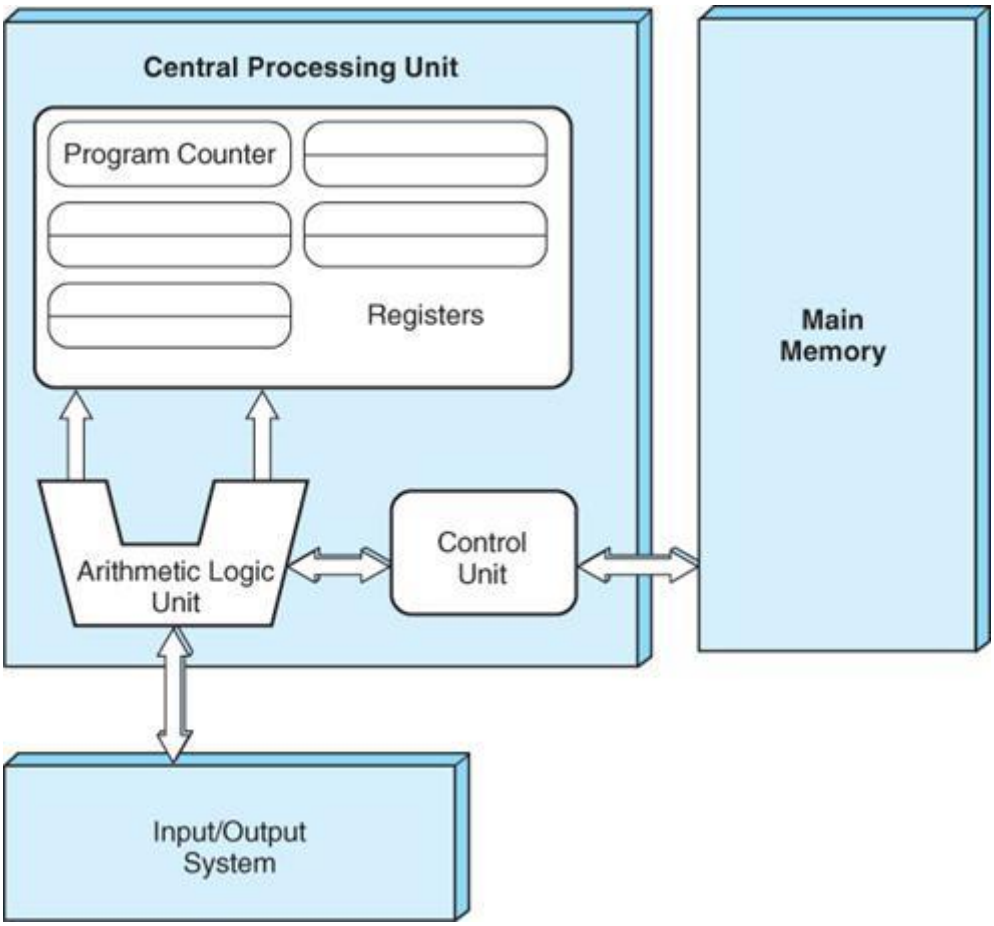
[1] Central Processing Unit

Wanneer we een eenvoudige CPU, een kleine hoeveelheid geheugen en een aantal communicatiekanalen in één enkele chip (behuizing) stoppen, hebben we een microcontroller. Een microcontroller is dus niks meer is dan een miniatuurversie van een computersysteem.

Binnen de microcontroller worden alle componenten met elkaar verbonden via een intern bussysteem. Dit systeem zorgt ervoor dat ieder element van de controller met elk ander element van de controller verbonden is. Op die manier kan men ervoor zorgen dat de gegevensuitwisseling binnen de controller zo efficiënt mogelijk verloopt.

1.2.1 De Von Neumann architectuur

De meeste microprocessoren zijn opgebouwd volgens de Von Neumann architectuur. Deze architectuur vergt dat het uit te voeren programma zich in het geheugen bevindt. Alle extra data die het programma nodig heeft bevindt zich in datzelfde geheugen.



Figuur 1: De Von Neumann architectuur

1.2.2 De Harvard architectuur

Bij sommige microcontrollers, waaronder de 8051 van Intel, opteert men voor de Harvard architectuur. Het opvallendste verschil met de Von Neumann architectuur is wellicht de strikte opdeling van het geheugen in twee delen, met name het programmageheugen en het datageheugen. Zoals de benaming zegt bevat het programmageheugen de uit te voeren code en bevat het datageheugen de bijhorende data. Door deze opsplitsing is het mogelijk om beide geheugens op hetzelfde moment te raadplegen waardoor onnodige wachttijden vermeden worden.

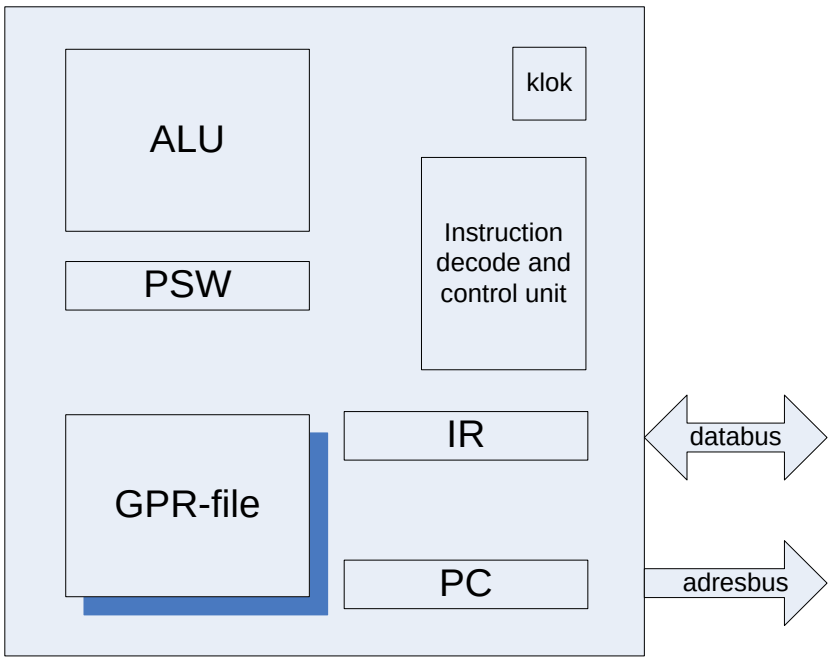
1.2.3 Soorten geheugens

Aangezien een microcontroller gedurende zijn volledige levensloop steeds hetzelfde programma uitvoert lag het voor de hand om het programma in te bedden in ROM-geheugen.

Tegenwoordig is de complexiteit van de software echter dermate toegenomen dat het voorkomen van eventuele softwarefouten na distributie eerder de regel is dan wel de uitzondering. Dit heeft als gevolg dat het programma van de controller sporadisch aangepast moet kunnen worden. Hierdoor zal men bij moderne microcontrollers gebruik maken van flash-programmageheugen i.p.v. ROM-programmageheugen.

Voor het datageheugen wordt gebruikt gemaakt van SRAM[2].

1.3 Structuur van de CPU



Figuur 2: Minimale CPU

[2] Static RAM

Volgens figuur 2 bevat de CPU volgende componenten:

- De ALU[3] die verantwoordelijk is voor het uitvoeren van rekenkundige en logische bewerkingen.
- De program counter, ook wel instruction pointer genoemd, die het adres bijhoudt van de volgende op te halen instructie.
- Het PSW[4] dat de status bevat van de laatst uitgevoerde instructie. De inhoud van dit speciaal register bestaat uit een aantal bits, vlaggetjes genoemd, die een welbepaalde betekenis hebben. Deze vlaggetjes, waaronder de zero vlag, de negative vlag, de overflow vlag, de carry-vlag, etc. stellen de CPU in staat om tijdens de uitvoer van een programma de nodige beslissingen te nemen. (cfr. if… else)
- Een aantal GPR's[5]. Iedere CPU heeft een kleine hoeveelheid geheugen nodig om tijdelijke gegevens te kunnen opslaan.

1.3.1 Soorten bussen

De CPU moet op een eenvoudige manier gegevens kunnen uitwisselen met zijn geheugen en vanzelfsprekend ook met zijn I/O. Een mogelijke, maar helaas onhaalbare, oplossing bestaat erin de CPU rechtstreeks te verbinden met alle andere componenten. Bovendien moet dat dan ook toegepast worden op het geheugen en uiteraard ook op het I/O-gedeelte. Dit leidt onvermijdelijk tot een enorme hoeveelheid aan individuele verbindingen.

Een betere oplossing bestaat erin om alle componenten aan te sluiten op één gemeenschappelijke bus. Zo is het aantal verbindingslijnen onafhankelijk van het aantal aangesloten componenten. Helaas kan de bus op een gegeven moment niet zomaar door gelijk welke component worden gebruikt . Wie deze bus mag gebruiken en wanneer wordt bepaald door de controle-eenheid. Het externe kloksignaal zorgt voor de bustiming.

De uiteindelijke oplossing maakt gebruik van twee aparte bussen, een bus voor het transport van data, de databus, en een bus voor het adresseren van een geheugenvakje, de adresbus. Iedere component moet gegevens op de databus kunnen plaatsen en moet gegevens van deze bus kunnen halen. De databus is dus bidirectioneel .

Voor de adresbus is de situatie enigszins anders. Enkel de CPU kan een adres op deze bus plaatsen. Andere componenten kunnen enkel een adres van de bus halen maar kunnen er zelf geen adres op plaatsen. De adresbus is dus een voorbeeld van een unidirectionele bus.

Bemerk dat één bit verstuurd kan worden over één lijn, één byte over 8 lijnen, één woord over 16 lijnen, etc. . De breedte van de databus bepaalt dus in grote mate de snelheid van de CPU. Opgelet, deze redenering gaat niet op bij het gebruik van een hoogfrequent kloksignaal.

[3] Arithmetic and Logical Unit
[4] Program Status Word
[5] General Purpose Registers

De breedte van de adresbus is dan weer bepalend voor de omvang van het geheugen. Zo kan de CPU met een adresbus van 8 lijnen 256 bytes geheugen adresseren, met 16 lijnen 64 KB en met 32 lijnen 4 GB, dit in de veronderstelling dat ieder adres één byte aan data bevat.

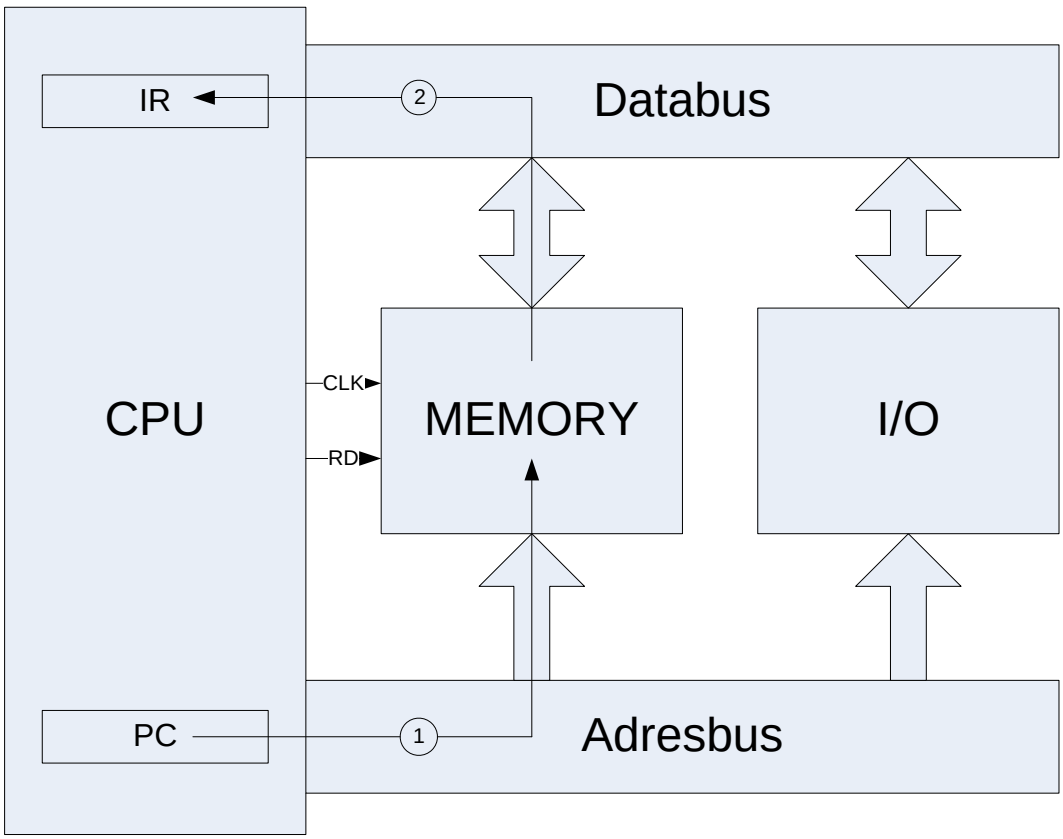
1.3.2 De Von Neumann cyclus

Voor het uitvoeren van het programma zelf zal de processor instructie per instructie uitvoeren volgens de Von Neumann cyclus. Deze cyclus beschrijft welke stappen de processor moet ondernemen om één instructie uit te voeren. In zijn meest eenvoudige vorm kent de Von Neumann cyclus 3 fasen.

De processor begint met het ophalen van de instructie uit het geheugen, de zogenaamde instruction fetch. De opgehaalde instructie wordt vervolgens gedecodeerd om o.a. te kunnen achterhalen welke processoronderdelen er bij het uitvoeren zullen moeten gebruikt worden. Tot slot volgt dan de execute fase die de instructie daadwerkelijk uitvoert.

Er dient te worden opgemerkt dat de meeste instructies nood hebben aan extra gegevens. Dit heeft tot gevolg dat bij het doorlopen van de Von Neumann cyclus er na de decodeerfase en voor de uitvoerfase een extra data fetch kan optreden.

1.3.2.1 De instruction fetch



Figuur 3: Instruction fetch

Bij de instruction fetch plaatst de CPU het adres van de volgende op te halen instructie, dat zich in de PC bevindt, op de adresbus(1). Het geheugen plaatst vervolgens de instructie die zich op dat adres bevindt op de databus(2). Tot slot haalt de CPU de instructie van de databus en plaatst deze in zijn instructieregister.

1.3.2.2 Instruction decoding

Na het ophalen van de instructie bevindt zich in het IR een reeks van eentjes en nulletjes die een instructie voorstellen. Aan de hand van dit bitpatroon zal de CPU trachten te achterhalen over welk soort instructie het gaat. Tijdens deze fase zal ook uitgemaakt worden welke GPR's zullen gebruikt worden.

1.3.2.3 Execute

Tijdens het uitvoeren van een instructie kunnen er tal van gebeurtenissen optreden. Een aantal van deze gebeurtenissen worden in het PSW, door het zetten van een eentje op een bepaalde bitpositie (cfr. vlaggetje) binnen dat register, bijgehouden. Indien aan de daaropvolgende instructie een voorwaarde gekoppeld is, zal de CPU, aan de hand van de inhoud van dit register, een beslissing nemen die bepalend zal zijn voor het verdere programmaverloop.

De meest voorkomende vlaggetjes zijn:

- De zero vlag voor het geval dat het resultaat nul is.
- De overflow vlag voor wanneer het resultaat groter is dan de breedte van een GPR.
- De carry-vlag. Indien je bijvoorbeeld twee 8-bit getallen samentelt kan dit een 9-bit getal opleveren. Als we aannemen dat de registerbreedte slechts 8 bit is worden de minst significante 8 bits van de som in een GPR gestopt en wordt het negende bit in het PSW opgeslagen. De carry-vlag dient dus voornamelijk als extra bit bij bepaalde rekenkundige bewerkingen.
- De negative-vlag. Deze vlag geeft aan dat het bekomen resultaat als een negatief getal moet worden aanzien. (cfr. tekenbit bij 2-complement voorstelling)
- …

Deze fase wordt beëindigd door de PC te laden met het adres van de volgende op te halen instructie. Opgelet, dit is niet noodzakelijk de inhoud van de PC vermeerderd met één!

1.4 Instructies

Iedere controller beschikt over een instructieset die hoort bij zijn architectuur. Doorgaans hebben controllers die ontwikkeld zijn door dezelfde firma een gelijkaardige instructieset.

Een instructie bestaat uit één of meerdere stukken. Het eerste stuk is steeds de opcode[6]. Dit bitpatroon identificeert de aard van de instructie. De andere stukken van de instructie zijn dan de operanden die bij deze opcode horen.

Aan iedere opcode wordt een assembler equivalente naam gekoppeld. Dit is veelal in de vorm van een drieletterwoord, of soms een vierletterwoord, waaraan de ontwikkelaar onmiddellijk kan zien wat het doel van deze opcode is. Zo'n korte benaming wordt een mnemonic genoemd. Voorbeelden hiervan zijn mov, add, mul, inc, dec, jnz, … .

Doorgaans zal een ontwikkelaar niet rechtstreeks met opcodes en operanden in aanraking komen maar zal hij zijn programma in assembleertaal coderen. Deze leesbare machinecode wordt dan door een assembleerder of cross-assembleerder omgezet naar echte machinecode.

Omdat het programmeren in assembleertaal vrij omslachtig is en er bovendien vrij veel kennis over de hardware-architectuur vereist is, beschikken de meer recente controllers over een mini Java virtuele machine. Hierdoor kan de ontwikkelaar met een beperkte kennis van de onderliggende hardware zijn programma's coderen in de programmeertaal Java. Sommige controllers beschikken zelfs over een eigen embedded operating systeem wat de gebruiksvriendelijkheid enkel maar ten goede komt.

1.5 Evaluatiekits

Naast losse microcontroller IC's kun je bij de meeste fabrikanten ook evaluatiekits aanschaffen. Zo'n kit bevat een printplaat met daarop de microcontroller IC samen met een aantal uitvoerpinnen waaraan de gebruiker extra periferiecomponenten kan koppelen. Bovendien hoort bij een evaluatiekit de nodige PC-software, meestal in de vorm van een IDE[7], om een programma te kunnen schrijven, assembleren en over te brengen naar het programmageheugen van de controller. De printplaat in kwestie moet dan wel fysisch, via een bepaalde interface, met de computer verbonden zijn. Deze kits zijn dan ook zeer handig voor het ontwikkelen van embedded software en kunnen ook worden gebruikt voor educatieve doeleinden.

Tijdens het labo hardware zullen we gebruik maken van een evaluatiekit, uitgerust met een C8051F120 microcontroller, van de firma Silicon Laboratories. Deze controller is gebaseerd op 8051 van Intel.

[6] Operation code
[7] Integrated Development Environment

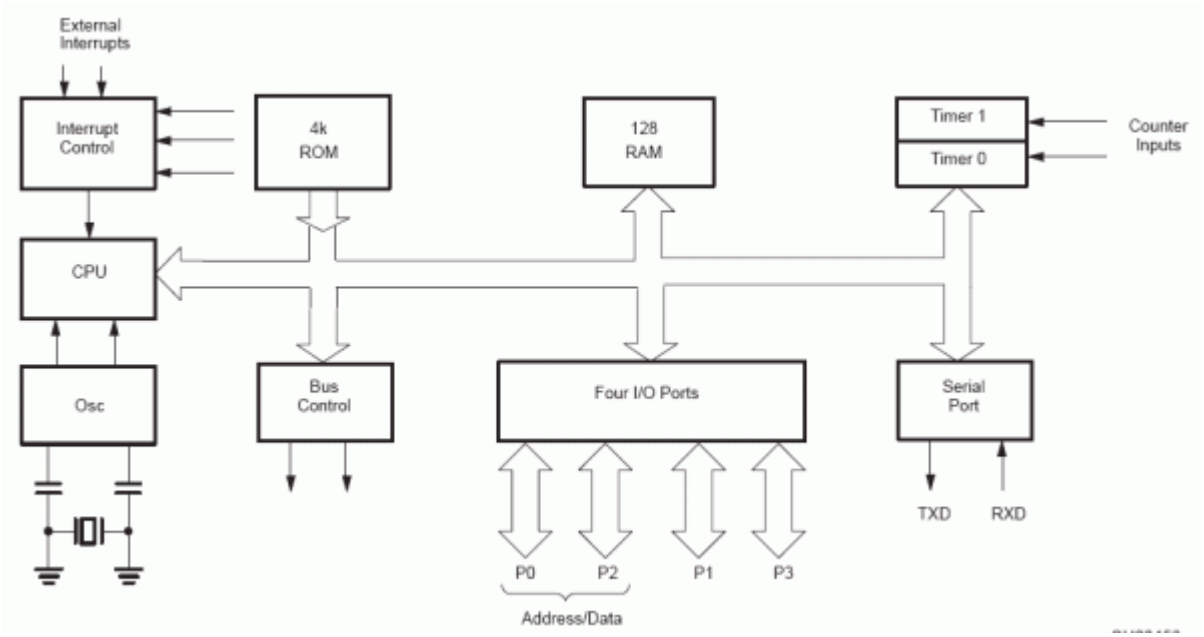
2 De 8051 microcontroller

2.1 Inleiding

De oorspronkelijke 8051 microcontroller is ontwikkeld en ontworpen door Intel. De eerste versies van deze controller werden begin jaren tachtig wereldwijd verspreid. Naast Intel hebben tal van andere bedrijven, waaronder Philips en Sony, 8051-gebaseerde controllers op de markt gebracht. Desondanks het grote succes van deze controller heeft men aan de oorspronkelijke architectuur tal van zaken moeten toevoegen om nu, na bijna drie decennia, nog steeds de concurrentie te kunnen aangaan met recentere controllers.

Bij embedded toepassingen horen microcontrollers die aan een aantal zeer specifieke criteria moeten voldoen. Het spreekt dus voor zich dat Intel een ganse reeks microcontrollers, gebaseerd op dezelfde architectuur, op de markt heeft gebracht. Deze familie kreeg de naam MCS-51™. De generieke versie van de MCS-51™ IC wordt de 8051 genoemd.

2.2 Blokdiagram van de 8051 microcontroller



Figuur 4: Vereenvoudigd blokschema van de 8051

De voornaamste eigenschappen zijn:

- CPU met uitgebreide instructieset (111 instructies ; bit- en byte-bewerkingen)
- 32 digitale GPIO-lijnen[8] verdeeld over vier 8-bitpoorten (P0, P1, P2 en P3).
- Twee timer/counters die in vier modes kunnen geprogrammeerd worden:
    - Mode 0: 13-bit timer/counter

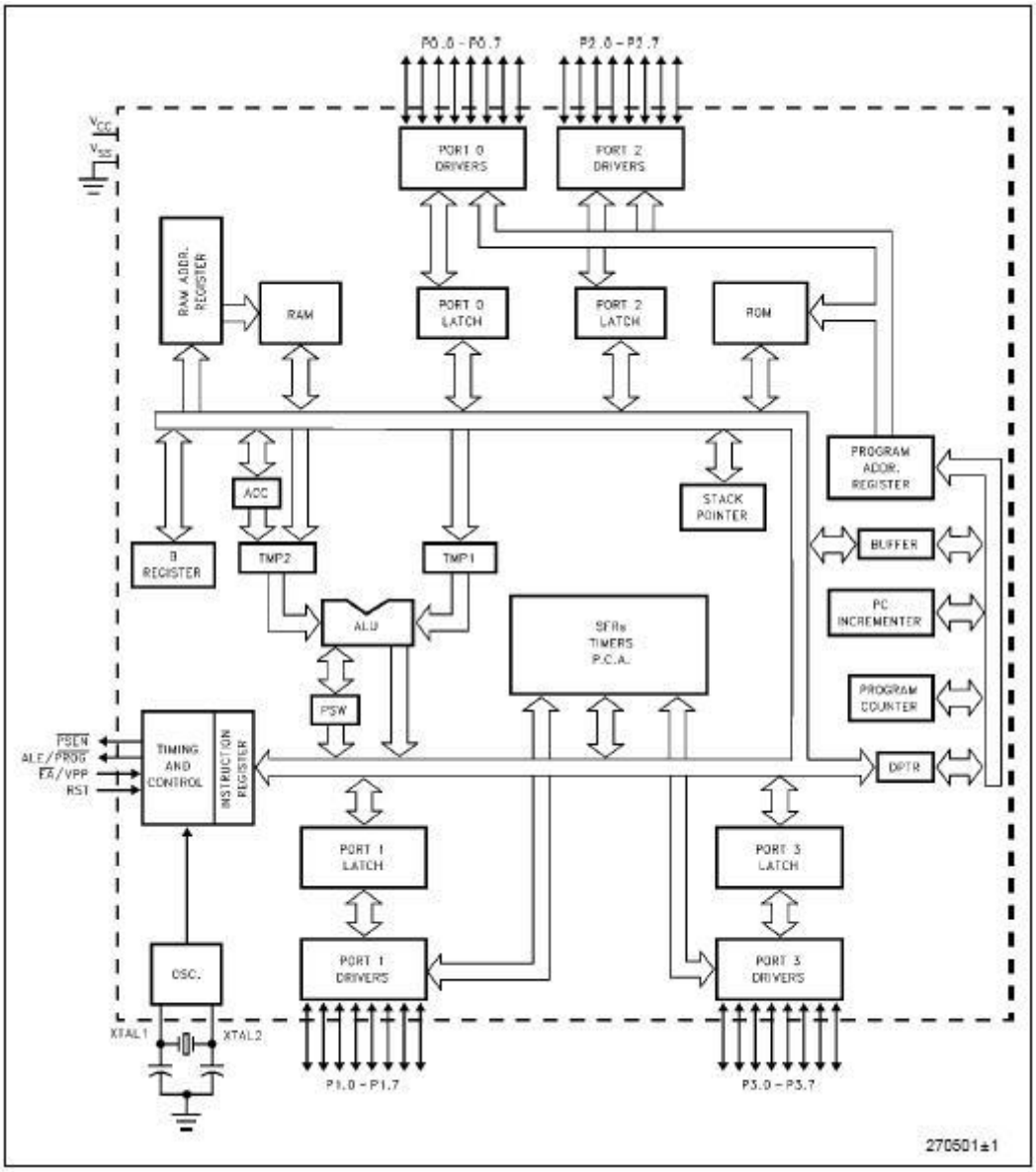
[8] General Purpose I/O

    - Mode 1: 16-bit timer/counter
    - Mode 2: 8-bit timer met autoreload
    - Mode 3: Bijzonder geval (opsplitsing naar drie timer/counters)
- Een full duplex serieel kanaal met instelbare baudrate. Voor het genereren van de baudrate zijn er diverse mogelijkheden
- Twee externe interruptlijnen. Naast deze externe interruptlijnen zijn er nog tal van interne interruptbronnen. Zo zijn er interne interrupts voorzien voor het serieel kanaal en de twee timers.
- 128 Bytes interne RAM waarvan een aantal adressen bitadresseerbaar zijn.
- 4 KB interne ROM. Naast deze versie zijn er tevens 8051-varianten die gebruik maken van EPROM en EEPROM.
- Uitbreidbaar met 64 KB datageheugen en 64 KB programmageheugen.
- …

Hierbij dient echter opgemerkt te worden dat de 8051-IC slechts over 40 aansluitpinnen beschikt. Indien we alle opgesomde eigenschappen echter gelijktijdig zouden benutten komen we uit op meer dan 40 aansluitpinnen. Sommige aansluitpinnen hebben dus een dubbele functie[9] waardoor slechts een deelverzameling van de hierboven opgesomde eigenschappen gelijktijdig kunnen gebruikt worden. Als we bijvoorbeeld opteren om extra datageheugen aan te sluiten dan zullen digitale I/O-poorten 0 en 3 niet kunnen gebruikt worden.

[9] Dit noemt men in het vakjargon multiplexing

2.3 Werking van de 8051 microcontroller



Figuur 5: Volledig blokschema van de 8051 microcontroller

Laat ons aan de hand van de eenvoudige instructie **mov A, 78H** de werking van de 8051 toelichten. Deze instructie vult de accumulator met de inhoud van adres 78H. Dit geheugenvakje bevindt zich, aangezien het om data gaat, in het interne datageheugen.

De PC plaatst het adres van de instructie in het PAR-register[10]. Vanuit het PAR-register wordt het adres over de interne adresbus verstuurd naar het interne programmageheugen. Het interne

[10] Program Address Register

programmageheugen stuurt de instructie die zich op het adres van de PC bevindt via de databus naar het instructieregister.

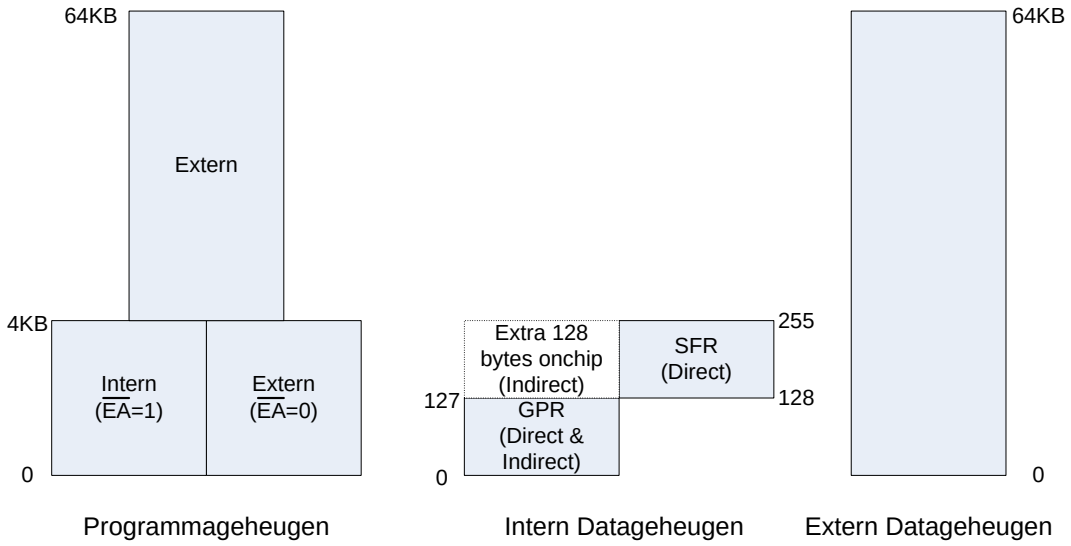
Een instructie kan afhankelijk van de gebruikte adresseermode één, twee of drie byte(s) in beslag nemen. Aangezien de oorsprong een direct adres is en de bestemming de accumulator wordt er gebruikt gemaakt van directe adressering. Dit heeft tot gevolg dat de instructie 2 bytes zal innemen, één byte voor de opcode en een tweede byte voor het direct adres. Om het direct adres op te halen wordt de PC vermeerderd met één en wordt opnieuw het intern programmageheugen geraadpleegd.

Na decodering van deze instructie blijkt dat een extra datafetch noodzakelijk is. Hiervoor wordt het adres 78H in het RAR-register[11] geplaatst. Dit adres wordt via de adresbus naar het intern datageheugen gestuurd waarna de bijhorende inhoud via de databus naar de accumulator zal worden gebracht.

Intern verloopt alle gegevensoverdracht over de gemeenschappelijke 8-bit adres/data-bus.

Wanneer er extern datageheugen en/of programmageheugen wordt toegevoegd dan blijkt uit figuur 5 dat poorten 0 en 2 zullen fungeren als data- en adresbus. De enige digitale I/O-poort die geen dubbele functie heeft is poort 1.

2.4 Geheugenmodel van de 8051 microcontroller



Figuur 6: Geheugenmodel van de 8051

Belangrijk is dat men steeds beseft dat men hier fysisch met vier verschillende geheugenblokken te maken heeft.

Er is enerzijds het interne programma- en datageheugen en anderzijds het al dan niet aanwezige externe programma en/of datageheugen. Een bepaald adres kan dus diverse malen voorkomen, maar steeds in een ander segment!

[11] RAM Address Register

Het intern programmageheugen omvat 4 KB ROM. Dit segment bevat het reset-vectoradres en de andere interruptvectoren.

Door het toevoegen van een extra geheugenchip kan de totale hoeveelheid programmageheugen worden uitgebreid tot 64 KB.

Het intern datageheugen omvat:

a) 128 bytes RAM (adressen 00H-7FH)

   Dit segment bevat het de 4 GPR-registerbanken, de stack en een aantal bitadresseerbare geheugenlocaties

b) Het Special Function Register blok (adressen 80H-FFH)

   Deze adressen worden voorbehouden voor de verschillende status- en stuurregisters van de I/O. Deze registers kunnen bitadresseerbaar zijn.

c) 128 bytes extra onchip RAM-geheugen (adressen 80H-FFH)

   De adressen binnen dit segment vallen samen met de adressen van het SFR-blok (zie figuur 6). Deze geheugenlokaties zijn uitsluitend te bereiken door indirect te adresseren (via een pointer). De adressen van het SFR-blok daarentegen kunnen alleen rechtstreeks worden benaderd.

Het extern datageheugen kan tot 64 KB oplopen. De adressering van het extern datageheugen kan via 8-bit en 16-bit adressering verlopen.

2.5 Overzicht van de verschillende registers

In tegenstelling tot de meeste microprocessoren hebben de registers van de 8051 een specifiek adres. Zij maken dus deel uit van het geheugen.

Ieder register kan dan ook op twee manieren worden geadresseerd, enerzijds door het rechtstreeks opgeven van het adres van het register en anderzijds door het opgeven van een symbolische naam. Het is aangewezen om bij het programmeren in assembleertaal, of zelfs in de programmeertaal C, gebruik te maken van de symbolische naam. De assembleerder of de compiler zal dan de symbolische namen vervangen door de corresponderende adressen.

Hieronder volgt een opsomming van de belangrijkste registers.

Symbool	Naam	Functie
Acc	Accumulator	Een belangrijk register zoals bij de meeste CISC-processoren. Veel instructies hebben dan ook betrekking op de accu. Voorstelling: A
B	B-register	Hulpregister bij het uitvoeren van vermenigvuldigingen en delingen.

PSW	Program Status Word	Statusregister van de 8051 ALU
SP	Stack Pointer	Wijst naar de laatst ingevulde positie van de stack(pre-increment). Bij het opstarten of bij een reset is de default waarde van SP 07H.
DPTR	Datapointer	Dit 16-bit register wordt gebruikt voor het indirect adresseren van het extern datageheugen.
P0, P1, P2, P3	I/O-poort registers	Deze registers worden gebruikt voor het inlezen en het wegschrijven van data naar een GPIO-poort of pin
IE	Interrupt enable control	Wordt gebruikt om in te stellen welke interrupts het programma kunnen onderbreken.
IP	Interrupt priority control	Bij het gelijktijdig optreden van twee interrupts zal dit register worden geraadpleegd om te bekijken welke van de twee interrupts het meest prioritair is.
TMOD	Timer mode register	Voor het instellen van de werkingsmodi van de 2 timers/counters.
TCON	Timer controle register	Statusregister van de twee timers/counters. Wanneer een timer overloopt zal in dit register een vlaggetje gezet worden om deze gebeurtenis vast te leggen.
TH0/TL0	Timer registers voor timer 0	Dit zijn de telregisters van timer/counter0. Er wordt geteld op het ritme van de klok (timer) of op het ritme van een extern signaal (counter)
TH1/TL1	Timer registers voor timer 1	Idem als voor timer 0

2.5.1 Layout van het Program Status Word

PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
CY	AC	F0	RS1	RS0	OV	NA[12]	P

[12] Not applicable. PSW.1 is gereserveerd!

- CY: Carry-vlag
- AC: Auxiliary carry. Dit vlaggetje wordt gezet wanneer er bij een optelling een overdracht plaatsvindt na de eerste vier bits.
- F0: De functie van dit vlaggetje kan door de gebruiker zelf worden ingevuld.
- RS1 en RS0: M.b.v. deze bits kan de programmeur instellen welke registerbank hij wil gebruiken. Er zijn in totaal 32 GPR-registers verdeeld over vier registerbanken van telkens acht registers. Op een gegeven moment kan er slechts één registerbank worden benut. Wanneer er dus nood is aan meer dan acht registers kunnen deze bits gemanipuleerd worden om het aantal registers uit breiden naar 16, 24 of zelfs 32.

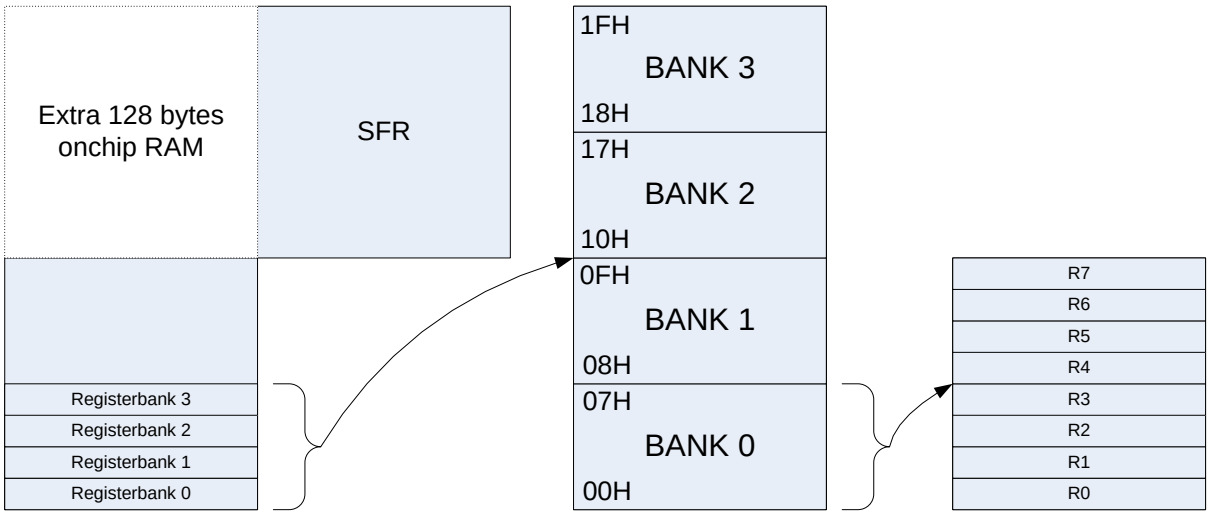
RS1	RS0	
0	0	Registerbank 0
0	1	Registerbank 1
1	0	Registerbank 2
1	1	Registerbank 3

- OV: overflow-vlag
- P: pariteit-vlag. Deze vlag wordt gezet wanneer de accumulator een oneven aantal enen bevat. Deze vlag is nul bij een even aantal enen.

Zoals blijkt zijn de zero en negative vlag niet beschikbaar maar ze zijn uiteraard wel aanwezig!

2.5.2 De General Purpose Registers

Zoals reeds eerder aangehaald beschikt de 8051 over vier registerbanken, met acht algemene registers per bank, waarvan er slechts één actief kan zijn. Deze algemene registers worden symbolisch voorgesteld door de namen R0, R1, R2, …, R7. Afhankelijk van de vlaggetjes RS1 en RS0 kunnen de symbolen R0, …, R7 elk overeenstemmen met vier verschillende adressen (zie figuur 7).



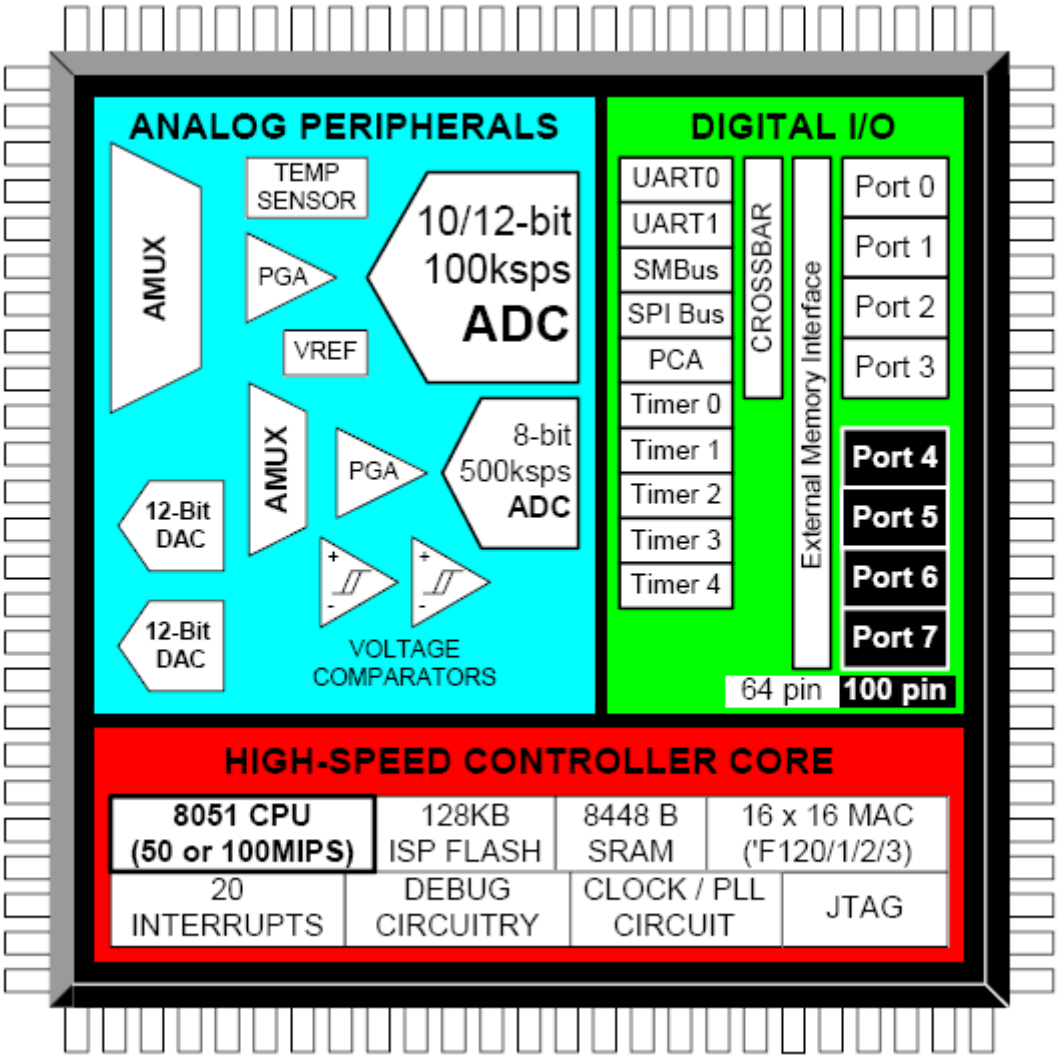
Figuur 7: Registerbanken van de 8051

Bij het opstarten van de controller of na een reset is steeds bank 0 actief.

2.6 De C8051F120 microcontroller

De C8051F120 microcontroller is een vrij recente microcontroller van Silicon Laboratories gebaseerd op de oorspronkelijke 8051 microcontroller van Intel. De CPU van de C8051F120, inclusief de instructieset, is dan ook volledig compatibel met de 8051. Uiteraard zijn de mogelijkheden van deze controller, in vergelijking met zijn voorvader, vrij uitgebreid.

2.6.1 Blokdiagram van de C8051F120

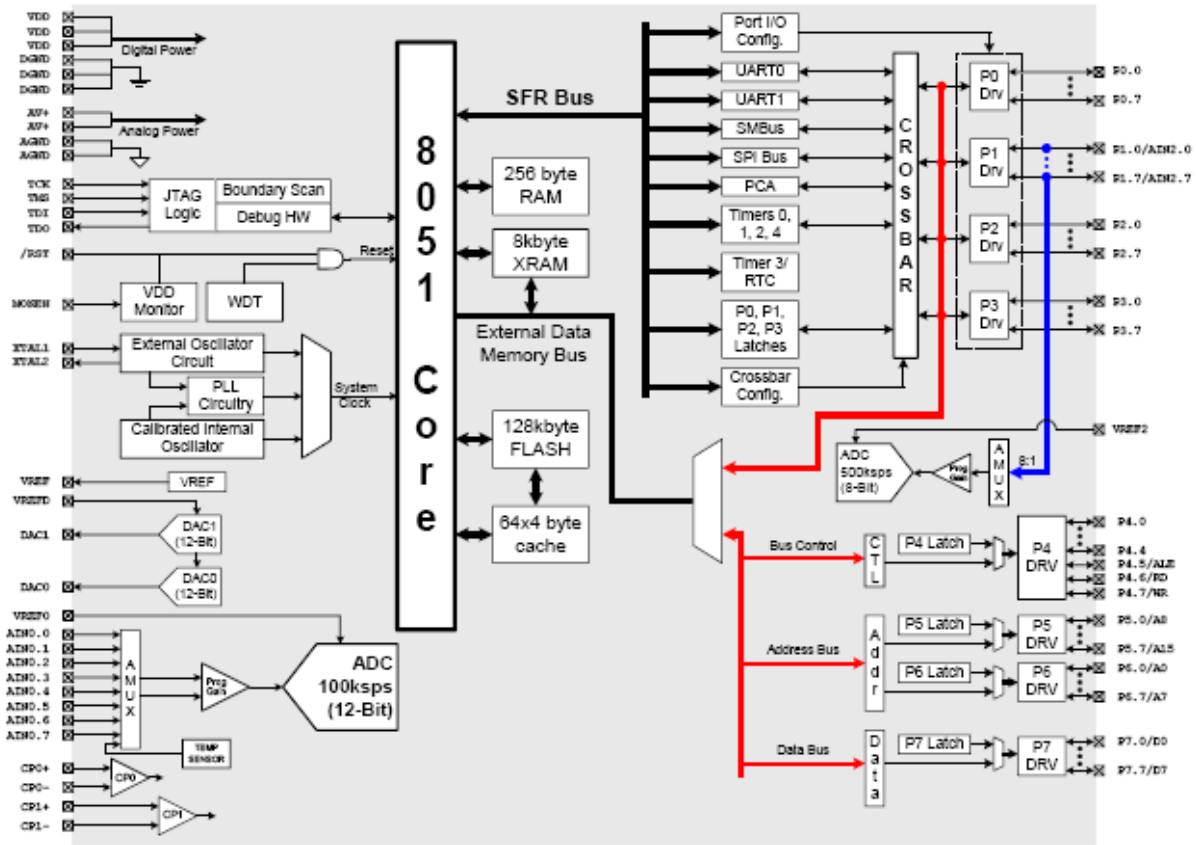


Figuur 8: Vereenvoudigd blokschema van de C8051F120

Volgens figuur 8 heeft de C8051F120 volgende features:

- Een 8051-compatibele CPU
- 128 KB onchip FLASH programmageheugen
- 8448 bytes (8 KB + 256 bytes) onchip datageheugen
- 20 interruptbronnen
- Een 24,5 MHz interne oscillator
- Wat analoge onchip periferie waaronder:
    - Een 10/12-bit en een 8-bit ADC[13]
    - 2 x 12-bit DAC's[14]
    - …
- Onderstaande Digitale I/O
    - 2 full duplex seriële kanalen met instelbare baudrate
    - 5 Timer/counters. De werking van timers 0 en 1 is volledig analoog aan de werking van timers 0 en 1 van de 8051.
    - 64 digitale GPIO-lijnen verdeeld over acht 8-bitpoorten (P0, P1, P2,…, P7)
    - …
- …

Figuur 9 toont het volledig blokschema van de C8051F120 controller.



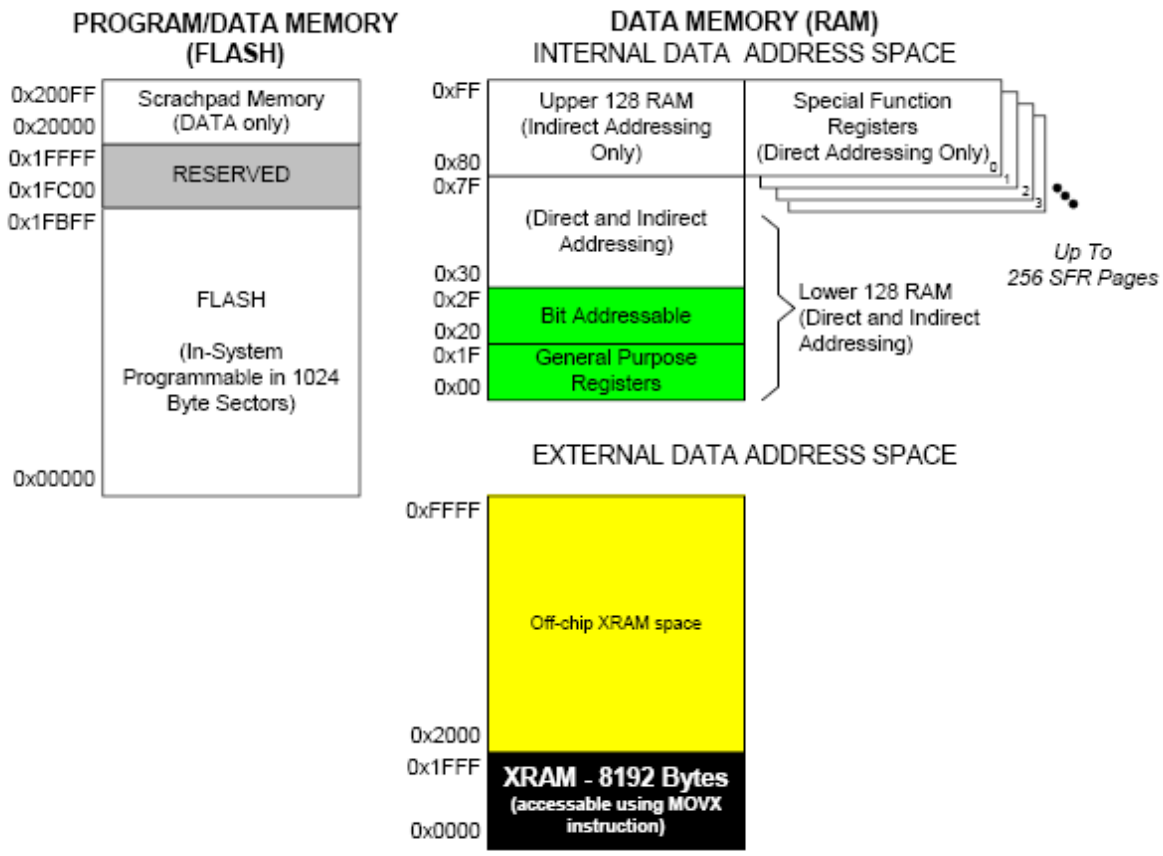
Figuur 9: Volledig blokschema van de C8051F120

[13] Analog to Digital Converter
[14] Digital to Analog Converter

Net zoals de oorspronkelijke 8051 heeft de C8051F120 meer mogelijkheden dan dat er aansluitingen voorzien zijn. Door het configureren van de digitale prioriteits crossbar kan de ontwikkelaar programmatorisch aanstippen welke functies er op een gegeven moment beschikbaar zijn.

Bij het aansluiten van extern geheugen zullen, volgens figuur 9, poorten 4, 5, 6 en 7 gebruikt worden. Hierbij zullen poorten 5 en 6 fungeren als adresbus, poort 7 als databus en poort 4 als controlebus. Eveneens volgens figuur 9 worden analoge ingangssignalen via de pinnen van digitale I/O-poort 1 naar de ADC gebracht.

2.6.2 Geheugenmodel van de C8051F120 microcontroller



Figuur 10: Geheugenmodel van de C8051F120

Indien we de geheugenmodellen van de C8051F120(figuur 6) en de 8051(figuur 10) naast elkaar leggen merken we op dat de verschillen vrij miniem zijn.

De 8448 bytes datageheugen zijn verdeeld over 4 segmenten:

- 128 bytes SRAM voor de 4 registerbanken en de stack. Bovendien bevinden zich in dit segment 16 bitadresseerbare geheugenlocaties.

- 128 bytes SRAM-geheugen waarvan de adressen overlappen met die van het SFR-blok. Indien er gebruikt gemaakt wordt van indirecte adressering komt men automatisch in dit segment terecht. Bij het gebruik van directe adressering kom je terecht in het SFR-blok.
- 8 KB intern RAM-geheugen dat overlapt met de adressen bestemd voor extern gebruik. Dit geheugen kan enkel en alleen worden bereikt door gebruik te maken van de movx instructie.

Net zoals vroeger bevat het SFR-blok alle stuur- en status registers van de CPU alsook van de I/O. Dit levert bij de C8051F120 wel problemen op omdat het aantal registers zo omvangrijk is dat 128 bytes niet meer volstaat om ze allemaal een uniek adres toe te kennen. Om dus enerzijds neerwaarts compatibel te zijn met de vroegere 8051 en anderzijds voldoende plaats te voorzien om alle registers een uniek adres toe te kennen heeft men het SFR-blok opgesplitst in 256 pagina's. Ieder adres binnen het SFR-blok komt dus 256 keer voor!

Dus in tegenstelling met de 8051 hoort bij een SFR niet alleen een adres maar tevens een SFR-pagina. Sommige registers zoals bijvoorbeeld P0, P1, P2, … komen voor in iedere pagina terwijl andere registers uitsluitend terug te vinden zijn in één pagina.

Alle verkeer van en naar het SFR-blok verloopt over een aparte bus, de SFR-bus. Bovendien bevinden de SFR's zich op diverse fysische plaatsen (zie figuur 9) waardoor het SFR-blok geen deel uitmaakt van de totale hoeveelheid datageheugen.

3 Instructieset van de 8051 microcontroller

3.1 Algemeen

Instructies bestaan uit opcodes en operanden die door de CPU uitgevoerd kunnen worden. Zo'n instructie kan één, twee of drie bytes programmageheugen in beslag nemen.

Bij de opbouw van een instructie wordt eerst de bestemming en pas dan de oorsprong vermeld.

Voorbeeld:

```
mov A,R1                    ; verplaats de inhoud van R1 naar de
                              accumulator
```

Om het onderscheid te maken tussen data en een adres zal men data steeds laten voorafgaan door een #-teken. Data of adressen kunnen binair, decimaal of hexadecimaal worden voorgesteld.

Voorbeelden:

```
mov A,#33h                  ; plaats de waarde 33 hexadecimaal in de accu
mov A,#33d                  ; plaats de waarde 33 decimaal in de
                              accumulator
mov A,#10100110b            ; plaats de waarde 10100110 binair in de
                              accumulator
mov A,33h                   ; de accu wordt geladen met de inhoud van
                              geheugenadres 33 hexadecimaal
```

Wanneer we het adres waarvan we de data willen ophalen laten aanwijzen door een pointer (R0, R1 of het DPTR-register), dan zal men dit register laten voorafgaan door een @-teken.

Voorbeeld:

```
mov A,@R0                   ; verplaats de inhoud van het adres dat zich in
                              het register R1 bevindt naar de accumulator
```

Bij de syntax van een instructie wordt een algemeen register dikwijls voorgesteld door R_r of R_i. Hierbij staat R_r voor een willekeurig algemeen register (R0,R1,…,R7) en R_i voor registers R0 of R1.

Omdat het omslachtig is de adressen van de verschillende SFR's op te zoeken, worden aan deze adressen symbolen toegekend die de functie van het register aangeven. Bij het assembleren van het programma zal de cross-assembler deze symbolen vervangen door het bijhorende adres. Ter herinnering: bij een SFR hoort een adres en een SFR-pagina. Niettegenstaande de assembler in staat is om het verband te leggen tussen symbool en adres kan hij niet de bijhorende SFR-pagina invullen. Het invullen van de SFR-pagina is dus de verantwoordelijkheid van de programmeur!

Voorbeelden:

```
mov A,P1                    ; verplaats de inhoud van digitale I/O-poort 1
                            naar de accumulator. Het opgeven van een
                            SFR-pagina is hier niet nodig omdat het register
                            P1 in iedere SFR-pagina voorkomt.

mov SFRPAGE,#00h            ; plaats de waarde 10 hexadecimaal in het
mov TMOD,#10h               register TMOD. Het register bevindt zich echter
                            in SFR-pagina 0 waardoor de eerste mov-
                            instructie noodzakelijk is.
```

Sommige SFR's en zelfs enkele gewone adressen zijn bitadresseerbaar. Wanneer een bepaalde bit moet geadresseerd worden, moet er gebruik gemaakt worden van een byte.bit notatie.

Voorbeelden:

```
mov C,26H.3                 ; plaats het vierde bit van adres 26H in het
                            carry-bit.
mov C, P1.1                 ; plaats de status van het tweede pinnetje van
                            digitale I/O-poort 1 in het carry-bit.
```

3.2 Adresseringsmodi

Zoals reeds aangehaald bepaalt de adresseringsmethode de lengte van de instructie. De 8051 kent volgende adresseringsmodi:

- Register adressering

  De 8051 beschikt over 8 algemene registers, R0 t.e.m. R7. Instructies die gebruik maken van register adressering worden verpakt in 1 byte. Hierbij worden 5 van de 8 bits gebruikt voor de functiecode terwijl de overige 3 bits benut worden voor het adresseren van het register.

  Voorbeeld:

  ```
  add A,R7                  ; tel de inhoud van R7 bij de inhoud van de
                            accumulator en sla de som op in de
                            accumulator.
  ```

  Deze instructie zal door de assembleerder vertaald worden naar 00101111. De meest significante 5 bits stellen de instructie voor en de 3 minst significante bits het registernummer.

- Directe adressering

  Met behulp van directe adressering kan ieder onchip geheugenvakje of SFR worden bereikt. Een instructie vergt bij deze modus 2 bytes, 1 byte voor de opcode en een tweede byte voor het direct adres.

  Voorbeeld:

  ```
  mov P1,A          ; De inhoud van de accumulator wordt naar digitale I/O-poort 1 gestuurd.
  ```

  De oorsprong, hier de accumulator, wordt impliciet in de opcode opgegeven. De binaire voorstelling van de volledige instructie is 10001001 10010000. De tweede byte stemt overeen met 90h, het adres van P1.

- Indirecte adressering

  Als we het DPTR-register even buiten beschouwing laten, kunnen enkel registers R0 en R1 gebruikt worden als pointervariabele. Net zoals bij registeradressering volstaat één byte om zo'n instructie te coderen. Deze byte bevat dan de functiecode en één bit voor het adresseren van R0 of R1.

  Voorbeeld:

  ```
  mov A,@R0         ; plaats de inhoud van het geheugenvakje waarnaar R0 verwijst in de accu.
  ```

- Immediate adressering

  Wanneer de oorsprong van een instructie een constante waarde voorstelt wordt er gebruik gemaakt van immediate adressering. Naast de opcode is er dan nood aan een tweede byte voor het opslaan van deze "immediate" data.

  Bij het gebruik van het DPTR-register zal de immediate data 16 bits, of twee bytes, in beslag nemen. De totale instructielengte komt dan op 3 bytes.

  Voorbeelden:

  ```
  mov A,#12d        ; schrijf de waarde 12 naar de accumulator
  mov DPTR,#8000H   ; de datapointer wordt geladen met de waarde 8000H
  ```

- Relatieve adressering

  Relatieve adressering wordt gebruikt bij sommige spronginstructies waar het adres van de volgende uit te voeren instructie zich niet al te ver van de huidige instructie bevindt. Bij deze modus wordt slechts één extra byte aan de opcode toegevoegd, i.e. het aantal posities dat bij de program counter moet geteld worden om de volgende instructie te vinden. Aangezien slechts één extra byte (met tekenbit) wordt gebruikt kan de volgende uit te voeren instructie zich maximum 127 posities verder of 128 posities terug in het programmageheugen bevinden.

  Voorbeeld:

  ```
  sjmp  verder                    ; springt onvoorwaardelijk naar het label
                                  verder.
  ```

  Het is niet ongebruikelijk dat men in assembleertaal werkt met een label i.p.v. een letterlijk adres. Dit heeft als voordeel dat de programmeur zich niet constant hoeft af te vragen op welk adres van het programmageheugen een bepaalde instructie zich bevindt. Afhankelijk van de instructie zal de assembleerder bij het omzetten van het programma de verschillende labels vervangen door een adres of zoals hier door het aantal bytes dat bij de program counter moet geteld worden.

- Absolute adressering

  Deze adresseringsmode wordt uitsluitend gebruikt bij de instructies ajmp en acall. Beide vergen een extra byte die het absolute adres bevat waarnaar gesprongen moet worden. De instructie ajmp voert een onvoorwaardelijke sprong uit naar dit adres terwijl de instructie acall een subroutine op dat adres begint uit te voeren.

  Net zoals bij relatieve adressering worden adressen veelal voorgesteld door labels.

  Voorbeeld:

  ```
  ajmp  verder                    ; springt onvoorwaardelijk naar het label
                                  verder.
  ```

  Het adres dat voorgesteld wordt door het label “verder” zit verstopt in het tweede byte. De opcode bevindt zich, zoals steeds, in het eerste byte.

- Long adressering

  Long adressering wordt enkel en alleen gebruikt bij de instructies ljmp en lcall. Deze methode is noodzakelijk voor doeladressen die bestaan uit 2 bytes.

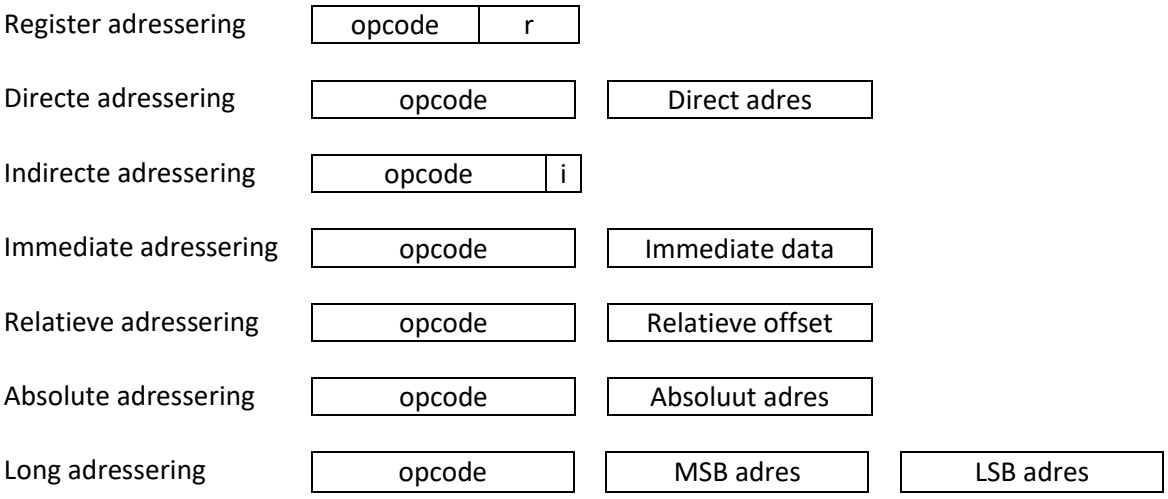
- Indexed adressering

  Bij Indexed adressering wordt in de instructie een adres gevonden dat eerst nog moet worden berekend. De inhoud van accu wordt samengeteld met de waarde van de PC of het DPTR-register.

  Voorbeeld:

```
movc A,@A+PC          ; de inhoud van de geheugencel met als adres de som van de accu en de program counter wordt in de accu geladen. Het doel is steeds een geheugencel uit het programmageheugen!
```

3.2.1 Samenvatting adresseringsmodi



3.3 Instructieset

Voor het bespreken van de instructieset zullen we volgende onderverdeling maken:

- Load/store instructies
- Logische bewerkingen
- Bitbewerkingen
- Rekenkundige bewerkingen
- Voorwaardelijke- en onvoorwaardelijke spronginstructies

3.3.1 Load/store instructies

Deze instructies worden gebruikt voor zowel intern datatransport als voor data-uitwisseling met de periferie te bewerkstelligen. De load/store instructies dragen informatie van de bron over naar de bestemming. De inhoud van het bestemmingsadres wordt daarbij overschreven.

3.3.1.1 Dataoverdracht naar de accumulator

a) Laden met een constante

Algemeen: **mov A,#constante**

Voorbeelden:

```
mov A,#20H      ;  De accumulator zal geladen worden met de waarde 20 hexadecimaal.
mov A,#11d      ; De accumulator zal na het uitvoeren van deze instructie de waarde 11 decimaal bevatten.
mov A,#10110011b ;De waarde 10110011 binair wordt in de accumulator geplaatst.
```

b) Laden met de inhoud van een adres

Algemeen: **mov A,dadr**

Voorbeeld:

```
mov A,22h       ; De inhoud van de geheugencel met adres 22 hexadecimaal wordt naar de accu weggeschreven.
```

c) Laden met de inhoud van een register

Algemeen: **mov A, R_r**

Voorbeeld:

```
mov A,R6        ; De inhoud van register R6 wordt naar de accu gekopieerd. De inhoud van register R6 blijft behouden.
```

d) Indirect laden via R0 en R1

Algemeen: **mov A,@R_i**

```
mov A,@R1       ; Verplaatst de inhoud van het adres dat zich in register R1 bevindt naar de accu.
```

3.3.1.2 Dataoverdracht van de accumulator

Bij een load/store-instructie kan de accumulator ook als bron opgegeven worden. De volgende varianten kunnen voorkomen:

mov R_r , A
mov dadr, A
mov @R_i , A

3.3.1.3 Dataoverdracht met een register

Bij een load/store instructie komt veelal een register voor. Dit is doordat een register bij de 8051 eigenlijk niks meer is dan een gewoon adres binnen het inwendig datageheugen met een eenvoudige symbolische naam.

Onderstaande varianten kunnen voorkomen:

mov R_r , A
mov R_r , dadr
mov R_r , #constante
mov A, R_r
mov dadr, R_r

3.3.1.4 Dataoverdracht met het inwendig datageheugen

Afhankelijk van de plaats in het inwendig datageheugen kan een geheugencel via directe en/of indirecte adressering worden bereikt.

a) Overdracht van een interne geheugencel

Bij directe adressering treft men volgende instructies aan:

mov A, dadr
mov R_r, dadr
mov dadr, dadr

Bij indirecte adressering zijn volgende varianten mogelijk:

mov A, @R_i
mov dadr, @R_i

Voorbeeld:

```
mov R0,#20H        ; Het adres waarnaar R0 verwijst wordt
                   ingesteld op 20 hexadecimaal.
mov A,@R0          ; De inhoud van adres 20 hexadecimaal wordt
                   via indirecte adressering naar de accumulator
                   gebracht.
```

b) Overdracht naar een interne geheugencel

Instructies met directe adressering:

mov dadr, A
mov dadr, R_r
mov dadr, #constante

Instructies met indirecte adressering:

mov @R_i, A
mov @R_i, dadr
mov @R_i, #constante

Voorbeeld:

```
mov R1,#20H      ; Het adres waarnaar R0 verwijst wordt ingesteld op 20 hexadecimaal.
mov A,#55H       ; De waarde 55 hexadecimaal wordt naar de accu gebracht.
mov @R1,A        ; De inhoud van de accu wordt naar het adres dat zich in R1 bevindt geschreven. De geheugencel met adres 20 hexadecimaal bevat de waarde 55 hexadecimaal.
```

3.3.1.5 Dataoverdracht met het externe datageheugen

Het extern datageheugen kan uitsluitend via de accumulator en d.m.v. indirecte adressering beschreven of gelezen worden.

Om het onderscheid te maken met de mov-instructie, die gebruikt wordt om interne geheugencellen te benaderen, gebruikt men voor het externe datageheugen de movx-instructie.

Met de registers R0 en R1 kan men slechts 256 bytes van de mogelijke 64 KB indirect adresseren. Adressen buiten het interval [0,255] moeten via een apart 16-bit pointerregister geadresseerd worden. Dit pointerregister wordt voorgesteld door DPTR[15].

De datapointer kan via volgende instructie geladen worden:

mov DPTR,#constante_16

Adressen beneden de 256 bytes kunnen zoals eerder vermeld ook via de registers R0 en R1 worden bereikt.

mov R1,#constante_8

a) Overdracht naar een externe geheugencel

movx @DPTR,A

Deze instructie draagt de inhoud van de accu over naar de externe geheugencel die door de datapointer aangewezen wordt.

[15] Datapointer

Voorbeeld:

```
mov DPTR,#6000H          ; Het adres 6000 hexadecimaal wordt
                           opgeslagen in de datapointer.

mov @DPTR, A             ; De inhoud van de accumulator wordt naar
                           adres 6000 hexadecimaal geschreven.
```

b) Gegevensoverdracht van het externe datageheugen

movx A,@DPTR

Deze instructie draagt de inhoud van de externe geheugencel, die door de datapointer aangewezen wordt, over naar de accumulator.

Voorbeeld:

Stel dat de inhoud van de externe geheugenplaats 5020 hexadecimaal via de pinnen van digitale I/O-poort 1 naar buiten moet worden gestuurd.

```
mov DPTR,#5020H          ; Het adres 5020 hexadecimaal wordt
                           opgeslagen in de datapointer.

mov A, @DPTR             ; De inhoud van adres 5020 hexadecimaal
                           wordt naar de accu gebracht.

mov P1, A                ; De inhoud van de accu, die gelijk is aan de
                           inhoud van geheugencel 5020 hexadecimaal
                           wordt naar poort 1 gestuurd.
```

Bemerk dat het onmogelijk is om de inhoud van adres 5020 hexadecimaal rechtstreeks naar poort 1 te schrijven. Enkel de accumulator kan bij een movx-instructie als bron of bestemming opgegeven worden!

3.3.1.6 Dataoverdracht met het programmageheugen

Bij een movc[16]-instructie zal men de accumulator laden met een code-byte of een constante uit het programmageheugen.

Algemeen:

movc A ,@A+DPTR
movc A ,@A+PC

a) Dataoverdracht via de datapointer

movc A,@A+DPTR

[16] Move constant

Hier zal de accu geladen worden met de inhoud van het adres dat bekomen wordt door de som te maken van de accu en de datapointer.

b) Dataoverdracht via de programmateller

movc A,@A+PC

De accumulator wordt geladen met de inhoud van het adres dat de som is van de inhoud van de accu en de inhoud van de programmateller na het uitvoeren van deze instructie.

3.3.1.7 Verwisselinstructies met de accumulator

Een verwisselinstructie is eigenlijk niks meer dan een klassieke mov-instructie maar dan wel in twee richtingen. Data wordt van de bron naar de bestemming gestuurd en omgekeerd.

Algemeen:

xch A, R_r
xch A,dadr
xch A,@R_i

Voorbeeld:

xch A, R6

Voor het uitvoeren	Na het uitvoeren
Accu=56H	Accu=07H
R6=07H	R6=56H

3.3.1.8 Bewerkingen op de stack

Onderstaande instructies sturen de inhoud van een direct adres naar de stack (push) of sturen de inhoud waarnaar de stackpointer wijst naar een direct adres (pop).

Algemeen:

push dadr
pop dadr

De stack wordt bij de 8051 geïmplementeerd als een pre-increment (of post decrement) LIFO-gegevensstructuur. De stackpointer (SP-register) wijst dus naar de laatst opgevulde plaats.

Bij het opstarten van de controller bevat het register SP de waarde 07H wat betekent dat de eerste vrije plaats (08H) van de stack samenvalt met register R0 van registerbank 1. Om te vermijden dat de stack overlapt met een aantal registers is het dus aangewezen om bij aanvang van het uit te voeren programma de stackpointer een andere waarde te geven.

Opgelet: Gebruik niet push A maar wel push ACC als de inhoud van de accu op de stack moet worden geplaatst!

3.3.2 Logische bewerkingen

De 8051 microcontroller ondersteunt volgende logische operaties:

- AND
- OR
- EXOR

Naast bovenstaande instructies bestaan er een aantal instructies die enkel van toepassing zijn op de accumulator. Dit zijn clear, complement, swap en alle rotatie-instructies.

3.3.2.1 AND (ANL)

A	B	Q
0	0	0
1	0	0
0	1	0
1	1	1

De volgende instructies worden ondersteund:

anl A, R_r
anl A, dadr
anl A, @R_i
anl A,#constante
anl dadr, A
anl dadr, #constante

Voorbeeld:

Aan digitale I/O-poort 1 zijn 4 LED's aangesloten. Om de toestand van de LED's te achterhalen zijn we enkel geïnteresseerd in de minst significante nibble (= 4 bit) van poort 1.

```
mov A,P1          ; Verplaats de inhoud van digitale I/O-poort 1
                  naar de accumulator.
anl A,#00001111b  ; Door het aanbrengen van het binair masker
                  00001111 wordt de meest significante nibble
                  weggeknipt.
```

3.3.2.2 OR (ORL)

A	B	Q
0	0	0
1	0	1
0	1	1
1	1	1

Analoog aan de anl-instructie worden volgende varianten van de orl-instructie ondersteund:

orl A, R_r
orl A, dadr
orl A, @R_i
orl A,#constante
orl dadr, A
orl dadr, #constante

Voorbeeld:

Aan digitale I/O-poort P0 moet de informatie van digitale I/O-poort P1 naar buiten gebracht worden. Onafhankelijk van de toestand van poort 1 moet I/O-pin P0.1 de logische waarde 1 hebben.

```
mov A,P1          ; Verplaats de inhoud van digitale I/O-poort 1
                  naar de accumulator.
orl A,#00000010b  ; het eerste bit wordt op 1 gezet.
mov P0,A          ; stuur de inhoud van de accu naar digitale I/O-
                  poort 0.
```

3.3.2.3 EXOR (XRL)

A	B	Q
0	0	0
1	0	1
0	1	1
1	1	0

Net zoals de instructies anl en orl zijn de volgende instructies beschikbaar:

xrl A, R_r
xrl A, dadr
xrl A, @R_i
xrl A,#constante
xrl dadr, A
xrl dadr, #constante

3.3.2.4 Clear accumulator

Algemeen:

clr A

De instructie clr A reset de accumulator.

3.3.2.5 Complement accumulator

Algemeen:

cpl A

De instructie cpl A inverteert de inhoud van de accumulator bit per bit.

3.3.2.6 Swap accumulator

Algemeen:

swap A

Swap A verwisselt de laagste nibble van de accu met de hoogste nibble.

Voorbeeld:

Voor het uitvoeren	Na het uitvoeren
Accu=27H	Accu=72H

3.3.2.7 Rotaties

a) RL A

Door het uitvoeren van deze instructie worden alle bits van de accumulator 1 plaats naar links verschoven. Het meest significante bit van accu wordt het minst significante bit.

Bemerk dat door het uitvoeren van deze instructie de inhoud van de accu wordt vermenigvuldigd met 2!

b) RR A

Met deze instructie worden alle bits van de accu 1 plaats naar rechts verschoven. Het minst significante bit wordt het meest significante bit.

Het uitvoeren van deze instructie heeft als gevolg dat de inhoud van de accu gedeeld wordt door 2!

c) RLC A

Net zoals bij RL A worden alle bits van de accu 1 plaats naar links verschoven maar wordt het meest significante bit naar de carry-vlag van het PSW gestuurd. De inhoud van de carry komt terecht in het minst significante bit van de accumulator.

d) RRC A

Deze instructie roteert alle bits van de accu in wijzerzin, doorheen de carry-vlag.

De instructies RL (rotate left) en RR (rotate right) zijn zeer handig wanneer er moet vermenigvuldigd of gedeeld worden door/met een macht van 2.

3.3.3 Bitbewerkingen

De 8051 ondersteunt tal van instructies om individuele bits te manipuleren.

De rol die de accumulator vervult bij de overeenkomstige bytebewerkingsinstructies wordt hier ingenomen door de carry-vlag.

a) Bit verplaatsingen

mov bit, C
mov C, bit

b) Set en reset van een bit

setb C
setb bitadr

clr C
clr bitadres

c) Logische bewerkingen op bits

Bij deze instructies zal de carry-vlag van de 8-bit microprocessor als accumulator gebruikt worden voor de 1-bit processor. Voor één van de operanden, evenals voor het wegschrijven van het resultaat van de bewerking wordt gebruik gemaakt van de carry-vlag.

anl C,bitadr
anl C,/bitadr

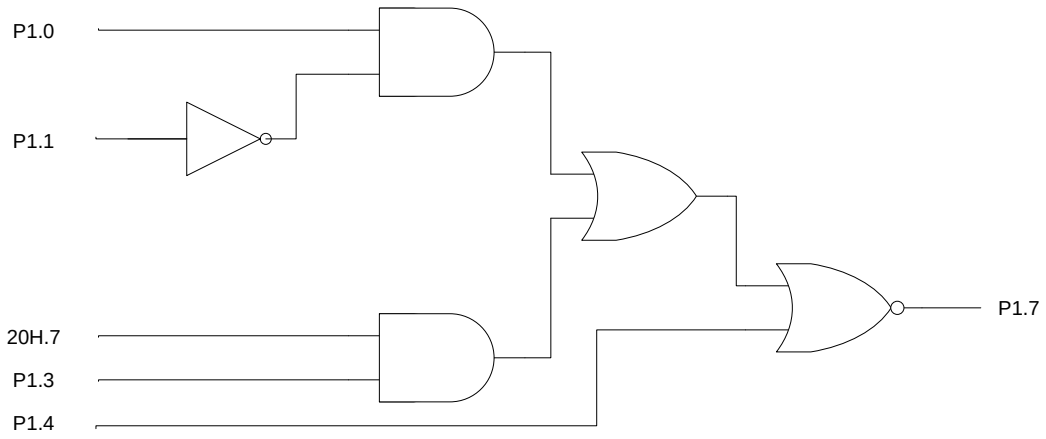
orl C,bitadr
orl C,/bitadr

Het /-teken betekent dat de geinverteerde bit moet genomen worden.

d) Complementbewerking

cpl C
cpl bitadr

3.3.3.1 Voorbeeld bitbewerkingen



Figuur 11: Logische schakeling

Gevraagd:

Schrijf een programmafragment dat bovenstaande logische schakeling softwarematig realiseert.

Oplossing:

```
Loop:   mov C, P1.0
        anl C,/P1.1
        mov F0, C          ; F0 uit het PSW wordt gebruikt voor tijdelijke opslag

        mov C,20H.7
        anl C, P1.3
        orl C, F0
        orl C,P1.4
        cpl C
        mov P1.7, C
        jmp Loop
```

Er dient opgemerkt te worden dat de 8051 enkel en alleen een logische 0 kan inlezen. Indien er niks aan een bepaalde poortpin wordt aangesloten heeft deze poortpin de logische waarde 1.

3.3.4 Rekenkundige bewerkingen

Met de 8051 microcontroller kunnen volgende rekenkundige bewerkingen uitgevoerd worden:

- Optellen
- Aftrekken met borrow
- Vermenigvuldigen
- Delen
- Verhogen met 1 (incrementeren)
- Verlagen met 1 (decrementeren)

Alle rekenkundige bewerkingen zullen invloed hebben op de overflow-, carry-, hulp-carry en pariteitsvlag.

De gebruikte afkortingen zijn:

- CY: carry flag
- AC: auxillary carry flag
- OV: overflow flag
- P: parity flag

Enkele bemerkingen:

- De carry-vlag zal men gebruiken bij optellingen, aftrekkingen en rotaties. De carry-vlag wordt gezet als 9de bit bij optellingen.
- De overflow-vlag is enkel van belang wanneer we gaan optellen en aftrekken van tweecomplement gehele getallen. In het tweecomplement maken we volgende verdeling:
    - Positieve getallen [00H,7FH]
    - Negatieve getallen [80H,FFH]

  De overflow-vlag wordt gezet wanneer het resultaat van een berekening buiten het bereik valt.

```
   50H   Positief getal
+  61H   Positief getal
   B1H   Negatief getal => OV=1
```

- De hulp-carry wordt gebruikt bij BCD-bewerkingen. De AC-vlag wordt gezet wanneer er een overdracht plaatsvindt van het vierde naar het vijfde bit. (Of beter van de minst significante nibble naar de meest signifante nibble)

```
   0FH
+  31H
   40H   AC=1
```

- De pariteitsvlag wordt gezet wanneer het resultaat een oneven aantal 1-bits telt.

3.3.4.1 Optelling

a) Optelling zonder overdracht

Algemeen:

add A, R_r
add A, dadr
add A,@R_i
add A,#constante

Beïnvloede vlaggen: CY, AC, OV, P

b) Optelling met overdracht

Algemeen:

addc A, R_r
addc A, dadr
addc A,@R_i
addc A,#constante

Deze instructie telt de tweede operand en de inhoud van de carry op bij de accumulatorinhoud. Het resultaat komt terecht in de accumulator.

3.3.4.2 Aftrekking

Hier bestaat er slechts één type aftrekking, nl. een aftrekking met borrow. Indien er geen rekening moet gehouden worden met de borrow (of ontleenbit) dan moet de carry-vlag vooraf gewist worden. Dit kan door de aftrekking te laten voorafgaan door de instructie clr C.

Algemeen:

subb A, R_r
subb A, dadr
subb A, @R_i
subb A, #constante

3.3.4.3 Vermenigvuldiging

Algemeen:

mul AB

De instructie mul zal de inhoud van de accumulator vermenigvuldigen met de inhoud van het B-register. Het resultaat, dat uit 2 bytes bestaat, wordt terug in A en in B geschreven. De minst beduidende byte komt terecht in A en de meest beduidende byte in B.

Functie van de verschillende vlaggen:

- Wanneer het resultaat groter is dan FF zal de OV-vlag op 1 gezet worden.
- De hulp-carry wordt niet beïnvloed.
- De carry-vlag wordt steeds op 0 gezet.
- De pariteitsvlag wordt wel gezet en is afhankelijk van de inhoud van de accumulator.

3.3.4.4 Deling

Algemeen:

div AB

Deze instructie deelt de inhoud van de accumulator door de inhoud van het B-register. Het quotiënt komt in A terecht en de rest van de deling in B.

Functie van de vlaggen:

- Bij een deling door 0 komt er “rommel” in A en B, terwijl OV gezet wordt.
- CY wordt bij een deling door 0 gezet.
- AC blijft onveranderd.

3.3.4.5 Verhogen en verlagen met 1

Algemeen:

inc A
inc R_r
inc dadr
inc @R_i
inc DPTR

dec A
dec R_r
dec dadr
dec @R_i

Opmerking: **dec DPTR** bestaat niet!

3.3.5 Spronginstructies

We zullen hier een onderscheid maken tussen:

- Onvoorwaardelijke spronginstructies
- Voorwaardelijke spronginstructies

3.3.5.1 Onvoorwaardelijke spronginstructies

Afhankelijk van de adresseermode bestaan volgende onvoorwaardelijke spronginstructies:

ajmp adres
ljmp adres
sjmp rel
jmp adres

Bij het gebruik van de synthetische jmp-instructie zal de assembleerder zelf een keuze maken uit ajmp, ljmp of sjmp.

Veelal wordt het adres nooit rechtstreeks opgegeven maar wordt er gewerkt met labels. Deze labels zullen dan bij het omzetten naar machinecode omgezet worden naar adressen binnen het programmageheugen.

Voorbeeld:

jmp $	; het $-teken slaat op het adres waar deze instructie zich bevindt. Dit komt er dus op neer dat deze instructie een oneindige lus veroorzaakt.

3.3.5.2 Voorwaardelijke spronginstructies

a) Testen op de carry-vlag

Algemeen:

jc rel	Voer de sprong uit wanneer de carry gezet is.
jnc rel	Voer de sprong uit als de carry niet gezet is.

Deze instructies zijn zeer handig wanneer gebruik gemaakt wordt van de rotatiefuncties RRC en RLC. Ook bij het gebruik van logische bewerkingen en/of bitbewerkingen zijn deze spronginstructies handig om de programmaflow te beïnvloeden.

b) Testen van bits

Algemeen:

jb bitadr,rel	Voer de sprong uit wanneer het bit gezet is.
jnb bitadr,rel	Voer de sprong uit als het bit niet gezet is.

De instructies jb en jnb zijn eigenlijk de veralgemeende vormen van jc en jnc. Beide instructies worden veel gebruikt om te wachten tot wanneer een bepaalde bit gezet wordt.

Voorbeeld:

```
jnb P1.4,$          ; Het programma zal wachten tot wanneer P1.4 1 wordt.
```

c) Testen van de accumulator op nul

Algemeen:

jz rel	Voer de sprong uit wanneer de inhoud van de accu gelijk is aan 0.
jnz rel	Voer de sprong uit als de accu een waarde bevat verschillend van 0.

d) Decrement jump if not zero (DJNZ)

Algemeen:

djnz R_r, rel
djnz dadr, rel

De instructie djnz R_r, rel decrementeert eerst het aangegeven register en test dan of de inhoud ervan nul geworden is. Wanneer de inhoud van dat register een waarde heeft die verschillend is van 0, wordt de sprong genomen.

Deze instructie wordt veelal gebruikt als wachtlus, vertragingslus of als assembler equivalent van de klassieke for-lus.

Voorbeeld:

```
mov R0,#10d         ; plaats de waarde 10 in het register R0.
djnz R0,$           ; wacht 10 klokcycli vooraleer het programma verder uit te voeren.
```

e) Compare and jump if not equal (CJNE)

Algemeen:

cjne A,#constante,rel
cjne A, dadr, rel
cjne R_r,#constante,rel
cjne @R_i,#constante, rel

De instructie CJNE vergelijkt de inhoud van de accu, register of de pointerinhoud met een constante en neemt de sprong indien beide waarden van elkaar verschillen. Inwendig voert de microcontroller de vergelijking uit door een aftrekking te maken terwijl de operanden onveranderd blijven en alleen de vlaggen beïnvloed worden overeenkomstig met de aftrekking.

3.3.6 Subroutines

Net zoals bij hogere programmeertalen kan redundante code vermeden worden door het gebruik van functies en/of procedures. Procedures in assembleertaal worden subroutines genoemd.

Een subroutine wordt aangeroepen via de instructie LCALL of ACALL. Wanneer je gebruikt maakt van de synthetische instructie CALL zal de assembleerder zelf een keuze maken uit LCALL of ACALL.

Alle instructies die betrekking hebben op het aanroepen van subroutines vergen slechts één parameter, i.e. het adres waar de subroutine begint.

Algemeen:

lcall adres_16
acall adres_11
call adres

Iedere subroutine wordt afgesloten met de instructie RET. Deze instructie zorgt ervoor dat de programmateller geladen wordt met het adres van de instructie die net na de call-instructie uitgevoerd moet worden.

Bij het uitvoeren van een call-instructie wordt het terugkeeradres (2 bytes) op de stack geplaatst. Na het uitvoeren van de ret-instructie wordt het terugkeeradres in de program counter geladen.

3.3.6.1 Interrupt verwerking

Wanneer subroutines vanuit de hardware aangeroepen worden, spreekt men van interrupts. De subroutine die uitgevoerd moet worden bij het optreden van een specifieke interrupt moet zich op een vaste plaatst in het programmageheugen bevinden. Zo kan men stellen dat bij het optreden van een interrupt de microcontroller precies weet waar hij de bijhorende subroutine moet gaan zoeken.

Net zoals een gewone subroutine moet een interrupt service routine afgesloten worden met een return-instructie. Om het verschil duidelijk te maken met gewone subroutines worden interrupt service routines beëindigd door een RETI[17]-instructie.

3.4 Verband tussen een instructie en een vlag

De instructies waarbij een vlag van toestand kan veranderen zijn hieronder weergegeven:

Instructie	Vlag			Instructie	Vlag		
	CY	OV	AC		CY	OV	AC
add	X	X	X	setb C	1		
addc	X	X	X	cpl C	X		
subb	X	X	X	clr C	0		
mul	0	X		anl C,bit	X		
div	0	X		orl C, bit	X		
rrc	X			mov C,bit	X		
rlc	X			cjne	X		

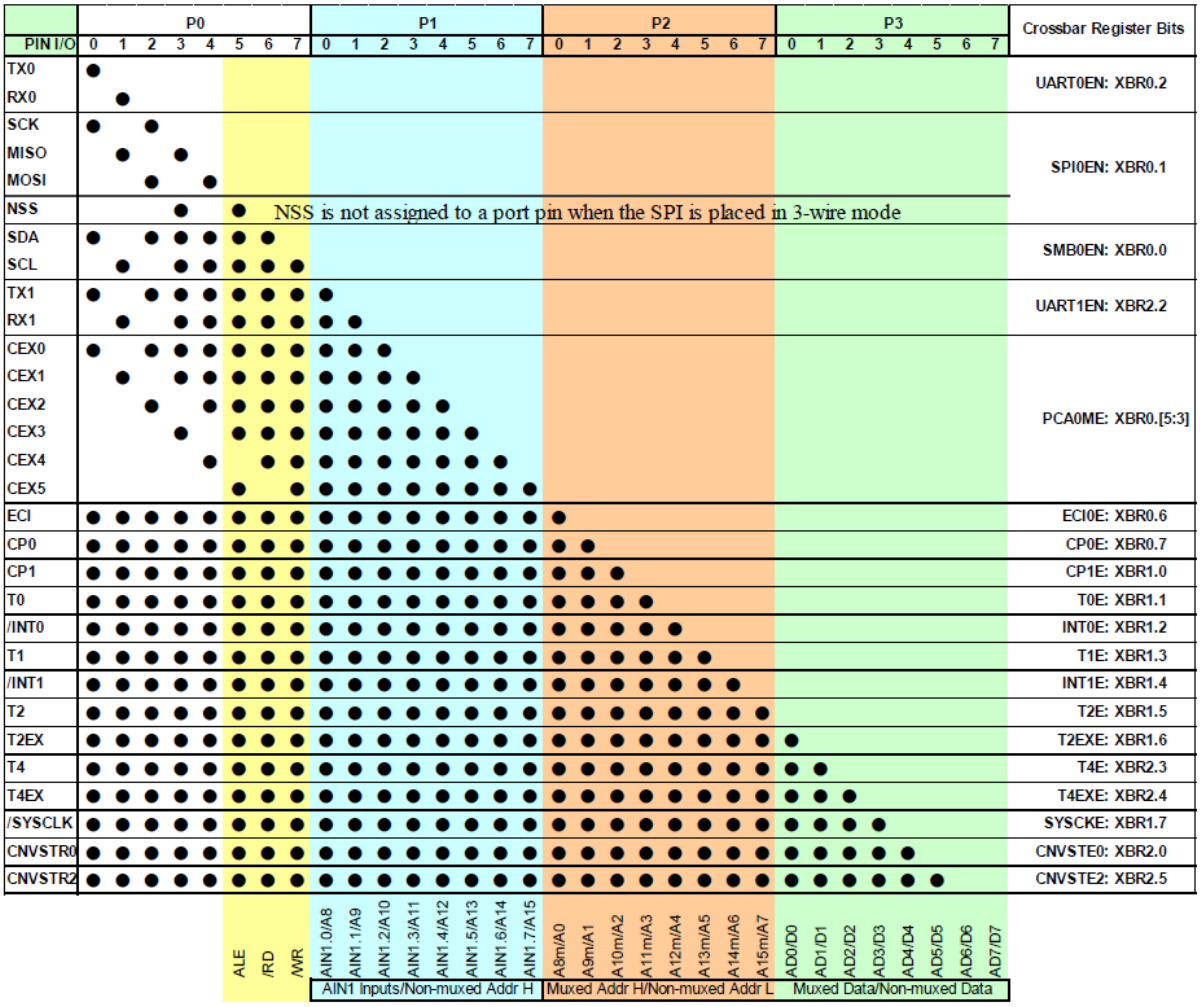
[17] Return from interrupt

4 Digitale I/O-mogelijkheden van de C8051F120 microcontroller

4.1 De digitale prioriteitscrossbar

Door een tekort aan aansluitpinnen is het onmogelijk om alle functies van de controller op een gegeven moment te benutten. Welke signalen naar buiten gebracht worden, wordt bepaald door de configuratie van de digitale prioriteitscrossbar.

De crossbar kan softwarematig worden geconfigureerd door de registers XBR0, XBR1 en XBR2. Figuur 12 toont de decodeertabel van de prioriteitscrossbar.



Figuur 12: De digitale prioriteitscrossbar

Figuur 12 toont welke registerbits er moeten gezet worden om bepaalde functies naar buiten te brengen. Bovendien wordt getoond welke poortpinnen daarbij zullen gebruikt worden (enkel poortpinnen van GPIO-poorten 0,1,2 of 3 komen in aanmerking)

Welke poortpinnen toegekend worden is afhankelijk van de prioriteit van de functie. Volgens figuur 12 is UART0, het eerste serieel kanaal, het meest prioritair en het CNVSTR2-bit, het minst prioritair. Indien dus UART0 beschikbaar moet zijn dan zijn pinnen P0.0 en P0.1 niet beschikbaar voor andere toepassingen.

Bijvoorbeeld:

Schrijf een programmafragment waarbij de digitale crossbar ingesteld wordt zodat het eerste serieel kanaal, het tweede serieel kanaal en de externe interruptlijn 0 beschikbaar wordt.

```
…
mov SFRPAGE,#0FH        ; alle configuratieregisters van de digitale
                        crossbar bevinden zich in SFR-pagina F.
mov XBR2,#40H           ; de crossbar wordt aangezet.
mov XBR0,#04H           ; UART0 naar buiten brengen.
mov XBR2,#04H           ; idem voor UART1
mov XBR1,#04H           ; idem voor externe interruptlijn 0
…
```

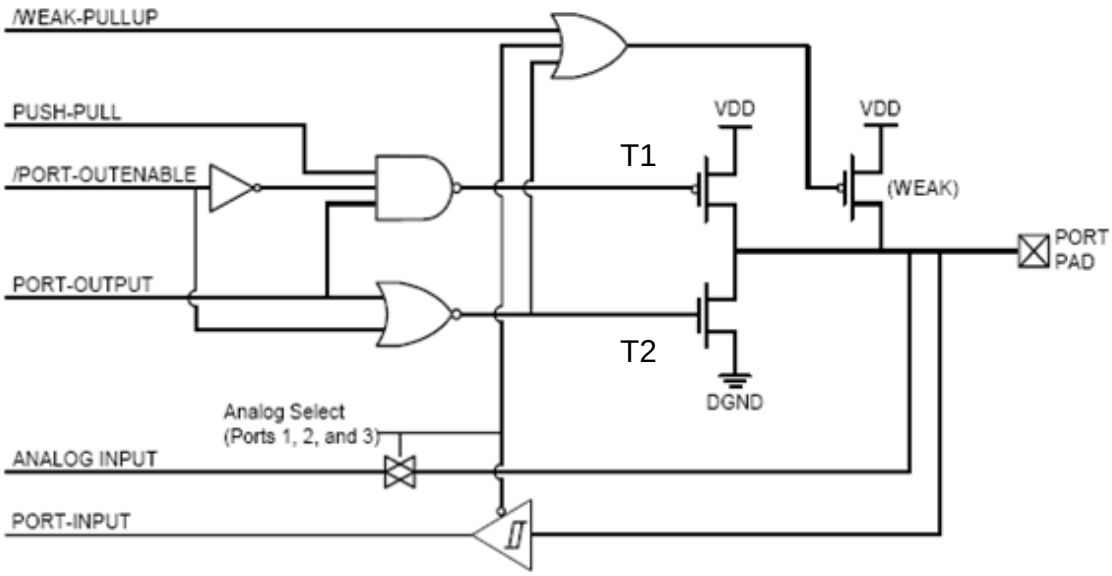
Opmerkingen:

- De instructie **mov XBR2,#40H** zet de crossbar aan. Deze instructie zal bijna in ieder programma voorkomen omdat het weglaten ervan GPIO- poorten 0,1,2 en 3 blokkeert.
- De instructies mov XBR2,#40H en mov XBR2,#04H kunnen samengenomen worden tot mov XBR2,#44H
- De pinverdeling wordt als volgt:

UART0	TX=P0.0	RX=P0.1
UART1	TX=P0.2	RX=P0.3
/INT0	P0.4	

4.2 Structuur van de digitale I/O-poorten

Een poortpin ziet er inwendig als volgt uit:



Figuur 13: Structuur digitale I/O pin

Voor het uitschrijven van digitale waarden wordt gebruikgemaakt van 2 transistoren.

Indien transistor T1 in geleiding staat, dan wordt V_{DD} doorverbonden met de poortpin. In het andere geval staat transistor T2 in geleiding waardoor de poortpin rechtstreeks met de massa verbonden is. Om kortsluiting te vermijden zullen beide transistoren nooit gelijktijdig geleiden.

a) Poortpin als uitvoer

Wanneer een I/O-pin als uitvoer geconfigureerd is, dan is /PORT-OUTENABLE gelijk aan 0.

Bij het uitschrijven van een logische 0 wordt de datalijn PORT-OUTPUT op een logische 0 gebracht. Hierdoor komt de GATE van transistor T2 op een logische 1 waardoor de transistor in kwestie begint te geleiden. Transistor T1 geleidt niet omdat de GATE via de uitvoer van de NAND-poort op een logische 1 komt te staan en deze transistor pas begint te geleiden bij een logische 0 (bemerk het inversie-symbool).

Voor het uitschrijven van een logische 1 kan dezelfde redenering gevolgd worden. Echter om transistor T1 te laten geleiden moet de PUSH-PULL lijn een logische 1 bevatten.

b) Poortpin als invoer

Om te kunnen inlezen moeten beide uitgangstransistoren gesperd worden.

Indien een poortpin als invoer moet dienen, zal de /PORT-OUTENABLE lijn op een logische 1 gebracht worden. Hierdoor zullen beide transistoren sperren.

4.2.1 Configuratie van een digitale I/O-poort

Iedere GPIO-poort beschikt over 2 registers, nl. PxMDOUT en Px.

Om een GPIO-poort, of een aantal pinnen van een GPIO-poort, te configureren als output of input gebruiken we het PxMDOUT-register.

Om data weg te schrijven of ingelezen data op te vragen wordt er gebruikgemaakt van het Px-register. Dit register wordt ook wel het dataregister genoemd.

Voorbeelden:

a) Poort 0 configureren als uitvoerpoort

```
…
mov SFRPAGE,#0FH        ; alle configuratieregisters van de digitale
                        crossbar bevinden zich in SFR-pagina F.
mov XBR2,#40H           ; de crossbar aanzetten.
mov P0MDOUT,#0FFH       ; Alle pinnen van poort 0= uitvoer
…
```

b) Poort 3 configureren als invoerpoort

```
…
mov SFRPAGE,#0FH        ; alle configuratieregisters van de digitale
                        crossbar bevinden zich in SFR-pagina F.
mov XBR2,#40H           ; de crossbar aanzetten.
mov P3MDOUT,#00H        ; Alle pinnen van poort 3 = invoer
mov P3,#0FFh            ; Om poort 3 te gebruiken als invoerpoort
                        moeten alle invoerpinnen na configuratie op
                        een logische 1 gezet worden.
…
```

c) Pin 4 van poort 5 configureren als uitvoerpin

```
…
mov SFRPAGE,#0FH        ; alle configuratieregisters van de digitale
                        crossbar bevinden zich in SFR-pagina F.
mov XBR2,#40H           ; de crossbar aanzetten.
mov P5MDOUT,#10H        ; Pin P5.4 = uitvoer
…
```

Opmerkingen:

- Wanneer een poortpin als invoerpin wordt geconfigureerd moet er een logische 1 naar worden weggeschreven.
- Bij het opstarten van de controller of na het uitvoeren van reset zijn alle poortpinnen geconfigureerd als invoerpin.
- Bij poorten 0,1,2 of 3 kunnen diverse pinnen een andere bestemming hebben gekregen (zie instellingen crossbar). Ook deze pinnen moet juist geconfigureerd worden, d.w.z. dat bijvoorbeeld de TX-pin (transmit) van UART0 als uitvoerpin moet geconfigureerd worden en de RX-pin (receive) als invoerpin.
- De configuratie- en dataregisters van poorten 0,1,2 en 3 bevinden zich in alle SFR-pagina's, terwijl de registers voor de overige poorten (P4, P5, P6 en P7) enkel in SFR-pagina F te bereiken zijn. De reden hiervoor is dat poorten 0,1,2 en 3 beheerd worden door de crossbar en dat dus niet op voorhand geweten is welke periferie gebruik zal maken van welke poortpinnen.

4.2.2 Een voorbeeldprogramma

Onderstaand programma zal 20 keer een bit inlezen van P3.4 en wegschrijven naar P1.7 .

```
        cseg  at 0000H                ;Schrijf onderstaande code in het programma-
                                      geheugen te beginnen bij adres 0000H
        jmp main                      ; onvoorwaardelijke sprong naar het
                                      hoofdprogramma
        cseg at 0050H                 ;Schrijf onderstaande code in het
                                      programmageheugen te beginnen bij adres
                                      0050H
main:   clr EA
        mov WDTCN ,#0DEH              ;uitschakelen van de watchdog timer
        mov WDTCN,#0ADH
        setb EA
        mov SFRPAGE,#0FH
        mov XBR2,#40H                 ; crossbar aanzetten
        mov P3MDOUT,#00H              ; P3.4 => invoer. De overige pinnen zijn don't
                                      cares
        mov P3,#0FFH
        mov P1MDOUT, #0FFH            ; P1.7 =>uitvoer. De overige pinnen zijn don't
                                      cares
        mov R0,#20d                   ; R0 gebruiken als telregister
Loop:   mov C, P3.4                   ; P3.4 inlezen en wegschrijven naar de carry
        mov P1.7,C                    ; De carry wegschrijven naar P1.7
        djnz R0,loop                  ; Loop 20 keer doorlopen
        jmp $                         ; programma stoppen met een oneindige lus
        END
```

Opmerkingen:

- Het cseg directive geeft aan waar de assembleerder de overeenstemmende machinecode in het programmageheugen moet wegschrijven. Bij het opstarten van de controller of na het uitvoeren van een reset bevat de programmateller de waarde 0000H.
- De eerste instructie is een onvoorwaardelijke spronginstructie om ervoor te zorgen dat de interruptvectoren, die zich vooraan in het programmageheugen bevinden, niet worden overschreven.
- In het hoofdprogramma wordt de watchdog[18] timer uitgeschakeld. Dit gebeurt door achtereenvolgens de waarden DE en AD (of samen DEAD) naar het configuratieregister van de watchdog timer te schrijven. Dit proces mag niet worden onderbroken door interrupts. Vooraf worden interrupts afgezet en na het uitzetten van de watchdog timer terug aangezet.
- Ieder programma eindigt uiteindelijk met een oneindige lus. Indien dit niet gebeurt zal de programmateller continu worden geïncrementeerd tot wanneer de controller op het aangeduide adres een bitpatroon vindt dat niet kan worden gedecodeerd. Op dat moment zal de controller vastlopen en de verbinding met de IDE verbreken. Op zijn beurt leidt dit tot het crashen van de IDE waardoor de gebruiker het programma opnieuw zal moeten starten!
- Het END directive geeft aan dat de assembleerder mag stoppen met het omzetten van assembleertaal naar machinecode. Het geeft niet aan dat daar het programma stopt!

4.3 Timer-counters

De C8051F120 beschikt over 5 timer/counters. De werking van timers 0 en 1 is volledig analoog aan de werking van de timer/counters van de oorspronkelijke 8051. Deze paragraaf behandelt uitsluitend de werking van de oorspronkelijke timer/counters.

4.3.1 Verschil tussen een timer en een counter

Wanneer een timer/counter ingesteld is als timer worden de telregisters van de timer/counter bij iedere machinecyclus met 1 verhoogd. De duur van een machinecyclus is instelbaar via het CKCON register. Een machinecyclus kan hierdoor bestaan uit 4, 12 of 48 klokcycli.

Bij een counter worden er pulsen geteld die afkomstig zijn van een externe bron. Bij elke 1 naar 0 overgang zullen de telregisters worden verhoogd met 1.

[18] Een watchdog timer is een timer die bij overlopen een reset uitvoert. Dit is een handige eigenschap om ongewenste oneindige lussen te onderbreken. Het overlopen van de watchdog timer kan vermeden worden door hem op discrete tijdstippen te resetten waardoor hij dus nooit overloopt.

4.3.2 Het timer/counter configuratieregister TMOD

M.b.v. het register TMOD kunnen timers 0 en 1 geconfigureerd worden. De meest significante nibble bevat de instellingen van timer 1, de minst significante nibble de configuratiegegevens van timer 0.

Timer 1				Timer 0			
GATE1	C/T1	T1M1	T1M0	GATE0	C/T0	T0M1	T0M0

GATE1 Dit bit zetten betekent dat de timer/counter enkel kan gestart worden als /INT1=1. Externe interruptlijn 1 kan er dus voor zorgen dat de timer/counter vergrendeld wordt.

C/T1 C/T1=0 betekent dat de timer/counter zal fungeren als timer
C/T1=1 betekent dat de timer/counter zal fungeren als counter

T1M1 Beide bits leggen voor het desbetreffende telregister de werkwijze vast:
T1M0

M1	M0	Mode
0	0	13 bit timer/counter
0	1	16 bit timer/counter
1	0	8 bit timer/counter met autoreload
1	1	Timer 1 is niet actief

GATE0 Identiek aan GATE1 maar hier zal externe interruptlijn 0 timer/counter 0 vergrendelen.

C/T0 C/T1=0 betekent dat de timer/counter zal fungeren als timer
C/T1=1 betekent dat de timer/counter zal fungeren als counter

T0M1 Beide bits leggen voor het desbetreffende telregister de werkwijze vast:
T0M0

M1	M0	Mode
0	0	13 bit timer/counter
0	1	16 bit timer/counter
1	0	8 bit timer/counter met autoreload
1	1	Timer 0 configureren als 2 8 bit timer/counters

4.3.3 Het timer/counter controleregister TCON

In dit punt gaan we ons enkel beperken tot het bespreken van de 4 meest significante bits.

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1 Timer Flag 1. Bij een overflow van de timer/counter 1 van FFFFH (16 bit T/C) of FFH (8 bit T/C) naar 00H wordt dit bit op 1 gezet.
Indien er geen gebruik gemaakt wordt van interrupts kan er softwarematig op dit vlaggetje getest worden. Na het overlopen van de timer/counter moet dit vlaggetje dan wel softwarematig worden gewist.

Bij het gebruik van interrupts wordt dit vlaggetje automatisch gewist.

TR1 Het zetten van dit bit zorgt ervoor dat het tellen, in de telregister TH1 en TL1, begint. Het bit op 0 zetten stopt het tellen.

TF0 Zelfde functie als TF1.
TR0 Zelfde functie als TR1.

4.3.4 Het klokcontroleregister (CKCON)

Bij het instellen van een tijdsinterval komt het vaak voor dat dit interval meer klokpulsen vergt dan dat een 16 bit timer/counter aankan. Een mogelijke oplossing bestaat erin om te proberen de klok te delen.

-	-	-	T1M	T0M	-	SCA1	SCA0

T1M Via dit bit wordt de klokbron ingesteld. Er kan gekozen worden om de systeemklok te gebruiken of de systeemklok gedeeld door een waarde bepaald door de settings van SCA1 en SCA0.

T0M Zelfde functie als T1M

SCA1
SCA0

SCA1	SCA0	Prescaled clock
0	0	Systeemklok gedeeld door 12
0	1	Systeemklok gedeeld door 4
1	0	Systeemklok gedeeld door 48
1	1	Externe klok gedeeld door 8

Instellingen in dit register hebben enkel impact op de snelheid waarmee er geteld wordt. Deze instellingen veranderen niets aan de frequentie van de systeemklok!

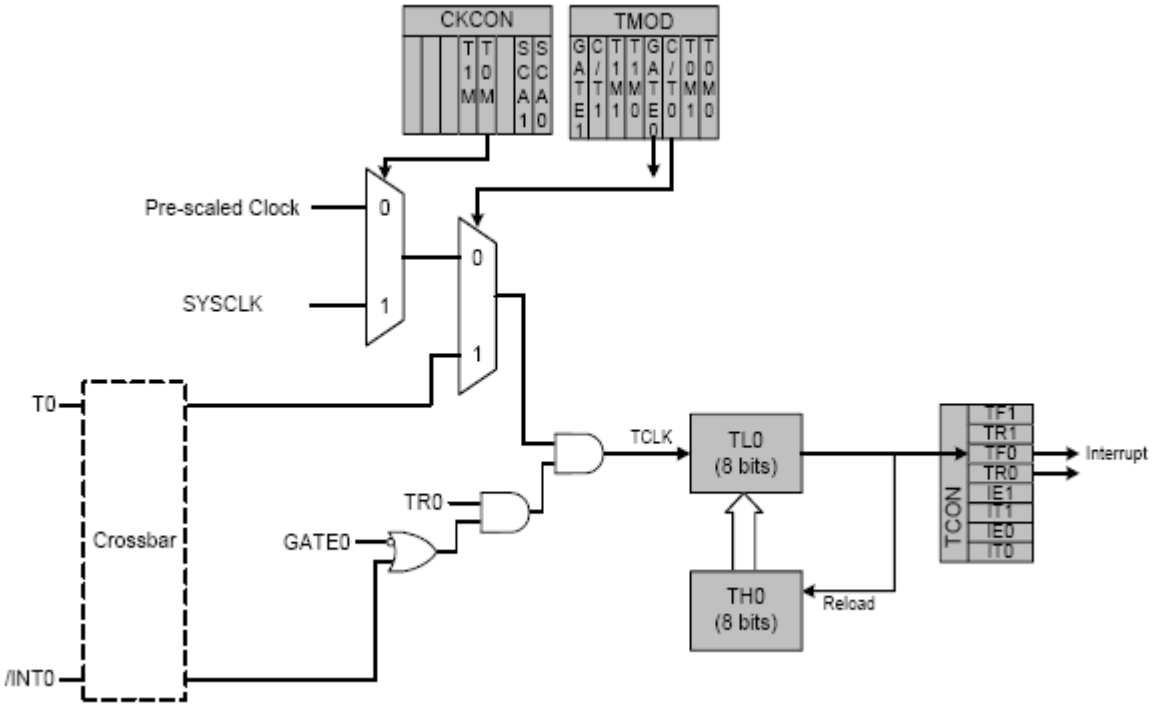
De systeemklok is gelijk aan 24,5 MHz gedeeld door 8!

4.3.5 Bespreking van timermodes 1 en 2

4.3.5.1 Mode 1(16 bit timer/counter)

Bij het starten van een timer/counter in mode 1 worden er pulsen geteld in de 2 telregisters THx en TLx. Het register THx bevat de meest significante byte van de tellerwaarde, TLx de minst significante byte.

4.3.5.2 Mode 2(8 bit timer/counter met autoreload)



Figuur 14: 8 bit timer/counter met autoreload

Wanneer het tijdsinterval voldoende klein is, kan een 8 bit timer volstaan. Bovendien kan er dan gebruik gemaakt worden van de autoreload functionaliteit.

Bij het overlopen van de timer/counter wordt de autoreload-waarde, die zich in THx bevindt, naar TLx gekopieerd.

4.3.6 Voorbeeld

Configureer timer 0 om op pin P2.1 een blokgolf te genereren met een periode van 120 ms (omschakelen om de 60 ms).

- Bepalen van het aantal te tellen klokpulsen

  Het aantal klokpulsen volgt uit volgende formule:

  $$\#klokpulsen = klokfrequentie * tijdsinterval$$

  Indien we kiezen voor de systeemklok hebben we $\frac{24{,}5*10^6}{8} * 60 * 10^{-3} = 183750$ klokpulsen nodig. Een 16 bit timer kan slechts 65536 pulsen tellen.

  Wanneer we de systeemklok nogmaals delen door 48 hebben we nog nood aan 3828,125 of afgerond 3828 klokpulsen. Dit is te realiseren met een 16 bit timer!

- Bepalen van de telregisterwaarden

  Aangezien een 16 bit timer overloopt bij een overgang van FFFFH naar 0000H moet er van de waarde FFFFH de waarde 3828d afgetrokken worden. De timer zal dan na exact 3828 klokpulsen overlopen.

  $FFFFH - EF4H = F10BH$

  TH0=F1H en TL0=0BH

- Het volledig programma

```
        cseg  at 0000H                  ;Schrijf onderstaande code in het programma-
                                        geheugen te beginnen bij adres 0000H
        jmp main                        ; onvoorwaardelijke sprong naar het
                                        hoofdprogramma
        cseg at 0050H                   ;Schrijf onderstaande code in het
                                        programmageheugen te beginnen bij adres
                                        0050H
main:   clr EA
        mov WDTCN ,#0DEH                ;uitschakelen van de watchdog timer
        mov WDTCN,#0ADH
        setb EA
        mov SFRPAGE,#0FH
        mov XBR2,#40H                   ; crossbar aanzetten
        mov P2MDOUT,#0FFH               ; P2.1 =>uitvoer. De overige pinnen zijn don't
                                        cares
        mov SFRPAGE#00H                 ; alle timerregisters bevinden zich in SFR-
                                        pagina 0
        mov TMOD,#01H                   ; Timer 0 mode 1 (16 bit timer)
        mov CKCON,#02H                  ; SYSCLK nog eens extra delen door 48
        mov TH0,#0F1H                   ; TH0=F1H
        mov TL0,#0BH                    ; TL0=0BH
        setb TR0                        ; timer 0 starten
Loop:   jnb TF0, $                      ; wacht 60 ms
        clr TF0                         ; Timer overflow vlag wissen
        cpl P2.1                        ;polariteit P2.1 omwisselen
        clr TR0                         ; timer 0 stoppen
        mov TH0,#0F1H                   ; telregisters opnieuw instellen
        mov TL0,#0BH
        setb TR0                        ; timer 0 opnieuw starten
        jmp loop                        ; oneindige lus
        END
```

4.4 Interrupts

Interrupt Source	Interrupt Vector	Priority Order	Pending Flags	Bit addressable?	Cleared by HW?	Enable Flag	Priority Control
Reset	0x0000	Top	None	N/A	N/A	Always Enabled	Always Highest
External Interrupt 0 (/INT0)	0x0003	0	IE0 (TCON.1)	Y	Y	EX0 (IE.0)	PX0 (IP.0)
Timer 0 Overflow	0x000B	1	TF0 (TCON.5)	Y	Y	ET0 (IE.1)	PT0 (IP.1)
External Interrupt 1 (/INT1)	0x0013	2	IE1 (TCON.3)	Y	Y	EX1 (IE.2)	PX1 (IP.2)
Timer 1 Overflow	0x001B	3	TF1 (TCON.7)	Y	Y	ET1 (IE.3)	PT1 (IP.3)
UART0	0x0023	4	RI0 (SCON0.0) TI0 (SCON0.1)	Y		ES0 (IE.4)	PS0 (IP.4)
Timer 2	0x002B	5	TF2 (TMR2CN.7) EXF2 (TMR2CN.6)	Y		ET2 (IE.5)	PT2 (IP.5)
Serial Peripheral Interface	0x0033	6	SPIF (SPI0CN.7) WCOL (SPI0CN.6) MODF (SPI0CN.5) RXOVRN (SPI0CN.4)	Y		ESPI0 (EIE1.0)	PSPI0 (EIP1.0)
SMBus Interface	0x003B	7	SI (SMB0CN.3)	Y		ESMB0 (EIE1.1)	PSMB0 (EIP1.1)
ADC0 Window Comparator	0x0043	8	AD0WINT (ADC0CN.1)	Y		EWADC0 (EIE1.2)	PWADC0 (EIP1.2)
PCA 0	0x004B	9	CF (PCA0CN.7) CCFn (PCA0CN.n)	Y		EPCA0 (EIE1.3)	PPCA0 (EIP1.3)
Comparator 0 Falling Edge	0x0053	10	CP0FIF (CPT0CN.4)	Y		ECP0F (EIE1.4)	PCP0F (EIP1.4)
Comparator 0 Rising Edge	0x005B	11	CP0RIF (CPT0CN.5)	Y		ECP0R (EIE1.5)	PCP0R (EIP1.5)
Comparator 1 Falling Edge	0x0063	12	CP1FIF (CPT1CN.4)	Y		ECP1F (EIE1.6)	PCP1F (EIP1.6)
Comparator 1 Rising Edge	0x006B	13	CP1RIF (CPT1CN.5)	Y		ECP1R (EIE1.7)	PCP1F (EIP1.7)
Timer 3	0x0073	14	TF3 (TMR3CN.7) EXF3 (TMR3CN.6)	Y		ET3 (EIE2.0)	PT3 (EIP2.0)
ADC0 End of Conversion	0x007B	15	AD0INT (ADC0CN.5)	Y		EADC0 (EIE2.1)	PADC0 (EIP2.1)
Timer 4	0x0083	16	TF4 (TMR4CN.7) EXF4 (TMR4CN.7)	Y		ET4 (EIE2.2)	PT4 (EIP2.2)
ADC2 Window Comparator	0x008B	17	AD2WINT (ADC2CN.0)	Y		EWADC2 (EIE2.3)	PWADC2 (EIP2.3)
ADC2 End of Conversion	0x0093	18	AD2INT (ADC2CN.5)	Y		EADC2 (EIE2.4)	PADC2 (EIP2.4)
RESERVED	0x009B	19	N/A	N/A	N/A	N/A	N/A
UART1	0x00A3	20	RI1 (SCON1.0) TI1 (SCON1.1)	Y		ES1 (EIE2.6)	PS1 (EIP2.6)

Figuur 15: Interrupts

Volgens figuur 15 beschikt de C8051F120 over 20 interruptbronnen.

Bij het optreden van een interrupt wordt het hoofdprogramma onderbroken en wordt de code die bij die specifieke interrupt hoort uitgevoerd.

Om een interrupt af te handelen moet een ISR[19] voorhanden zijn. Het beginadres van deze speciale subroutine ligt vast. Zo zal de ISR van UART0 moeten beginnen op adres 0023H (zie figuur 15).

Aangezien het verschil tussen 2 opeenvolgende startadressen van ISR's slechts een aantal bytes bedraagt is deze hoeveelheid geheugen niet voldoende om de volledige interrupt af te handelen. Meestal vind je op die adressen slechts één instructie, nl. een onvoorwaardelijke sprongінstructie naar een locatie die zich wat verderop in het programmageheugen bevindt.

[19] Interrupt Service Routine

4.4.1 Het IE- register

Het IE-register is één van de registers die moet aangesproken worden om een bepaalde interrupt toe te laten.

Het IE-register heeft volgende layout:

EA	IEGF0	ET2	ES0	ET1	EX1	ET0	EX0

EA	Enable all interrupts. Het op nul zetten van dit bit betekent dat geen enkele interrupt de programmacode kan onderbreken. Het zetten van dit bit laat interrupts toe. Welke interrupts toegelaten worden is afhankelijk van de individuele settings van iedere interruptbron (te vinden in IE, EIE1 en EIE2 registers).
ET2	Timer 2 interrupt enable
ES0	UART0 interrupt enable
ET1	Timer 1 interrupt enable
EX1	External interrupt 1 enable
ET0	Timer 0 interrupt enable
EX0	External interrupt 0 enable

4.4.2 Het IP-register

Met de registers IP, EIP1 en IEP2 is het mogelijk om bepaalde interrupts een hogere prioriteit te geven dan de andere. Individuele prioriteiten toewijzen is onmogelijk. De enige in te stellen prioriteiten zijn “Hoog” en “Laag”. Indien niets naar deze registers geschreven wordt, hebben alle interrupts een lage prioriteit.

4.4.3 Voorbeeld

Als voorbeeld nemen we het voorbeeld 4.3.6 maar met het gebruik van interrupts.

```
cseg  at 0000H          ;Schrijf onderstaande code in het programma-
                        geheugen te beginnen bij adres 0000H
jmp main                ; onvoorwaardelijke sprong naar het
                        hoofdprogramma
cseg at 000BH           ; interrupt vector voor timer 0

jmp ISRTR0              ; onvoorwaardelijke sprong naar het label
                        ISRTR0

cseg at 0050H           ;Schrijf onderstaande code in het
                        programmageheugen te beginnen bij adres
                        0050H
```

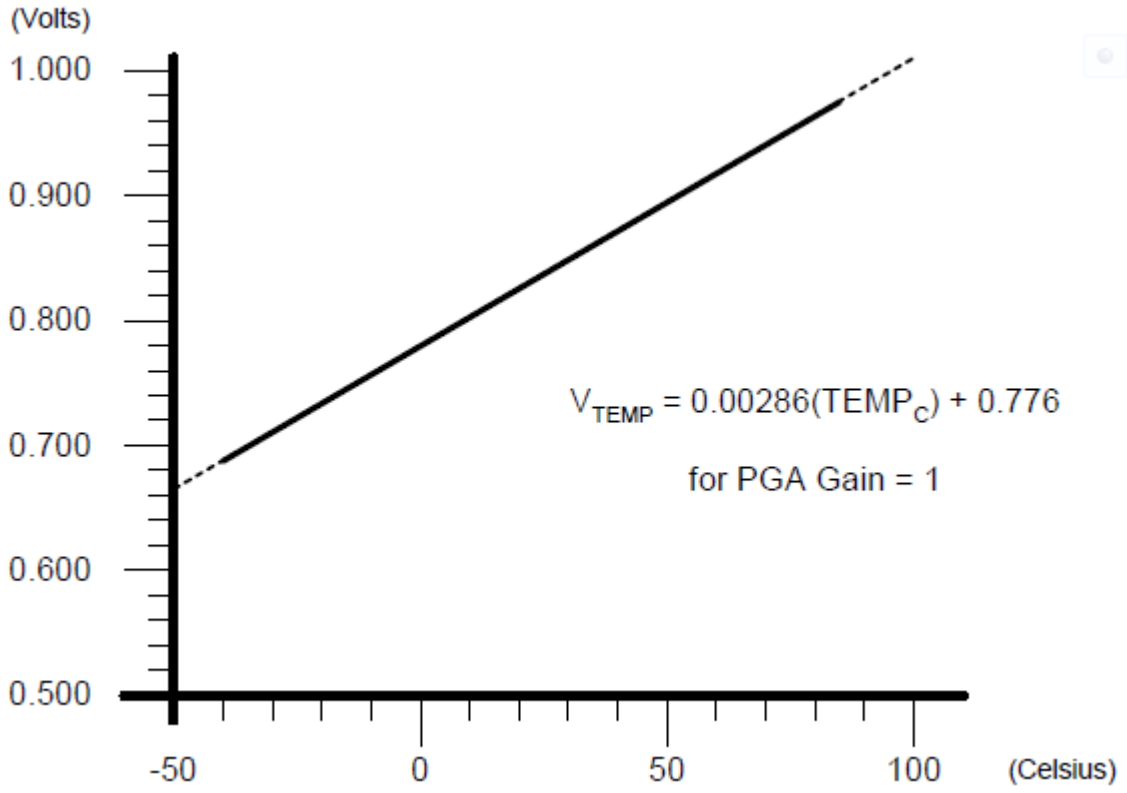
```
main:   clr EA
        mov WDTCN ,#0DEH        ;uitschakelen van de watchdog timer
        mov WDTCN,#0ADH
        setb EA
        setb ET0                ;interrupts van timer 0 toelaten
        mov SFRPAGE,#0FH
        mov XBR2,#40H           ; crossbar aanzetten
        mov P2MDOUT,#0FFH       ; P2.1 =>uitvoer. De overige pinnen zijn don't
                                cares
        mov SFRPAGE#00H         ; alle timerregisters bevinden zich in SFR-
                                pagina 0
        mov TMOD,#01H           ; Timer 0 mode 1 (16 bit timer)
        mov CKCON,#02H          ; SYSCLK nog eens extra delen door 48
        mov TH0,#0F1H           ; TH0=F1H
        mov TL0,#0BH            ; TL0=0BH
        setb TR0                ; timer 0 starten
        jmp $                   ; oneindige lus en wachten op interrupts

ISRTR0: cpl P2.1                ; polariteit P2.1 omwisselen
        clr TR0                 ; timer 0 stoppen
        mov TH0,#0F1H           ; telregisters opnieuw instellen
        mov TL0,#0BH
        setb TR0                ; timer 0 opnieuw starten
        RETI                    ; einde ISR
        END
```

5 Analoge I/O-mogelijkheden van de C8051F120 microcontroller

5.1 De onboard temperatuurssensor

De C8051F120 beschikt net zoals de meeste processoren over een ingebouwde temperatuursweerstand om oververhitting van de kern te voorkomen. Bij een dergelijke sensor verandert de weerstand in functie van de temperatuur. Als de weerstandswaarde daalt bij stijgende temperatuur spreekt men van een NTC[20]-weerstand, in het andere geval van een PTC[21]-weerstand.

Om vlot temperaturen te kunnen omzetten naar analoge spanningen kan je gebruikmaken van de onderstaande grafiek die je ook kan terugvinden op pagina 50 van de datasheet.



Figuur 16: Omzettingsgrafiek van de onboard temperatuurssensor

Zoals uit de grafiek opgemaakt kan worden, stijgt de spanning naarmate de chiptemperatuur toeneemt. Gelet op de wet van Ohm, $V = R * I$, is de spanning rechtevenredig met de weerstandswaarde wat er op duidt dat de onboad temperatuurssensor een lineaire PTC-weerstand is. De temperatuurssensor is lineair omdat, onafhankelijk van de werktemperatuur, iedere temperatuursverandering ΔT een gelijke spanningswijziging ΔV teweegbrengt. Wiskundig uitgedrukt is de afgeleide van de spanning naar de temperatuur gelijk aan een constante, hier

[20] Negatieve temperatuurscoëfficiënt
[21] Positieve temperatuurscoëfficiënt

0.00286. Grafisch is dit onmiddellijk zichtbaar aangezien het verband tussen beide grootheden gegeven wordt door een rechte.

Het moet opgemerkt worden dat er geen enkele sensor volkomen lineair is. Het verband tussen de spanning en de temperatuur is een willekeurige functie. Afhankelijk van sensor tot sensor zal de grafiek in een relatief klein interval quasi lineair zijn. Het is dan ook de bedoeling om er voor te zorgen dat het quasi lineaire gebied overeenstemt met het werkgebied. Het gebied waar de sensor als lineair kan worden beschouwd wordt dan ook meestal in de documentatie vermeld.

Het belang van een lineaire karakteristiek is onmiddellijk voelbaar bij het verwerken van gegevens. Het temperatuursverschil bij een lineaire sensor kan worden berekend door eenvoudige bewerkingen zoals optellen, aftrekken, delen en vermenigvuldigen. Bij een kwadratisch verband, of nog erger bij een logaritmisch verband, is dit met een beperkte instructieset al niet meer zo eenvoudig.

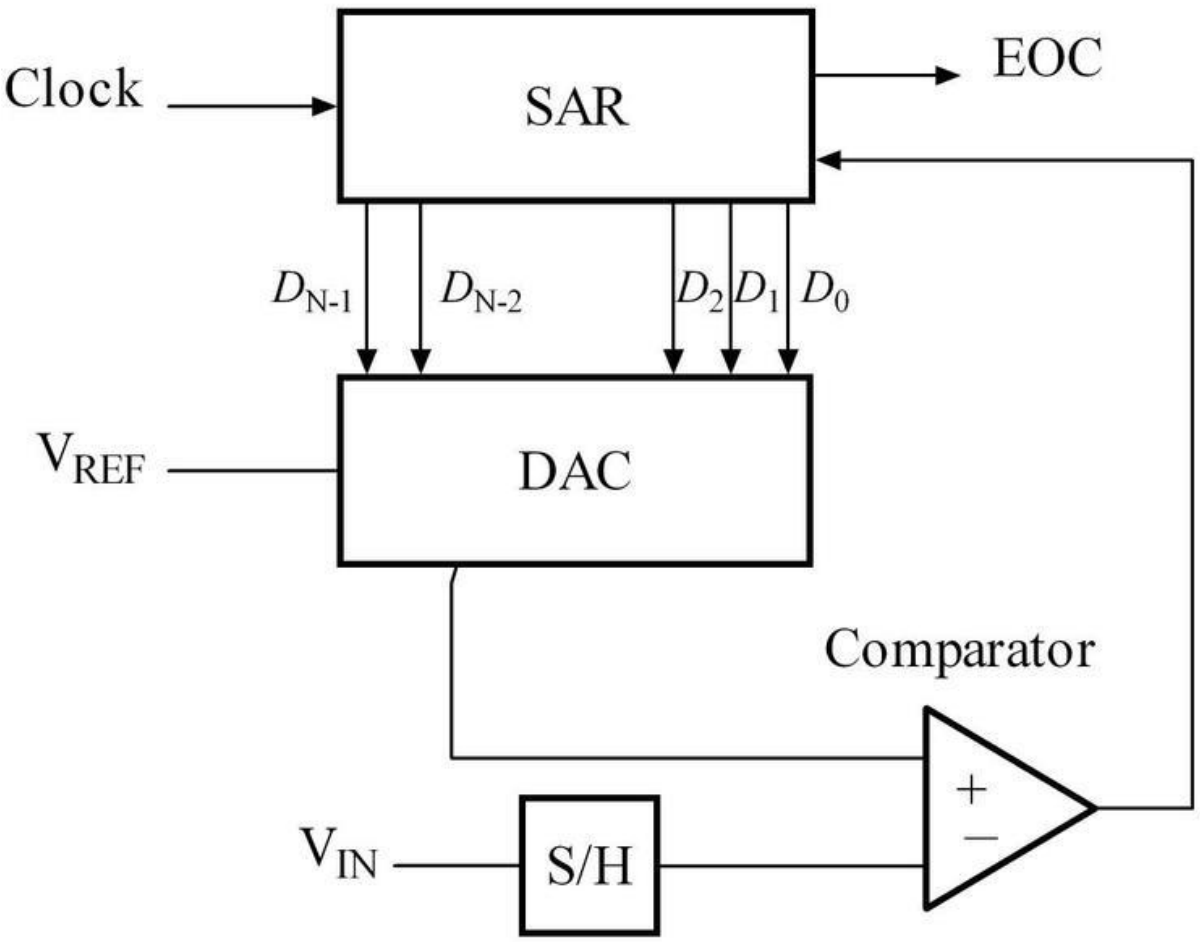
5.2 Analoog naar digitaal conversie

Aangezien een analoge sensor zorgt voor het omzetten van temperatuur, druk, lichtsterkte, … naar een spanning, moet deze spanning worden omgezet naar een getal. Een microcontroller kan immers enkel met getallen werken. Het omzetten van een spanning naar een getal, of beter digitale waarde, gebeurt door een analoog naar digitaal omzetter of afgekort ADC[22]. Een DAC[23] doet het omgekeerde, ttz. een digitale waarde omzetten naar een analoge spanning.

Hedendaagse microcontrollers hebben intern één of meerdere ADC's en DAC's. Zo beschikt de C8051F120 over een 12-bit SAR[24] ADC (ADC0, zie pagina 49 en volgende) en over een 8-bit SAR ADC (ADC2, vanaf pagina 85).

[22] Analog to Digital converter
[23] Digital to Analog Converter
[24] Successive Approximation Register

5.2.1 Werking van een SAR ADC



Figuur 17: Blokschema van een SAR ADC

Zoals uit figuur 17 blijkt, bevat een SAR ADC de volgende onderdelen:

- Een **Sample & Hold** (S/H) schakeling, die tijdens de conversie de analoge ingangsspanning V_{in} constant houdt.
- Een intern register, **S**uccessive **A**pproximation **R**egister genoemd, dat instaat voor de opslag van de digitale waarde. Het einde van de conversie wordt aangegeven door de **EOC**-vlag (end of conversion). Men kan deze vlag gebruiken om te pollen of om de microcontroller te onderbreken via een interrupt.
- Een **DAC** die de waarde van de ADC, opgeslagen in het SAR-register, omzet naar een analoge spanning. Voor het omzetten van een getal naar een spanning wordt een refentiespanning $Vref$ gebruikt. De uitgangsspanning van de DAC is dus afhankelijk van het getal, opgeslagen in het SAR, en de referentiespanning.
- Een **comparator** die de ingangspanning vergelijkt met de uitgangsspanning van de DAC. Afhankelijk of de ingangsspanning groter of kleiner is, zal de inhoud van het SAR-register worden aangepast.

Zoals de naam van de ADC gedeeltelijk verklapt, wordt de analoge spanning omgezet naar een digitale waarde via opeenvolgende benaderingen (cfr. **S**uccessive **A**pproximation **R**egister). De omzetting verloopt als volgt:

1. Het meest significante bit van het SAR-register, D_{N-1}, wordt op 1 gezet, terwijl de resterende N-1 bits 0 zijn.
2. De DAC zet de waarde van het SAR-register om in een analoge spanning. Hierbij worden alle 1-bits uit het SAR omgezet naar een spanning die afhankelijk is van de positie van het desbetreffende 1-bit. Zo telt het meest significante bit mee voor $\frac{Vref}{2}$, het tweede meest significante bit voor $\frac{Vref}{4}$, … ,en het minst significante bit voor $\frac{Vref}{2^N}$.

   Kortom het bit op positie i telt mee voor $\frac{Vref}{2^{N-i}}$ waarbij i gelegen is in het gesloten interval $[0, N-1]$.

   De uitgangsspanning van de DAC wordt bijgevolg gegeven door $\sum_{i=0}^{N-1} D_i * \frac{Vref}{2^{N-i}}$.
3. De comparator vergelijkt de ingangsspanning met de uitgangsspanning van de DAC. Indien blijkt dat de ingangsspanning kleiner is dan de uitgangsspanning van de DAC, wordt het gezette bit gewist. In het andere geval wordt het bit in het SAR-register ongemoeid gelaten.

   In het geval van een 12-bit ADC bevat het SAR-register nu 800H(V_{in} groter dan V_{DAC}) of 000H(V_{in} kleiner dan V_{DAC}).

4. Stappen 1 t.e.m. 3 worden hernomen maar nu door het eerstvolgende bit op 1 te zetten. Dit is na de eerste iteratie D_{N-2} en bij de laatste iteratie D_0. In totaal worden stappen 1-3 N keer doorlopen

5.2.2 Voorbeeld 1

We beschikken over een 12-bit SAR ADC en na conversie bevat het SAR-register de waarde A7FH (1010 0111 1111). Met welke analoge spanning komt dit ongeveer overeen?

Om de omzetting te doen, moeten we enkel rekening houden met de 1-bits en de bijhorende gewichtscoëfficiënten.

$$V = \frac{Vref}{2} + 0 + \frac{Vref}{8} + 0 + 0 + \frac{Vref}{64} + \frac{Vref}{128} + \frac{Vref}{256} + \frac{Vref}{512} + \frac{Vref}{1024} + \frac{Vref}{2048} + \frac{Vref}{4096}$$

Na het gelijknamig maken van alle breuken geeft dit:

$$V = \frac{Vref}{4096}(2048 + 512 + 64 + 32 + 16 + 8 + 4 + 2 + 1)$$
$$= \frac{Vref}{4096} * 2687$$
$$= \frac{Vref}{4096} * (A7F)_{10}$$

De spanning waarmee een bepaald getal overeenstemt kan dus rechtstreeks worden berekend door de decimale voorstelling van het SAR-register te vermenigvuldigen met de referentiespanning en te delen 2^{12} !

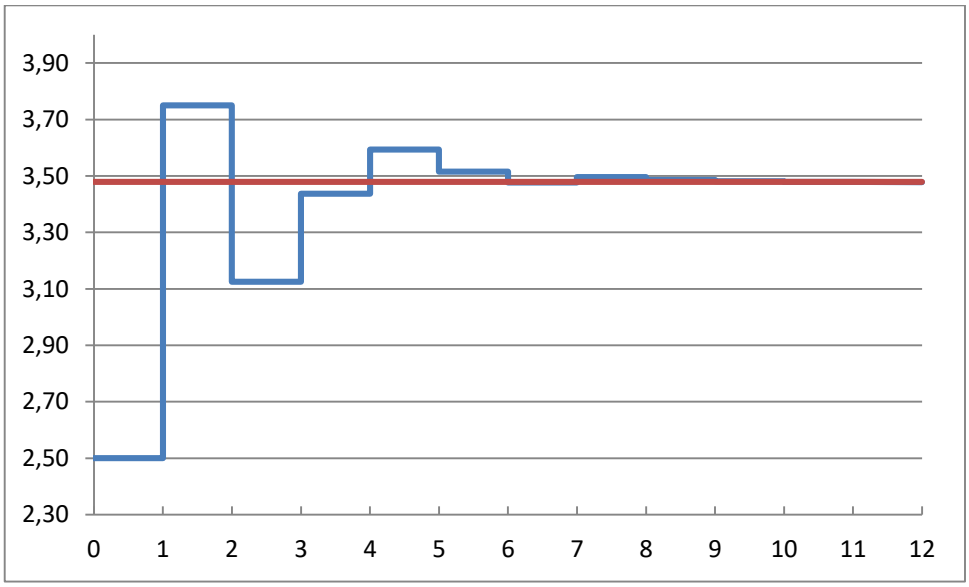
5.2.3 Voorbeeld 2

We beschikken opnieuw over een 12-bit SAR ADC en over een referentiespanning van 5V. Wat is na conversie de inhoud van het SAR-register indien de ingangsspanning 3.4789V bedraagt?

Iteratie	Inhoud SAR	Benaderende waarde (V_{DAC}) in Volts	V_{in} groter of kleiner dan benaderende waarde
1	**1**000 0000 0000	2.5	Groter
2	1**1**00 0000 0000	3.75	Kleiner
3	1**01**0 0000 0000	3.125	Groter
4	101**1** 0000 0000	3.4375	Groter
5	1011 **1**000 0000	3.59375	Kleiner
6	1011 **01**00 0000	3.515625	Kleiner
7	1011 0**01**0 0000	3.4765625	Groter
8	1011 001**1** 0000	3.49609375	Kleiner
9	1011 001**0 1**000	3.486328125	Kleiner
10	1011 0010 **01**00	3.4814453125	Kleiner
11	1011 0010 0**01**0	3.47900390625	Kleiner
12	1011 0010 00**01**	3.477783203125	Groter => EOC

Bij gebruik van een referentiespanning van 5V en een 12-bit SAR stemt 3.4789V overeen met 1011 0010 0001 of B21H.

Bemerk dat hoe meer bits de SAR ADC telt, hoe nauwkeuriger de benadering zal zijn. Echter hoe meer bits, hoe langer een conversie duurt. Het is dus belangrijk een goed compromis te vinden tussen enerzijds precisie en anderzijds conversietijd!



Figuur 18: Grafisch verloop van een SAR ADC-omzetting

5.3 Analoog naar digitaal conversie met de C8051F120

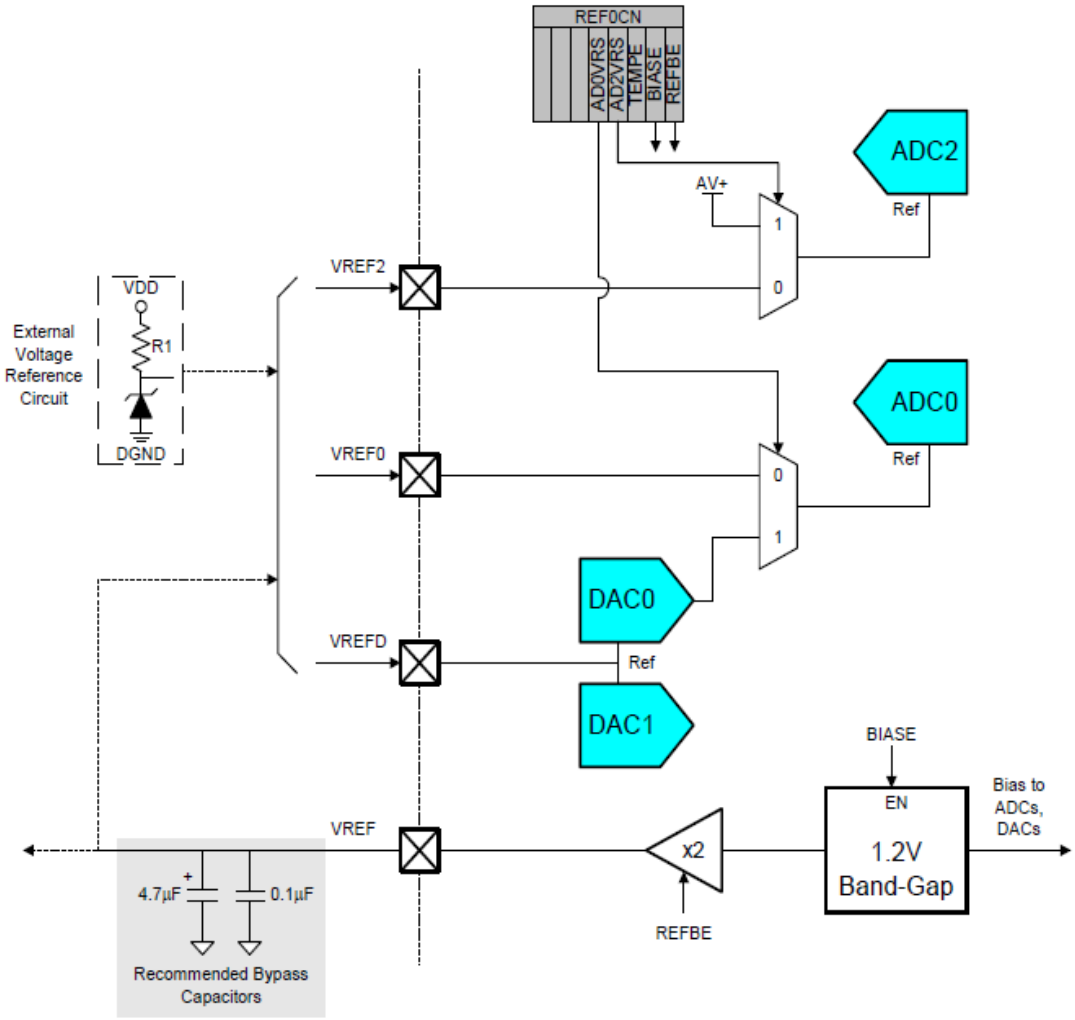
Zoals reeds eerder vermeld beschikt de C8051F120 over een 12-bit SAR ADC, **ADC0**, en een 8-bit SAR ADC, **ADC2**. De onderstaande paragrafen geven aan hoe de C8051F120 geconfigureerd moet worden om naast digitale invoer ook analoge invoer te kunnen verwerken.

5.3.1 Het instellen van de referentiespanning

Zoals uiteengezet in paragraaf 5.2 vereist iedere SAR ADC een referentiespanning. Voor de ADC's van de C8051F120 zijn er in principe twee mogelijkheden. Of je voorziet zelf een referentiespanning, of je maakt gebruik van de interne referentiespanning (cfr. pagina 107 en 108).

Uit figuur 19 blijkt dat een extern referentiecircuit, hier schematisch weergegeven door een zenerdiode en bijhorende voorschakelweerstand, kan aangesloten worden via de aansluitpinnen **VREF2**, **VREF0** en **VREFD**. Bemerk dat bij het gebruik van verschillende externe referentiecircuits de referentiespanningen voor ADC0 en ADC2 kunnen verschillen!

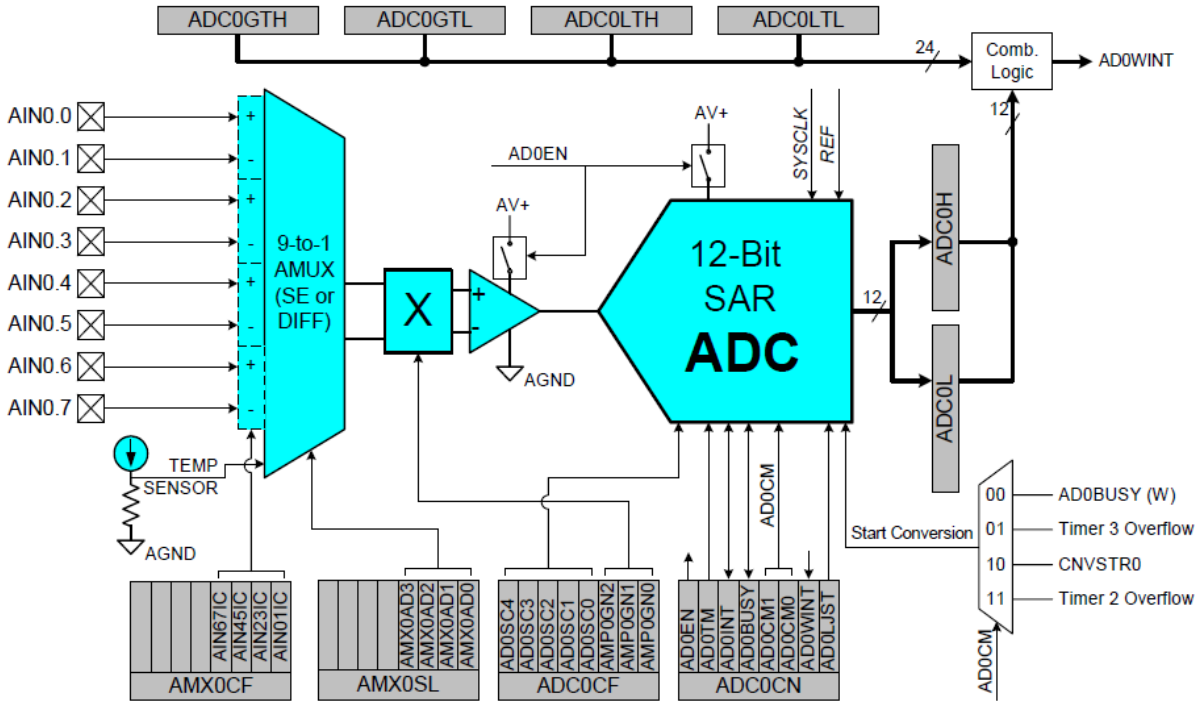
Wanneer de interne referentiespanning gebruikt wordt, wordt deze naar buiten gebracht via de **VREF**-pin die intern doorverbonden is met de pinnen VREF2, VREF0 en VREFD. Bij het gebruik van de interne referentiespanning hebben beide ADC's dezelfde referentiespanning! Interne en externe referentiecircuits kunnen dus **niet** gemengd worden!



Figuur 19: Referentiespanning

Het interne referentiecircuit bestaat uit een 1.2V spanningsbron die bij het inschakelen automatisch verdubbeld wordt tot ongeveer 2.43V. Na het inschakelen van het interne referentiecircuit wordt de spanning over twee bypass capaciteiten naar de VREF-pin gestuurd. Indien men wil beschikken over een constante gelijkspanning, waar alle eventuele ruis en rimpel worden weggefilterd, is het gebruik van bypass capaciteiten aangewezen. Het constant blijven van de referentiespanning is een must omdat de analoog naar digitaal omzetting steunt op een constante referentiespanning (cfr. paragrafen 5.2.2 en 5.2.3). Door het gebruik van die bypass capaciteiten duurt het ongeveer 2ms (cfr. tabel 9.1 pagina 108) vooraleer beide condensatoren volledig opgeladen zijn. Vooraleer er dus een analoog naar digitaal omzetting kan gestart worden, moet er na het inschakelen van de interne referentiespanning eenmalig 2ms gewacht worden. Dit kan bv . door onmiddellijk na het inschakelen van het referentiecircuit een dubbele wachtlus te implementeren.

5.3.2 Configuratie van ADC0



Figuur 20: Blokschema van ADC0

Volgens figuur 20 bestaat ADC0 uit volgende componenten:

- Een 9-naar-1 multiplexer die van de 9 analoge invoerpinnen (AIN0.0 tot AIN0.7 plus de interne temperatuurssensor) 1 invoerpin doorverbindt met de ADC. Welke invoerpin doorverbonden wordt, wordt bepaald door de SFR-registers **AMX0CF** en **AMX0SL**.
- Een vermenigvuldiger die eventueel het doorverbonden signaal kan versterken of verzwakken. Dit is bv. handig bij zeer kleine ingangsspanningen of bij spanningen die net iets groter zijn dan de referentiespanning. De maximale vermenigvuldigingsfactor bedraagt 16 en de minimale ½. De minstsignificante 3 bits van het **ADC0CF**-register worden gebruikt om de vermenigvuldigingsfactor in te stellen. Default staat de vermenigvuldigingsfactor ingesteld op 1.
- Een 12-bit SAR ADC waarvan de werking bepaald wordt door de registers **ADC0CF** en **ADC0CN**. Aangezien het **ADC0CF**-register enkel en alleen gebruikt wordt om de conversieklok in te stellen en omdat er default gebruikgemaakt wordt van de systeemklok SYSCLK, wordt dit register verder niet meer besproken.

  Na conversie kan je de meestsignificante 4 bits van de omzetting terugvinden in het register **ADC0H** en de minstsignificante 8 bits in het register **ADC0L**.

5.3.2.1 Bespreking van de verschillende werkingsmodi

Volgens de datasheet zijn er vier manieren om een conversie te starten. Deze zijn:

- Door een logische 1 weg te schrijven naar AD0BUSY, i.e. bit 4 van ADC0CN. Dit is handig wanneer je softwarematig de opdracht wil geven om een conversie te starten.
- Bij het overlopen van Timer 3. Dit is nuttig wanneer op welbepaalde momenten conversies moeten plaatsvinden.
- Bij detectie van een stijgende flank (0->1) op de externe pin CNVSTR0. In tegenstelling tot punt 1 wordt hier hardwarematig een conversie gestart.
- Gelijkaardig aan punt 2, maar nu bij het overlopen van Timer 2.

Om een conversie softwarematig te starten, moet de volgende procedure worden gevolgd:

- Wis de AD0INT-vlag (bit 5 van ADC0CN). AD0INT is de vlag die gezet wordt wanneer er een conversie werd voltooid.
- Start een nieuwe conversie door AD0BUSY op 1 te zetten.
- Wacht tot wanneer de AD0INT vlag wordt gezet.
- Verwerk de digitale waarde die is opgeslagen in de registers ADC0H en ADC0L.

5.3.2.2 Bespreking van de tracking modi

Naast het instellen van de werkingsmode, moet je bij het ADC0CN-register ook vermelden wat de trackingmode is. Er zijn twee mogelijkheden, nl. continue tracking of niet-continue tracking.

Bij continue tracking wordt het signaal continue gevolgd. Dit betekent dat ADC0 vrijwel onmiddellijk na het ontvangen van het startsignaal de gevraagde conversie kan starten. Bij niet-continue tracking zal ADC0 zich na iedere conversie in “low power” mode brengen. Dit betekent dat ADC0 na het ontvangen van het startsignaal **niet** onmiddellijk kan beginnen met de gevraagde conversie. Eerst moet ADC0 gedurende drie klokcycli het signaal volgen alvorens de conversie te kunnen aanvatten.

De meest eenvoudige, maar tevens de minst energievriendelijke manier van werken gaat uit van continue tracking.

5.3.2.3 Configuratie van de multiplexer

Via de 9-naar-1 kanaalkiezer wordt bepaald welk ingangssignaal doorverbonden wordt met de ADC. Omdat men de mogelijkheid heeft om zowel single ended signalen[25] als differentiële signalen[26] te verbinden met de analoge invoerpinnen van de C8051F120, heeft men twee registers nodig. Het ene register, **AMX0CF**, geeft aan of een analoge invoerpin verbonden is met een single ended invoerlijn, dan wel of ze deel uitmaakt van een differentieel invoerpaar. Het tweede register, **AMX0SL**, bepaalt dan welk signaal of signaalpaar verbonden wordt met de ADC.

[25] Bij single ended communicatie worden twee geleiders gebruikt. Eén geleider voor het overbrengen van het signaal en een tweede signaallijn die verbonden is met de analoge massa. Aangezien de controller intern over een massalijn beschikt, volstaat het om enkel de signaallijn met de analoge invoerpin te verbinden.
[26] Bij differentiële communicatie worden twee geleiders gebruikt voor het overbrengen van een signaal. De signaalinformatie wordt verstopt in het spanningsverschil tussen beide geleiders. Aangezien beide geleiders het signaal overbrengen, zijn er dus twee analoge invoerpinnen nodig.