Rijkswaterstaat Water, Verkeer en Leefomgeving

RWS C-regelaar
Handleiding

Rijkswaterstaat Water, Verkeer en Leefomgeving

RWS C-regelaar
Handleiding

Datum	25 maart 2014
Kenmerk	WVL016/Dht/
Eerste versie	7 december 2000

Documentatiepagina

Opdrachtgever(s)	Rijkswaterstaat Water, Verkeer en Leefomgeving
Titel rapport	RWS C-regelaar Handleiding
Kenmerk	WVL016/Dht/
Datum publicatie	25 maart 2014
Projectteam opdrachtgever(s)	de heer H. Heikoop
Projectteam Goudappel Coffeng	de heren ing. T.G. Dijkshoorn en ing. F.P.M. Peters
Projectomschrijving	
Trefwoorden	RWS C-regelaar, handleiding, verkeersregelprogramma's, basisspecificatie

Inhoudsopgave

Inhoud (vervolg) Pagina

Inhoud (vervolg) Pagina

Bijlagen

Literatuurlijst

1. Inleiding

1.1 Structuur van de handleiding

Ter inleiding worden in hoofdstuk 1 achtereenvolgens vijf zaken kort behandeld, namelijk:
- Wat de RWS C-regelaar is. (paragraaf 1.2)
- De test- en ontwikkelomgeving FLASH voor de RWS C-regelaar. (paragraaf 1.3)
- Het gebruik van de RWS C-regelaar (paragraaf 1.4)
- De internetsite van de RWS C-regelaar (paragraaf 1.5)

Snelle starters kunnen zich beperken tot het lezen van paragraaf 1.4.

Vervolgens worden in hoofdstuk 2 kort enkele belangrijke punten van de programmeertaal C in de RWS C-regelaar behandeld:
- de Microsoft Visual C++-compiler. (paragraaf 2.1)
- afspraken over extensies (paragraaf 2.2).

In hoofdstuk 3 wordt gedetailleerd beschreven hoe een applicatiespecificatie er als invoer voor de RWS C-regelaar uit moet zien. Een applicatiespecificatie in C bestaat hierbij uit drie files, te weten:
- CRAPDEF.H definitie van de kruispuntsafhankelijke gegevens zoals blokken signaalgroepen en dergelijke
- CRAPTAB.C declaratie van de tabellen
- CRAPCOD.C overige voorwaarden.

In hoofdstuk 4 worden een aantal bijzondere aspecten behandeld met betrekking tot het specificeren van de regeling zoals:
- opbouw en volgorde van de software en
- gedetailleerde bespreking van een aantal specificatietechnische zaken.

In hoofdstuk 5 worden de hulpmiddelen beschreven, waarmee binnen de RWS C-regelaar applicatiespecificaties getest kunnen worden. Het belangrijkste hulpmiddel is het programma FLASH (FLEXSYT Application Shell) [FLASH] waarmee applicatiespecificaties kunnen worden getest, gevisualiseerd en gesimuleerd.

In hoofdstuk 6 wordt de debugger uitgebreid beschreven. Uitgelegd wordt hoe de gebruiker de debugschermen met daarin de variabelen en hulpvariabelen kan definiëren en manipuleren. In hoofdstuk 7 worden de overige hulpprogramma's van de RWS C -regelaar beschreven.

Bijlage 1 geeft een overzicht van alle foutmeldingen die door de RWS C -regelaar gegeven kunnen worden.

Tot slot wordt er in bijlage 2 een overzicht gegeven van alle mogelijke referenties en instructies van de RWS C -regelaar.

1.2 De RWS C-regelaar

De RWS C-regelaar is een tool voor het ontwerpen en specificeren van programma's voor verkeersregelinstallaties (regelprogramma's) op kruispunten, gebaseerd op de basisspecificatie [BS]. De basisspecificatie bevat de op een bepaalde regelfilosofie gebaseerde basisregels voor verkeerslichtenregelingen.

De basisspecificatie is door de Dienst Verkeer en Scheepvaart (voorheen Adviesdienst Verkeer en Vervoer (AVV) en Dienst Verkeerskunde (DVK)) van de Rijkswaterstaat (RWS) ontwikkeld rond 1980 met als hoofddoel: de mogelijkheid te bieden op een efficiënte en relatief eenvoudige wijze flexibele voertuigafhankelijke verkeerslichtenregelingen te maken. In 1991 is de basisspecificatie in C geprogrammeerd en is de RWS C-regelaar ontstaan (CR). In 1999 is een versie van de RWS C-regelaar zonder blokprocedure ontwikkeld: de RWS C-regelaar speciaal (CS). In 2014 zijn de CR- en de CS-versie geïntegreerd. In deze nieuwe versie is het blokkenschema wijzigbaar en zijn o.a. de hiaattijden niet langer per signaalgroep, maar per detector.

De basisspecificatie is vastgelegd in [BS]. Er is een toelichting [TOEL_BS] op de basisspecificatie beschikbaar. De RWS C-regelaar wordt in brede kring als ontwerpstandaard aanvaard en gebruikt in Nederland en er worden regelmatig cursussen in gegeven.

De RWS C-regelaar is opgebouwd uit drie componenten, namelijk de basisspecificatie, de applicatiespecificatie en de procesbesturing. Deze drie componenten worden nu kort toegelicht.

1.2.1 De basisspecificatie

Dit is een specificatie van de basisregels die voor iedere regeling gelden en waarin twee niveaus onderscheiden worden, te weten:

1. De blokprocedure
   Elke signaalgroep (een richting die verkeerslantaarns met dezelfde groen-geel-rood opeenvolging stuurt), wordt ingedeeld bij een blok. Dit is een groepje van richtingen dat in principe tegelijkertijd groen kan worden. Deze blokken worden in een zgn. blokkenketen gezet: dit is de volgorde waarin ze groen kunnen krijgen.
   De blokprocedure legt alle mogelijke volgorden van afwikkeling van signaalgroepen vast. Dat betekent dat is voorgeschreven hoe het doorlopen van een blokkenketen op de standaardmanier hoort te gebeuren en welke variaties daarop zijn toegestaan.
   Overigens is het ook mogelijk een regeling te programmeren zonder blokkenschema (bijvoorbeeld een halfstarre regeling met een vaste cyclustijd).

2. De signaalgroepprocedure
   Binnen de roodfase en groenfase van een signaalgroep worden verschillende toestanden (soorten rood respectievelijk groen) onderscheiden. Zo is er een toestand ‘recht op groen’ waarbij een signaalgroep op rood staat en aan de beurt is om naar groen geschakeld te worden, omdat bijv. het blok waarin hij zit actief is. Binnen groen zijn er bijv. toestanden waarbij de groentijd wordt verlengd met het oog op verkeer dat zich heeft gemeld via een detectielus op grotere afstand van de stopstreep.
   In de signaalgroepprocedure ligt vast volgens welke voorwaarden de overgangen tussen deze verschillende toestanden mogen plaatsvinden.

1.2.2 De applicatiespecificatie

Dit is een specificatie van de verkeersregeltechnische gegevens en voorwaarden die voor een bepaalde regeling gelden. De ontwerper van een verkeersregelprogramma volgens de basisspecificatie definieert hierin voor zijn kruispunt de signaalgroepen, de conflicten tussen signaalgroepen, de detectie, etc. Verder kan hij hierin allerlei facilitaire opties en voorwaarden met behulp van logische formules specificeren in door de basisspecificatie voorgedefinieerde tabellen en functies.

1.2.3 De procesbesturing

De procesbesturing is de software die zorgt voor de communicatie tussen het applicatieprogramma en het verkeerssimulatieproces of de verkeersregelautomaat. De procesbesturing zorgt voor de uitwisseling van signaalgroeptoestanden, detectietoestanden en regelparameters. Deze uitwisseling vindt plaats volgens de afspraken van de commissie C-interface [Cie_C], een beschrijving van de scheiding tussen:

1. de noodzakelijke programmatuur van de fabrikant (de procesbesturing) en
2. de programmatuur, die op basis van een fabrikantonafhankelijk programmeersysteem door de ontwerper (gebruiker) van het regelprogramma moet worden gemaakt (applicatieprogramma); hier dus de basisspecificatie samen met de applicatiespecificatie.

De beschrijving van de interface laat voor bepaalde aspecten ruimte over voor nadere afspraken tussen de fabrikant en de wegbeheerder (schrijver van het applicatieprogramma). Voor de RWS C-regelaar zijn die nadere afspraken met de fabrikanten centraal geregeld en vastgelegd in een aanvulling [AANV_CIE_C] op het document van de CVN. Bij deze aanvulling is tevens een bestektekst gevoegd die in samenhang met het RWS standaardbestek voor verkeersregelinstallaties kan worden gebruikt bij toepassing van de RWS C-regelaar. Beide bestanden zijn te vinden op de internetsite van de RWS C-regelaar (zie hoofdstuk 1.5).

1.3 Test- en ontwikkelomgeving FLASH

Voor de RWS C-regelaar is de test- en ontwikkelomgeving FLASH (FLEXSYT Application SHell) nodig. De belangrijkste mogelijkheden van FLASH zijn:

- tekenen kruispuntplaatje
- ontwerpen en genereren FLEXSYT-netwerk
- koppelen signaalgroepen en detectoren aan het FLEXSYT -netwerk
- aanpassen en compileren applicatiebestanden
- visualisatie werking verkeersregeling door middel van handmatige test en/of FLEXSYT-II simulatie
- aansturen hulpprogramma's

Het tekenen van het kruispuntplaatje en het ontwerpen van het FLEXSYT-netwerk gebeurt in de netwerkeditor. Om een goede weergave van de regeling(en) te kunnen maken, moeten de signaalgroepen en detectoren aan de juiste elementen in het/de regelprogramma's worden gekoppeld. Deze koppeling gebeurt ook in de netwerkeditor.

De applicatiebestanden kunnen op een eenvoudige manier worden aangepast en gecompileerd.

De visualisatie van de werking van het/de regelprogramma's kan met de hand (met een muis aan- en uitzetten van de detectoren), random en plaatsvindendoor simulatie van het verkeersproces van FLEXSYT-II plaatsvinden. Door de koppeling van het kruispuntplaatje aan FLEXSYT worden de rood-, groen-, geelovergangen en detectoraansturingen in kleur getoond.

Bovendien kan via een gebruikersvriendelijk debugprogramma bijna alle interne informatie van de regelaar, zoals signaalgroepwisselingen, blokwisselingen, etc., on line gevolgd worden. De gebruiker kan hiertoe met een hulpprogramma debugschermen genereren of zelf samenstellen.

Na het ontwerpen, programmeren en testen van de regeling kan de broncode van de regeling aan de fabrikant worden aangeboden en machinaal worden omgezet in voor het verkeersregeltoestel geschikte programmatuur.

In hoofdstuk 6 wordt een beschrijving van de programma's FLASH en FLEXSYT-II gegeven.

1.4 Gebruik RWS C-regelaar

1.4.1 Benodigde hard- en software

In tabel 1.2 is de voor gebruik van de RWS C-regelaar vereiste hardware- en software-omgeving beschreven.

hardware	**software**
pc	minimaal Windows 7
	Microsoft C++ compiler vanaf versie 2010
	FLASH
	FLEXSYT-II
	RWS C-regelaar

Tabel 1.2: Benodigde apparatuur en programmatuur

1.4.2 Installatie van de RWS C-regelaar

Directorystructuur
De code van de RWS C-regelaar wordt digitaal beschikbaar gesteld. Er is een installatieprogramma beschikbaar.

Systeembestanden
Hiernavolgend wordt kort de standaard bestandsstructuur van de RWS C-regelaar beschreven. Bij de belangrijkste bestanden is een korte toelichting gegeven:

- in directory CRW\CRALG\BIN
  - . CRDEBUG.EXE: de debugger voor afnametesten, waarin tevens opgenomen de functionaliteit van DPCOM
  - . FLXRWSW.EXE: FLEXSYT-deel binnen de RWS C-regelaar (Windows-versie)
  - . MKDEBFLS.EXE: hulpprogramma voor het genereren van debugbestanden
  - . POSTDUMP.EXE: hulpprogramma voor het omzetten van een dumpbestand naar een functioneel leesbaar bestand
  - . CRDEBUG.DLL: de debugger voor de FLASH-omgeving
  - . CRDEBUG.ICO: icoon voor de debugger

- in directory CRW\REG
  - . bevat de kruispuntdirectories met de applicatiespecificaties van de verschillende kruispunten, er is één demoregeling aanwezig

- in directory CRW\REG\DEFAULT
  . bevat drie files met daarin een lege applicatiespecificatie, op basis waarvan een nieuwe regeling gemaakt kan worden: CRAPDEF.H, CRAPTAB.C en CRAPCOD.C
  . MAKEFILE: de te gebruiken makefile

- in directory CRW\STANDARD\CRFPBRT
  . CONVHDR.C: hulpbestand voor CONVGEN.EXE
  . CONVGEN.EXE: programma (dat automatisch bij compileren wordt aangeroepen) om een CONV.C-bestand aan te maken, wordt gebruikt voor interne hernummering blokken, signaalgroepen en detectoren
  . CRFPBRT.H: nodig voor gebruik procesbesturing binnen FLASH
  . CRFUTILS.H: nodig voor gebruik printxy-functionaliteit

- in directory CRW\STANDARD\DEFS
  . CRAPCOD.H: bevat de AVV-macro-definities voor het coderen van CRAPCOD.C
  . CRAPP.H: bevat de definities van de tabellenin CRAPTAB.C
  . CRAPTAB.H: bevat definities van structures en symbolische namen in CRAPTAB.C
  . CRBK.H: bevat macro-definities en prototypes van hulpvariabelen voor blokken alleen aanwezig in CR-versie, niet voor CS-versie
  . CRBKO.H: bevat supportfuncties voor het blokobject
  . CRDTO.H: bevat functies en macro's voor het detectorobject
  . CRHFO.H: bevat functies en macro's voor het hulpvariabele-object
  . CRSG.H: bevat macrodefinities en prototypes van hulpvariabelen voor signaalgroepen
  . CRSGO.H: bevat functies en macro's voor het signaalgroepobject
  . CRTMO.H: bevat functies en macro's voor het timerobject
  . CRTYPES.H: bevat gemeenschappelijke macro-definities (zoals elementaire datatypen)

- in directory CRW\STANDARD\EVTDISP
  . EVTDINTF.H: event dispatcher van FLASH, zie handleiding FLASH (bestanden verzorgen communicatie tussen FLASH en de procesbesturing van de regelaar)
  . EVTDISP.H
  . EVTDINTF.LIB
  . EVTDISP.LIB

- in directory CRW\STANDARD\LIB
  . CRW.LIB: de library van CR

- in directory CRW\STANDARD\OBJECTS
  . CRBKO.D: bevat datastructures in het blokobject
  . CRDTO.D: bevat datastructures in het detectorobject
  . CRHFO.D: bevat datastructures in het hulpvariabele-object
  . CSGO.D: bevat datastructures in het signaalgroepobject
  . CRTMO.D: bevat datastructures in het timerobject

- in directory CRW\STANDARD\PBRT
  . CCIE_INT.C: Commissie C-interface naar CR(S)W
  . CRAPDIM.C: bevat de dimensies van de regelaar (zoals aantal blokken, aantal signaalgroepen, etc.)
  . CCIE_INT.H: headerfile ten behoeve van applicatiespecificatie
  . RWS_DIM.H: headerfile waarin de dimensies van de RWS C-regelaar zijn vastgelegd

  . RWS_DIM2.H: idem
  . CIF.INC: includefile waarin CVN commissie C-interface is vastgelegd
  . FC_INTRN.INC: includiefile voor interne signaalgroeptoestanden
  . RWS_CIF.INC: includefile met de RWS C-regelaar aanvullingen op de CIF.INC file

- in directory CRW\STANDARD\RWSCREG
  . RWSCREG.C: C-regelaar initialisatie code voor de koppeling FLASH - C-regelaar
  . RWSCREG.H

Overige acties
In hoofdstuk 5.2 is beschreven welke acties de gebruiker moet uitvoeren om de RWS C-regelaar te kunnen gebruiken.

1.5 Internetsite

De RWS C-regelaar heeft een internetsite: *www.rwscregelaar.nl*. Op deze site staan diverse standaarden, voorbeelden en updates die gedownload kunnen worden. Ook zijn er enkele documenten opgenomen die belangrijk zijn voor het schrijven van bestekken met verkeersregelingen gespecificeerd in de RWS C-regelaar. Verder kan er via email informatie worden opgevraagd.

Het grootste deel van de standaarden en voorbeelden is opgesteld door de Werkgroep Standaard RWS C Applicaties. Een WSRA-standaard bestaat uit een beschrijving van de standaard (Word-document) en een voorbeeldtoepassing. Een voorbeeld bestaat uit een beschrijving (Word-document), vaak een include-file (*.inc) en een voorbeeldtoepassing. Onder updates staan de meest recente versies van enkele hulpprogramma's.

2. De programmeertaal C

2.1 De Microsoft Visual C++-compiler

De applicatiespecificaties volgens de basisspecificatie, die de gebruiker kan invoeren in de RWS C-regelaar, moeten uiteraard zijn geschreven in de hogere programmeertaal C. Zo'n applicatiespecificatie is dus het deel van het totale regelprogramma, dat de gebruiker als invoer voor de RWS C-regelaar maakt. De RWS C-regelaar zelf bevat de rest van het totale regelprogramma, namelijk de basisspecificatie met de overige software.

De RWS C-regelaar bevat de basisspecificatie in de vorm van objectprogramma's. Als een applicatiespecificatie eenmaal rijp is om te testen, hoeft de gebruiker dus maar twee stappen te doen om het totale regelprogramma in machinetaal te vertalen, namelijk:
1. de applicatiespecificatie door een C-compiler laten vertalen naar een objectprogramma
2. het verkregen objectprogramma en de objectprogramma's van de RWS C-regelaar die samen de basisspecificatie bevatten, door een linker tot een executeerbaar programma laten samenvoegen.

De RWS C-regelaar en FLASH maken het de gebruiker, door middel van een interactieve bediening, erg gemakkelijk deze stappen uit te voeren; op de gedetailleerde invulling hiervan komen we later terug.

Om de RWS C-regelaar te kunnen gebruiken, dient de gebruiker Microsoft Visual C++-compiler te installeren. Bij het ontwikkelen van de RWS C-regelaar is gekozen voor deze C-compiler. Een reden hiervoor is geweest dat om de objectfiles van FLEXSYT uit Microsoft-FORTRAN mee te kunnen linken, de RWS C-regelaar ook objectfiles van Microsoft moest leveren.

2.1.1 Installatie van de Microsoft Visual C++-compiler

Microsoft Visual C++ heeft een eigen installatieprocedure. Een standaardinstallatie is voldoende. In hoofdstuk 5.2 worden de benodige acties beschreven die genomen moeten worden om de regelaar onder FLASH te kunnen gebruiken.

2.2 Afspraken met betrekking tot extensies

Bij gebruik van de C-compiler gelden er bindende afspraken m.b.t. de extensies van bepaalde typen files:
- een source file (die een bronprogramma in C bevat) heeft de extensie .'C'
- een include file heeft de extensie .H. Include files bevatten definities, die worden gebruikt in het bronprogramma. De letter H is afkomstig van de term 'header file', zoals een include file ook vaak wordt genoemd, omdat er in de kop van het bronprogramma aan wordt gerefereerd. Deze referentie wordt namelijk bij de start van het compileren van de source direct vervangen door zijn inhoud
- bij een include file met als inhoud een standaardfunctionaliteit wordt bij de RWS C-regelaar de extentie .'INC' gebruikt
- een object file (resultaat van een gecompileerde source file) heeft de extensie .'OBJ'
- een bibliotheekfile heeft de extensie .'LIB'. Een bibliotheekfile bevat een aantal functies voor gemeenschappelijk gebruik door verschillende programma's
- uitvoerbare programma's in MS-DOS en windows hebben de extensie .'EXE'
- programma's (zoals de C-regelaar) die niet uit zichzelf hoeven te draaien maar onder Windows aangeroepen worden vanuit andere programma's hebben de extentie .'DLL'.

3. De applicatiespecificatie in C

3.1 Vorm van de invoer

3.3.1 De drie onderdelen van de applicatiespecificatie

Hiernavolgend wordt kort uitgelegd hoe een applicatiespecificatie er als invoer voor de RWS C-regelaar uit moet zien. In de drie volgende paragrafen wordt de syntaxis van de applicatiespecificatie voor een RWS C-regelaar beschreven aan de hand van het kruispunt uit figuur 3.1.
De applicatiespecificatie in C bestaat uit drie files, te weten:

1. CRAPDEF.H (C-Regelaar APplicatie DEFinitie)
   In deze file geeft de gebruiker de nummers van de signaalgroepen in de regeling en het aantal blokken in de regeling. Verder wordt nog vastgelegd, welke detectoren, tijden, etc. in de regeling aanwezig zijn.
   In deze file staan dus de elementen die het kruispunt definiëren voor gebruik van de basisspecificatie.

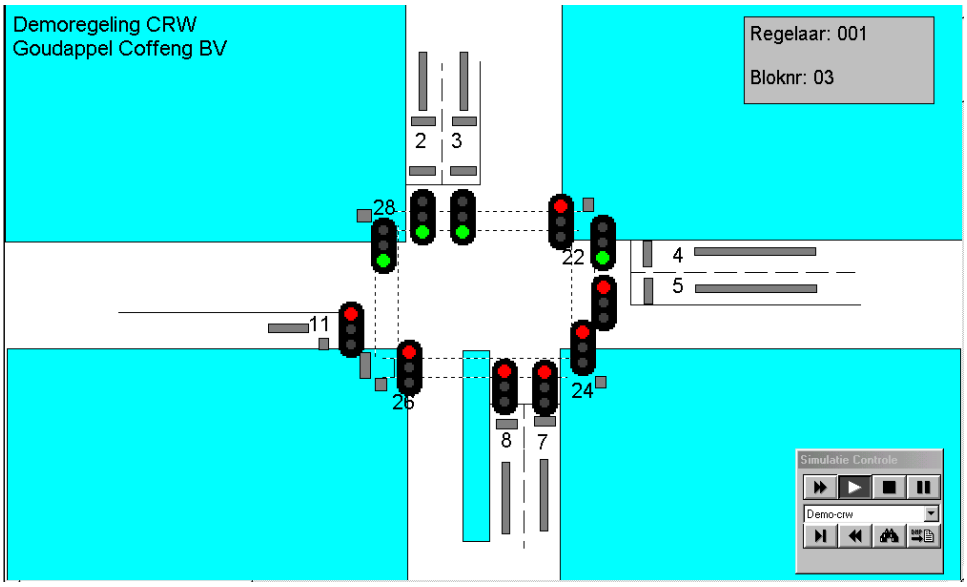
2. CRAPTAB.C (C-Regelaar APplicatie TABellen)
   In deze file worden de toewijzigingen aan de tabellen van de applicatiespecificatie gedaan. Het blokkenschema wordt ingevuld: welke signaalgroepen zitten in welk blok, en zitten ze primair of alternatief in dat blok. Verder wordt het nummer van het eventuele wachtblok ingevuld.
   Daarna worden de conflictmatrix met de garantie- en ontruimingstijden ingevuld. Verder kunnen ook alle tabellen met tijden, klokken, opties, etc., van de signaalgroepprocedure en de blokprocedure worden ingevuld.

3. CRAPCOD.C (C-Regelaar APplicatie CODering)
   Hierin worden alle applicatieformules ingevuld, die betrekking hebben op de zogenaamde overige voorwaarden. Zo kunnen bijvoorbeeld standaard functionaliteiten via include-files (.INC) worden opgenomen.

In de directory CRW\REG\DEFAULT zijn lege versies van de drie applicatiefiles beschikbaar.

Met name met betrekking tot de onderwerpen uit dit hoofdstuk is het verstandig deze handleiding te gebruiken in combinatie met de basisspecificatie [BS].

3.1.2 Voorbeeldkruispunt



Figuur 3.1: Voorbeeldkruispunt

3.2 Definitie van het kruispunt in CRAPDEF.H

3.2.1 Algemeen

In figuur 3.2 op de volgende pagina is een file CRAPDEF.H gegeven, zoals die voor het kruispunt uit figuur 3.1 gespecificeerd kan worden. Tekst tussen /* en */ of na // wordt in C beschouwd als commentaar.

In deze file wordt vastgelegd met welke symbolische namen de gebruiker de verschillende grootheden in zijn applicatiespecificatie aanduidt. Zo'n declaratie van symbolische namen wordt volgens de syntaxis van C aangegeven door

#define *symbolische naam arraynummer*

Het effect van deze declaratie is dat bij het compileren van de source overal in de tekst van de source de symbolische naam wordt vervangen door het arraynummer. In het geval van CRAPDEF.H worden deze namen intern in de RWS C-regelaar als index in diverse arrays en structures gebruikt, vandaar dat ze ook wel arraynummers genoemd worden.

3.2.2 Het kopje van CRAPDEF.H

Het is gebruikelijk de file te laten beginnen met wat commentaar om aan te geven wat de inhoud van de file is, wie er verantwoordelijk is voor die inhoud en gegevens voor versiebeheer. Dit commentaar is niet verplicht maar wel handig ter identificatie van de file. Ook kunnen wijzigingen in dit commentaar worden bijgehouden.

3.2.3 Inhoud van CRAPDEF.H

Eerst moeten de symbolische namen van de blokken worden gedeclareerd. In het voorbeeld zitten er 4 blokken in het blokkenschema: als namen voor de blokken moeten BK1, BK2, BK3, etc. gebruikt worden. De arraynummers moeten worden genummerd vanaf 0.

Op dezelfde manier worden de namen van respectievelijk
- de signaalgroepen, en
- de detectoren, tijden, hulpvariabelen, klokken, schakelaars, tellers en extra parameters gedeclareerd. Net zoals voor de blokken de namen BK1, etc. zijn gekozen, zijn ook hier weer suggestieve, functionele namen gebruikt.

```
/*
 * Copyright(c) 2014   Rijkswaterstaat
 *                     WVL
 * File: CRAPDEF.H
 *
 * Nummer kruispunt:        1
 * Regelprogramma vri:      Demoregeling RWS-C-REGELAAR
 *
 * Dienst:                  Goudappel Coffeng BV
 * Ontwerper:               Theo Dijkshoorn
 *
 * datum:                   14-02-2014
 * versie:                  1
 */

/* declaratie arraynummers voor blokken */
#define BK1                       0
#define BK2                       1
#define BK3                       2
#define BK4                       3

#define NUMBK                     4

/* declaratie arraynummers voor signaalgroepen */
#define SG02                      0
#define SG03                      1
#define SG04                      2
#define SG05                      3
#define SG07                      4
#define SG08                      5
#define SG11                      6
#define SG22                      7
#define SG24                      8
#define SG26                      9
#define SG28                     10

#define NUMSG                    11

/* declaratie arraynummers voor detectoren  */
#define SG02_1                    0
#define SG02_2                    1
#define SG02_3                    2
#define SG03_1                    3
#define SG03_2                    4
#define SG03_3                    5
#define SG04_1                    6
#define SG04_2                    7
#define SG05_1                    8
#define SG05_2                    9
#define SG07_1                   10
#define SG07_2                   11
#define SG08_1                   12
```

```
#define SG08_2                  13
#define SG11_1                  14
#define SG11_2                  15
#define SG22_1                  16  /* DK  */
#define SG24_1                  17  /* DK  */
#define SG26_1                  18  /* DK  */
#define SG26_2                  19
#define SG28_1                  20  /* DK  */

#define NUMD                    21

/* declaratie arraynummers voor tijden                    */
#define T_BO02                   0  /* tbv BO_optie      */
#define T_BO03                   1
#define T_BO04                   2
#define T_BO05                   3
#define T_BO07                   4
#define T_BO08                   5
#define T_BO22                   6
#define T_BO24                   7
#define T_BO26                   8
#define T_BO28                   9
#define T_PPA03                 10  /* tbv alternatief   */
#define T_PPA03P                11
#define T_PPA04                 12
#define T_PPA04P                13
#define T_PPA05                 14
#define T_PPA05P                15
#define T_PPA07                 16
#define T_PPA07P                17
#define T_PPA08                 18
#define T_PPA08P                19
#define T_PPA22                 20

#define NUMT                    21

/* declaratie arraynummers voor hulpfuncties              */
#define H_RVG02                  0 /* tbv deelconflicten */
#define H_RVG28                  1
#define KU_MG_OCHTEND            2
#define KU_MG_AVOND              3
#define KI_FILE_02_300           4
#define H_BLOK_GEWYZIGD          5

#define NUMHF                    6

/* declaratie arraynummers voor klokken */
#define OCHTEND                  0
#define AVOND                    1

#define NUMKL                    2

/* declaratie arraynummers voor schakelaars                                */
#define S_MA_04_03               0  /* wel/geen mee-aanvraag 04 van 03   */
#define S_MA_07_05               1  /*  "    "    "    "     07 van 05   */
#define S_WACHTST_ROOD           2  /* wachtstand groen / rood           */

#define NUMSCH                   3

/* declaratie arraynummers voor tellers                  */
#define TL_FB02                  0
#define TL_FB03                  1
#define TL_FB04                  2
#define TL_FB05                  3
#define TL_FB07                  4
#define TL_FB08                  5
#define TL_FB11                  6
#define TL_FB22                  7
#define TL_FB24                  8
#define TL_FB26                  9
```

```
#define TL_FB28                 10

#define NUMTL                   11

/* declaratie arraynummers voor extra geheel getal parameters           */
#define EG_FB                    0

#define NUMEGGP                  1

/* declaratie arraynummers voor extra parameters           */
#define EP_MG                    0

#define NUMEP                    1

#define NUMOV                   21


/* declaratie arraynummers voor overige ingangssignalen     */
#define NUM_OVI                  0

/* declaratie arraynummers voor overige uitgangssignalen    */
#define OVU_MG_OCHTEND           0
#define OVU_MG_DAL               1
#define OVU_MG_AVOND             2
#define NUM_OVU                  3
```

Figuur 3.2: Voorbeeld CRAPDEF.H

3.2.4 Naamgeving

De naamgeving BK. (voor blokken), SG.. (voor signaalgroepen) respectievelijk SG.._. (voor detectoren) uit het voorbeeld is een standaardnotatie voor de RWS C-regelaar. Door het gebruik van deze notatie, is het koppelen van detectoren aan het FLEXSYT-netwerk in FLASH mogelijk. Voor de overige elementen (tijden, hulpvariabelen, klokken, schakelaars, tellers en extra parameters) is de naamgeving “vrij” (maximaal 20 karakters en geen keywoorden van C).

3.2.5 Dimensies

De namen NUMBK, NUMSG, NUMOV en NUMEP na de vier respectievelijke declaraties, worden bij het starten van de RWS C-regelaar gebruikt om de groottes van diverse arrays en structures vast te stellen. Deze parameters zijn daarom van invloed op de doorloopsnelheid van de RWS C-regelaar.

De RWS C-regelaar gaat ervan uit dat bij NUMBK=i de blokken 0 tot en met i-1 gedefinieerd (= voorzien van een naam) zijn.
Bij NUMSG=n wordt ervan uitgegaan dat alle signaalgroepnummers in het gebied 0,..,n-1 liggen. Bij NUMOV=k wordt ervan uitgegaan dat alle detectornummers, tijdennummers,etc. in het gebied 0,..,k-1 liggen. Bij NUMOV=k kunnen er dus maximaal k detectoren, k tijden, k hulpvariabelen, k klokken, k schakelaars en k tellers gebruikt worden. Samen zijn dit maximaal 7 * k grootheden. Bij NUMEGGP=k wordt ervan uitgegaan dat alle extra geheel getal parameter nummers in het gebied 0,..,k-1 liggen.Bij NUMEP=k wordt ervan uitgegaan dat alle extra parameter nummers in het gebied 0,..,k-1 liggen.

De maximumwaarden in de RWS C-regelaar voor NUMBK, NUMSG, NUMOV, NUMEGGP en NUMEP zijn respectievelijk 10, 48, 256, 26.000 en 26.000. Het maximum aantal klokken dat in de klokkentabel kan worden opgegeven, is overigens beperkt tot 20. Voor het gebruik van het hulpprogramma POSTDUMP.EXE worden de volgende definities toegevoegd:

- NUMD: aantal detectoren
- NUMT: aantal tijdelementen

- NUMHF: aantal hulpvariabelen
- NUMKL: aantal klokken
- NUMSCH: aantal schakelaars
- NUMTL: aantal tellers;
- NUMEGGP: aantal extra geheel getal parameters;
- NUMEP: aantal extra parameters.
- NUM_OVI: aantal overige ingangssignalen;
- NUM_OVU: aantal overige uitgangssignalen.

De detectoren worden bij voorkeur aansluitend gedefinieerd (om problemen bij de implementatie in het verkeersregeltoestel te voorkomen. In FLASH/FLEXSYT-II kunnen maximaal 99 detectoren per signaalgroep worden gebruikt.

3.3 Declaratie van de tabellen in CRAPTAB.C

3.3.1 Algemeen

De file CRAPTAB.C bevat tabellen. De grootte van de tabellen is afhankelijk van het aantal regels dat de gebruiker invoert. Elke tabel moet worden beëindigd met het woord STOP, zodat de RWS C-regelaar het einde van die tabel kan onderscheiden. Tijdens het opstarten van de RWS C-regelaar voor een bepaalde applicatiespecificatie worden alle tabellen gelezen, beginnend met de eerste regel en eindigend bij STOP. Het woord STOP is dus bedoeld als referentie voor de RWS C-regelaar en wordt, als het ontbreekt, pas gemist als de RWS C-regelaar gaat draaien.

Bij het aanmaken van een regeling onder FLASH wordt een ‘lege’ CRAPTAB.C file naar de regelingdirectory (vanuit \reg\default directory) gekopieerd. Deze CRAPTAB.C file bevat voor alle tabellen al de begin- en einddelen. De applicatiebouwer kan zich dan concentreren op de daadwerkelijke inhoud van de tabellen.

Iedere tabel bevat één of meer regels en elke regel bevat één of meer items, die door een komma van elkaar gescheiden moeten worden. Een regel moet ook worden afgesloten met een komma. De volgorde van de regels is niet voorgeschreven, zodat er eventueel regels tussengevoegd kunnen worden.
Uiteraard moeten voor de grootheden uit de basisspecificatie de symbolische namen worden gebruikt zoals die in de definitiefile CRAPDEF.H vastgelegd zijn.

Alle tijden en extra parameters, met uitzondering van de klokken en extra geheel getal parameters, moeten worden ingevuld met een decimale punt (als 'floating point'), waarbij er tenminste één cijfer voor en maximaal één cijfer achter de punt moet staan (uitzondering: klokken 2 cijfers achter de punt). Intern worden alle tijden en extra parameters namelijk omgerekend naar een geheel getal dat het aantal tienden aangeeft.
Voorbeelden: een garantieontruimingstijd van één seconde wordt opgeschreven als ‘1.0’, een maximumgroentijd van tien seconden als ‘10.0’, en een extra parameter tweeënhalf als ‘2.5’. Intern worden respectievelijk de waarden 10, 100 en 25 gehanteerd.

Extra geheel getal parameters worden ingevuld als geheel getal (dus geen decimale punt). Voorbeeld: het jaartal 2003 wordt opgeschreven en gehanteerd als ‘2003’.

In de volgende subparagrafen worden de verschillende tabellen behandeld, die in CRAPTAB.C gedeclareerd moeten worden. Dit gebeurt aan de hand van het voorbeeld uit figuur 3.1 en uitgaande van de CRAPDEF.H uit figuur 3.2: van elke tabel wordt een voorbeeld gegeven en al deze voorbeelden samen vormen een gehele CRAPTAB.C.

Elke tabel die wordt besproken in deze paragraaf moet in de CRAPTAB.C worden opgenomen; ook de tabellen die leeg zijn. Er is geen volgorde voor de tabellen voorgeschreven, maar het wordt aanbevolen de volgorde uit deze handleiding aan te houden. De enige eis wat betreft volgorde is, dat de functies die bij opties worden gebruikt om booleaanse expressies te formuleren, worden gedeclareerd vóór de optie(s) waarin ze worden gebruikt (paragraaf 3.3.8).

3.3.2 Het kopje van CRAPTAB.C

Het is gebruikelijk de file te laten beginnen met wat commentaar om aan te geven wat de inhoud van de file is en wie er verantwoordelijk is voor die inhoud. Dit commentaar is niet verplicht maar wel handig ter identificatie van de file. Ook kunnen wijzigingen in dit commentaar worden bijgehouden.

```
/*
 * Copyright(c)  2014  Rijkswaterstaat
 *                     WVL
 * File: CRAPTAB.C
 *
 * Nummer kruispunt:        1
 * Regelprogramma vri:      Demoregeling CRW
 *
 * Dienst:                  Goudappel Coffeng BV
 * Ontwerper:               Theo Dijkshoorn
 *
 * datum:                   14-02-2014
 * versie:                  1
 */

#include "crtypes.h"
#include "crapcod.h"
#include "craptab.h"
#include "ccie_int.h"
#include "crapdef.h"
```

Figuur 3.3: Het kopje van CRAPTAB.C

De vijf include-regels uit figuur 3.3 die op het commentaar volgen zijn verplicht. Bij het compileren wordt namelijk de uitdrukking

#include *filenaam*

vervangen door de inhoud van de betreffende file. In de vijf files die altijd op deze wijze in de CRAPTAB.C moeten worden opgenomen, staan allerlei noodzakelijke definities die in de RWS C-regelaar gebruikt worden. Zo staan bijv. in de file CRAPTAB.H de definities van de structures die in CRAPTAB.C gebruikt gaan worden: de structures BLOKKEN, WB, CONFLICT, etc.

Het handige van deze include-faciliteit in C is voor de gebruiker, dat hij niet lastig gevallen wordt met heel veel tekst waar hij toch niets mee hoeft te doen.
In de makefile van de regeling (MAKEFILE) is, uitgaande van de in deze handleiding aanbevolen directorystructuur (paragraaf 1.4.2) vastgelegd waar de compiler deze includefiles kan vinden.

Als de gebruiker een andere directorystructuur heeft opgezet voor de RWS C-regelaar dan welke is aanbevolen, is het uiteraard mogelijk dat de includefiles ergens anders staan dan door de makefile wordt verondersteld. In dat geval moet de gebruiker zelf de juiste padnaam voor de includefiles aanbrengen in deze makefile. Dit wordt echter sterk afgeraden.

3.3.3 Blokkenschema en wachtstandrichtingen

Na het kopje van de CRAPTAB.C komen de tabellen voor het blokkenschema en de wachtstandrichtingen.

Declaraties ten behoeve van het blokkenschema en de wachtstandrichtingen
In figuur 3.4 is een voorbeeld van de twee tabellen voor het blokkenschema en de tabel voor de wachtstandrichtingen opgenomen.

```
/*
 * ============================================================
 * declaraties t.b.v. het blokkenschema
 * ============================================================
 */

/*     BLOKKEN schema                                          */
struct BLOKKEN  BLOKKEN_tabel[] = {
/*     blok:    sg:     toewijzing:                            */
/*     BKl,     SGn,    PRIM + ALT,                            */
       BK1,    SG02,    PRIM,
       BK1,    SG07,                ALT,
       BK1,    SG08,    PRIM,
       BK1,    SG24,    PRIM +       ALT,

       BK2,    SG02,                ALT,
       BK2,    SG03,    PRIM,
       BK2,    SG04,    PRIM,
       BK2,    SG26,    PRIM +       ALT,
       BK2,    SG28,    PRIM,

       BK3,    SG04,                ALT,
       BK3,    SG05,    PRIM,
       BK3,    SG07,    PRIM,
       BK3,    SG22,    PRIM +       ALT,
       BK3,    SG26,                ALT,
       BK3,    SG28,                ALT,

       BK4,    SG11,    PRIM,
       STOP};

/*     Wachtblok                                               */
struct WB  WB_tabel[] = {
/*     wachtblok:                                              */
       STOP};


/*
 * ============================================================
 * wachtstandrichtingen
 * ============================================================
 */

/*     Wachtstandrichtingen              */
struct WS_R  WSGR_tabel[] = {
/*     wachtstandrichting                */
/*     SGn,                              */
       SG02,
       SG08,
       STOP};
```

Figuur 3.4: Declaraties ten behoeve van het blokkenschema en de wachtstandrichtingen

Het blokkenschema
Per signaalgroep wordt in deze tabel minimaal één regel geplaatst. Eerst wordt opgegeven in welk blok de signaalgroep voorkomt, dan het symbolische signaalgroepnummer, en daarna een aantal attributen gescheiden door het teken "+".
De attributen kunnen zijn:
- PRIM: voor PRIMaire signaalgroep van het betreffende blok
- ALT: voor een ALTernatieve signaalgroep van het betreffende blok.

Voorbeeld: in het blokkenschema van figuur 3.4 zitten de richtingen 26 en 28 als alternatieve signaalgroepen in blok 3, zoals gegeven door de regels

```
BK3,    SG26,                ALT,
BK3,    SG28,                ALT,
```

en als primaire signaalgroepen in blok 2, zoals blijkt uit

```
BK2,    SG26,    PRIM +                ALT,
BK2,    SG28,    PRIM,
```

Richting 26 heeft in dit geval ook in blok 2 als alternatieve signaalgroep.

Als een signaalgroep meerdere keren voor hetzelfde blok wordt gedefinieerd, is alleen de laatste definitie geldig (niet doen dus).

Het wachtblok
In de wachtbloktabel wordt maar één regel ingevuld met daarin de symbolische naam van het wachtblok. Als er geen wachtblok is, is deze tabel leeg (zoals in tabel 3.4): ze bevat dan alleen de indicator STOP.

De wachtstandrichtingen
De richtingen in deze tabel zijn de wachtstand groen richtingen. In dit voorbeeld zullen de signaalgroepen 2 en 8 groen worden en in wachtgroen blijven als er geen activiteit op het kruispunt is. Als er geen wachtstandrichtingen zijn, is deze tabel leeg: ze bevat dan alleen de indicator STOP.

3.3.4 De matrices

De matrices uit de basisspecificatie zijn allemaal opgezet in tabelvorm. De twee signaalgroepnummers die de rij en de kolom in de matrix aangeven, worden per regel in de tabel genoemd. De RWS C-regelaar genereert bij het opstarten uit deze tabellen de matrices. Als een bepaalde matrix leeg is, bevat de tabel die die matrix weergeeft alleen de indicator STOP.

De conflict- en (garantie)ontruimingstijden-matrix
Deze matrix in tabelvorm bevat per conflict een regel met de volgende informatie: eerst wordt de naam van de ontruimende signaalgroep genoemd en dan de naam van een oprijdende conflicterende signaalgroep; vervolgens de garantie-ontruimingstijd en de ontruimingstijd van de eerste naar de tweede signaalgroep.

Tijden worden aangegeven als ‘floating point', zoals in paragraaf 3.3.1 is uitgelegd.

Per regel wordt een enkelzijdig conflict gedefinieerd. Het bijbehorend conflict van de tweede naar de eerste signaalgroep, wordt apart gedeclareerd en kan natuurlijk andere waarden voor de (garantie)ontruimingstijden bevatten.

Voorbeeld uit figuur 3.5 op de volgende pagina: de garantie-ontruimingstijd en de ontruimingstijd van signaalgroep 2 naar signaalgroep 5 bedragen beide 0.0 seconden, zoals blijkt uit

```
        SG02,     SG05,                 0.0,      0.0,
```

en de garantie-ontruimingstijd respectievelijk de ontruimingstijd van 11 naar 2 bedragen respectievelijk 1.0 en 2.0 seconden, volgens

```
        SG05,    SG02,                  1.0,     2.0,
```

```
/*
 * ================================================================
 * matrices
 * ================================================================
 */

/*     CONFLICT- en (garantie)ontruimingstijden matrix          */
struct CONFLICT  CONFLICT_tabel[] = {
/*                                    instellingen              */
/*                                    van de tijden:            */
/*     sg:       conflicterende sg:   GO_TIJD:  O_TIJD:         */
/*     SGn,      SGj,                 x.y,      x.y,            */
       SG02,     SG05,                0.0,      0.0,
       SG02,     SG11,                0.0,      1.0,
       SG02,     SG22,                0.0,      1.0,
       SG02,     SG26,                3.0,      3.0,

       SG03,     SG05,                5.0,      5.0,
       SG03,     SG07,                4.0,      4.0,
       SG03,     SG08,                3.0,      4.0,
       SG03,     SG11,                1.0,      1.0,
       SG03,     SG22,                1.0,      2.0,
       SG03,     SG24,                8.0,      8.0,

       SG04,     SG08,                0.0,      0.0,
       SG04,     SG11,                2.0,      3.0,
       SG04,     SG22,                6.0,      6.0,
       SG04,     SG24,                0.0,      0.0,

       SG05,     SG02,                1.0,      2.0,
       SG05,     SG03,                0.0,      1.0,
       SG05,     SG08,                0.0,      1.0,
       SG05,     SG11,                4.0,      4.0,
       SG05,     SG24,                0.0,      0.0,
       SG05,     SG26,                5.0,      6.0,
       SG05,     SG28,                3.0,      4.0,

       SG07,     SG03,                0.0,      0.0,
       SG07,     SG11,                0.0,      0.0,
       SG07,     SG24,                4.0,      5.0,
       SG07,     SG26,                0.0,      0.0,

       SG08,     SG03,                0.0,      0.0,
       SG08,     SG04,                0.0,      0.0,
       SG08,     SG05,                0.0,      1.0,
       SG08,     SG11,                5.0,      5.0,
       SG08,     SG22,                5.0,      5.0,
       SG08,     SG26,                0.0,      0.0,

       SG11,     SG02,                0.0,      1.0,
       SG11,     SG03,                1.0,      1.0,
       SG11,     SG04,                0.0,      0.0,
```

```
        SG11,     SG05,               2.0,      2.0,
        SG11,     SG07,               0.0,      1.0,
        SG11,     SG08,               0.0,      1.0,
        SG11,     SG22,               5.0,      5.0,
        SG11,     SG24,               4.0,      4.0,
        SG11,     SG26,               5.0,      5.0,
        SG11,     SG28,               0.0,      0.0,

        SG22,     SG02,               0.0,      1.0,
        SG22,     SG03,               0.0,      1.0,
        SG22,     SG04,               0.0,      0.0,
        SG22,     SG08,               0.0,      0.0,
        SG22,     SG11,               0.0,      0.0,
        SG22,     SG28,               3.0,      4.0,

        SG24,     SG03,               0.0,      0.0,
        SG24,     SG04,               2.0,      3.0,
        SG24,     SG05,               1.0,      2.0,
        SG24,     SG07,               0.0,      0.0,
        SG24,     SG11,               0.0,      0.0,

        SG26,     SG02,               0.0,      0.0,
        SG26,     SG05,               0.0,      0.0,
        SG26,     SG07,               3.0,      4.0,
        SG26,     SG08,               3.0,      3.0,
        SG26,     SG11,               0.0,      0.0,

        SG28,     SG05,               0.0,      0.0,
        SG28,     SG11,               3.0,      4.0,
        SG28,     SG22,               0.0,      0.0,
        STOP};
```

Figuur 3.5: De conflictmatrix

De fictief-conflictmatrix
Deze matrix in tabelvorm bevat per fictief conflict een regel met de namen van twee signaalgroepen. Ieder fictief conflict wordt dubbelzijdig geinterpreteerd, d.w.z. de eerstgenoemde signaalgroep conflicteert fictief met de laatstgenoemde en de laatstgenoemde conflicteert fictief met de eerstgenoemde.

Fictieve conflicten worden vaak opgenomen voor richtingen waarvan de realisatie gekoppeld moet plaatsvinden (bijvoorbeeld SG37 krijgt de conflicten van SG38 als fictief conflict en SG38 krijgt de conflicten van SG37 als fictief conflict). Fictieve conflicten zorgen bijvoorbeeld voor het gelijktijdig afsluiten van het aanvraaggebied van de betreffende richtingen.

De fictief-conflictmatrix in figuur 3.6 bevat een fictief conflict tussen de signaalgroepen 1 en 37.

```
/*     FICTief CONFLICT matrix                                  */
struct FICT_CONFLICT  FICT_CONFLICT_tabel[] = {
/*     sg:            fictief conflicterende sg:                */
/*     SGn,           SGj,                                      */
       SG01,          SG37,
       STOP};
```

Figuur 3.6: De fictief-conflictmatrix

Extra op te geven signaalgroepen volgens formule t uit BS_H5
In deze matrix kunnen paren van signaalgroepen opgegeven worden, die invloed hebben op de regeling volgens formule t (beïnvloeden einde wachtgroen) uit hoofdstuk 5 van de basisspecificatie [BS]. De signaalgroepen worden dubbelzijdig geinterpreteerd, dat wil zeggen de eerstgenoemde signaalgroep en de laatstgenoemde vormen een signaalgroepenpaar zoals bedoeld in formule t, en de laatstgenoemde signaalgroep en de eerstgenoemde vormen eveneens zo'n paar.

In het algemeen worden in deze tabel de fictieve conflicten van de wachtstandrichtingen opgenomen.

Deze matrix mag leeg zijn, zoals in figuur 3.7.

```
/*     extra op te geven SG'en k volgens BS_H5 formule t
*/
struct SG_BS_H5t  SG_BS_H5t_tabel[] = {
/*     sg:        sg k volgens BS_H5 formule t: */
/*     SGn,       SGk,   */
       STOP};                                 */
```

Figuur 3.7: Signaalgroepen volgens formule t uit BS_H5

Extra op te geven signaalgroepen volgens formule ac uit BS_H5
In deze matrix kunnen paren van signaalgroepen opgegeven worden, die invloed hebben op de regeling volgens formule ac (beïnvloeden einde meeverlenggroen) uit hoofdstuk 5 van de basisspecificatie [BS]. De signaalgroepen worden dubbelzijdig geinterpreteerd, d.w.z. de eerstgenoemde signaalgroep en de laatstgenoemde vormen een signaalgroepenpaar zoals bedoeld in formule ac, en de laatstgenoemde signaalgroep en de eerstgenoemde vormen eveneens zo'n paar.

In het algemeen worden in deze tabel alle fictieve conflicten opgenomen.

Deze matrix mag leeg zijn, zoals in figuur 3.8.

```
/*     extra op te geven SG'en j volgens BS_H5 formule ac       */
struct SG_BS_H5ac  SG_BS_H5ac_tabel[] = {
/*     sg:        sg k volgens BS_H5 formule ac:                */
/*     SGn,       SGk,                                          */
       STOP};
```

Figuur 3.8: Signaalgroepen volgens formule ac uit BS_H5

Gekoppelde signaalgroepen volgens BS_H6.6
In deze matrix kunnen paren van signaalgroepen opgegeven worden, die alleen samen de overgang van WR naar ROG maken en waarvoor alleen gelijktijdig de overname van de ene realisatievorm naar een andere mag plaatsvinden, zoals dat beschreven is in paragraaf 6.6 van de basisspecificatie [BS]. De gekoppelde signaalgroepen worden dubbelzijdig geinterpreteerd, dat wil zeggen de eerstgenoemde signaalgroep is gekoppeld met de laatstgenoemde en de laatstgenoemde signaalgroep is ook gekoppeld met de eerste. In figuur 3.9 is deze matrix leeg.

```
/*     gekoppelde signaalgroepen volgens BS_H6.6                */
struct SG_BS_H66  SG_BS_H66_tabel[] = {
/*     sg:        gekoppelde sg:                                */
/*     SGn,       SGj,                                          */
       STOP};
```

Figuur 3.9: Gekoppelde signaalgroepen volgens BS_H6.6

3.3.5 Declaraties ten behoeve van de signaalgroepprocedure

De tabellen onder dit hoofdstuk bevatten naast de signaalgroeptijden en detectorgegevens ook extra kenmerken van signaalgroepen met betrekking tot voetgangers- of fietsrichtingen en richtinggevoelige aanvraag.

De signaalgroeptijdentabel
Per signaalgroep is in deze tabel een regel gereserveerd. Achtereenvolgens worden hier het signaalgroepnummer, de garantiegroentijd, de vastgroentijd, de garantiegeeltijd, de geeltijd en de garantieroodtijd geplaatst. Alle tijden worden met één cijfer achter de decimale punt opgegeven (zie paragraaf 3.3.1).

```
/*
 * ================================================================
 * declaraties t.b.v. de signaalgroepprocedure
 * ================================================================
 */

/*     SignaalGroep_TIJDEN tabel                                  */
struct SG_TIJDEN  SG_TIJDEN_tabel[] = {
/*            instellingen van de tijden:                         */
/*     sg:    GG_TIJD:  VG_TIJD:  GGL_TIJD:  GL_TIJD:  GR_TIJD: */
/*     SGn,    x.y,      x.y,      x.y,       x.y,      x.y,    */
       SG02,   5.0,      6.0,      4.0,       5.0,      2.0,
       SG03,   5.0,      6.0,      3.0,       4.0,      2.0,
       SG04,   5.0,      6.0,      3.0,       4.0,      2.0,
       SG05,   5.0,      6.0,      3.0,       4.0,      2.0,
       SG07,   5.0,      6.0,      3.0,       4.0,      2.0,
       SG08,   5.0,      6.0,      4.0,       5.0,      2.0,
       SG11,   5.0,      6.0,      2.0,       3.0,      2.0,
       SG22,   5.0,      6.0,      2.0,       3.0,      2.0,
       SG24,   5.0,      6.0,      2.0,       3.0,      2.0,
       SG26,   5.0,      6.0,      2.0,       3.0,      2.0,
       SG28,   5.0,      6.0,      2.0,       3.0,      2.0,
       STOP};
```

Figuur 3.10: Signaalgroeptijdentabel

Voor elke signaalgroep dient in deze tabel één regel met tijden aanwezig te zijn, zoals dat voor het beschouwde voorbeeld in figuur 3.10 gegeven is.

De maximumgroentijdentabel
In deze tabel kunnen per signaalgroep maximaal drie maximumgroentijden (MG-tijden) gedefinieerd worden, waaruit in de applicatiespecificatie een keuze kan worden gemaakt (met de instructie kies_MG_TIJDj(), waarbij 1<=j<=12). De keuze vindt tegelijkertijd plaats voor alle signaalgroepen.

Eerst wordt het signaalgroepnummer genoemd. Hierna kan men kiezen uit twee mogelijkheden:
1. Als de signaalgroep slechts één MG-tijd kent, benoemt men die vaste MG-tijd met daarna driemaal de symbolische naam ‘NG’ (Niet Gebruikt)
2. Als de signaalgroep meerdere MG-tijden kent, vult men op de eerste plaats ‘NG’ in, met daarna de gewenste drie MG-tijden.

Default worden voor de maximumgroentijd van een signaalgroep bij initialisatie de waarde gekozen zoals vermeld bij de MG_TIJD. Als bij de MG_TIJD NG is ingevuld, wordt default de waarden gekozen zoals vermeld bij der MG_TIJD1. Als ook bij MG_TIJD1 NG is ingevuld, wordt default de waarde 0 gekozen.

Voor elke signaalgroep met groenverlenging dient in deze tabel één ingevulde regel met tijden aanwezig te zijn, zoals in figuur 3.11; signaalgroepen die geen verlenging kennen, zoals voetgangers, hoeven echter niet in deze tabel te worden opgenomen (de betreffende regels mogen worden weggelaten). Zie ook paragraaf 4.2.5.

```
/*     MG_TIJDEN tabel                                          */
struct MG_TIJDEN  MG_TIJDEN_tabel[] = {
/*            instellingen (x.y of NG) van de tijden:           */
/*     sg:    MG_TIJD:  MG_TIJD1:  MG_TIJD2:  MG_TIJD3:         */
/*     SGn,    x.y,      x.y,       x.y,       x.y,            */
/*                       DALPERIODE OCHTEND    AVOND           */
       SG02,    NG,     40.0,      60.0,      60.0,
       SG03,    NG,     40.0,      70.0,      60.0,
       SG04,    NG,     40.0,      60.0,      70.0,
       SG05,    NG,     30.0,      45.0,      35.0,
       SG07,    NG,     20.0,      25.0,      25.0,
       SG08,    NG,     40.0,      60.0,      60.0,
       SG11,    NG,     10.0,      10.0,      10.0,
       SG26,    NG,     15.0,      15.0,      15.0,
       STOP};
```

Figuur 3.11: MG-tijdentabel

De extra-maximumgroentijdentabelen
Vaak zijn drie MG-tijden niet voldoende en is er behoefte om negen extra MG-tijden te benoemen. Dat kan met deze drie tabellen. Eerst wordt de signaalgroep genoemd met daarna drie extra MG-tijden (in de tweede tabel de sets 4, 5 en 6, in de derde tabel de sets 7, 8 en 9 en in de vierde tabel de sets 10, 11 en 12.
Let op: bij gebruikmaking van een MG-TIJD moeten alle signaalgroepen met MG-TIJDEN in die kolom worden gedeclareerd (dat wil zeggen een instelling of NG).

Deze tabellen kunnen ook leeg zijn, zoals in figuur 3.12.

```
/*     Extra MG_TIJDEN tabel                                    */
struct E_MG_TIJDEN  E_MG_TIJDEN_tabel[] = {
/*            instellingen (x.y of NG) van de tijden:           */
/*     sg:    MG_TIJD4:  MG_TIJD5:  MG_TIJD6:                   */
/*     SGn;    x.y,       x.y,       x.y,                      */
       STOP};

/*     Extra2 MG_TIJDEN tabel                                   */
struct E_MG_TIJDEN       E2_MG_TIJDEN_tabel[] = {
/*            instellingen (x.y of NG) van de tijden:           */
/*     sg:    MG_TIJD7:  MG_TIJD8:  MG_TIJD9:                   */
/*     SGn,    x.y,       x.y,       x.y,                      */
       STOP};

/*     Extra3 MG_TIJDEN tabel                                   */
struct E_MG_TIJDEN       E3_MG_TIJDEN_tabel[] = {
/*            instellingen (x.y of NG) van de tijden:           */
/*     sg:    MG_TIJD10: MG_TIJD11: MG_TIJD12:                  */
/*     SGn,    x.y,       x.y,       x.y,                      */
       STOP};
```

Figuur 3.12: Extra-MG-tijdentabellen

De signaalgroepdetectietabel
De signaalgroepdetectietabel koppelt detectoren aan signaalgroepen en definieert de functies van de detectoren. De tabel is als volgt opgebouwd: eerst wordt de detectornaam genoemd, daarna het type detector; vervolgens wordt de bij de detector behorende signaalgroep genoemd, vervolgens de eventuele bezettijd van de detector, daarna de hiaattijd en extra hiaattijd van de betreffende detector en tenslotte de functies die de detector voor die signaalgroep vervult.

In de RWS C-regelaar worden vier typen detectoren onderscheiden:
- DP representeert aanwezigheidsdetectoren (en selectieve detectoren)
- DK representeert drukknoppen
- DS representeert selectieve detectoren, deze worden intern als een DP geïnterpreteerd
- DV representeert detectoren die geschikt zijn om snelheid te meten

Aan het type van een detector kan een toevoeging worden gemaakt (+) om aan te geven wat de status van een detectiemelding moet zijn bij een detectiefout voor die detector. Deze toevoeging is niet verplicht. De toevoegingen zijn:
- DFA
- DFU.

Als aan het type van een detector DFA wordt toegevoegd ziet de regelaar voor de betreffende detector altijd een D zolang DFOUT || DFOUT_OG || DFOUT_BG waar is. Als aan het type van een detector DFU wordt toegevoegd ziet de regelaar voor de betreffende detector altijd een !D zolang DFOUT || DFOUT_OG || DFOUT_BG waar is.
In figuur 3.15 op de volgende pagina zijn er voor het gemotoriseerd verkeer dus alleen aanwezigheidsdetectoren opgegeven, voor de fietsrichtingen ieder een drukknop en voor fietsrichting 26 ook een aanwezigheidsdetector.

De functies van een detector kunnen zijn:
- AVR: aanvraagfunctie
- BS_H5p: detector als bedoelt in de basisspecificatie hoofdstuk 5 formule p
- H1e: beïnvloedt de eerste hiaattijd
- H2e: beïnvloedt de tweede hiaattijd
- GEEN: de betreffende detector krijgt geen functie toegewezen bij het initialiseren van de regelaar, maar later kunnen er m.b.v. de kiesinstructies wel één of meerdere functies aan deze detector worden toegewezen of kan er aan de status van de detector worden gerefereerd.

Een detector kan één of meer van de functies vervullen. Als hij meer functies vervult, wordt dat aangegeven met het scheidingsteken ‘+’ tussen die functies: zo heeft de tweede detector van signaalgroep 5 zowel een aanvraagfunctie als een functie voor de eerste hiaattijd, zoals blijkt uit:

```
SG05_2,  DP,          SG05,   NG, 0.0,  1.0,   AVR          + H1e,
```

Iedere detector mag slechts één keer genoemd worden in de tabel, m.a.w. een detector kan slechts aan één signaalgroep gekoppeld worden. Uiteraard kunnen er wel meerdere detectoren aan één signaalgroep gekoppeld worden.

Als een detector abusievelijk meerdere keren in de tabel voorkomt, is de laatste regel waarop de detector wordt genoemd bepalend.

```
/*     SignaalGroep-DETECTIE tabel                                        */
struct SG_DETECTIE           SG_DETECTIE_tabel[] = {
/*     D():     type:       sg:    bzttijd: hiaattijd: extra_hiaattijd: toewijzing:                 */
/*     SGn_i,   DP (of DK), SGn,   x.y,       x.y,       x.y,           AVR + BS_H5p + H1e + H2e,  */
       SG02_1,  DP + DFU,   SG02,  3.0,       3.0,       3.0,           BS_H5p,
       SG02_2,  DP + DFA,   SG02,   NG,       0.0,       1.0,           GEEN,
       SG02_3,  DP,         SG02,   NG,       3.0,       4.0,           AVR               + H2e,
       SG03_1,  DP,         SG03   3.0,       3.0,       3.0,           BS_H5p,
       SG03_2,  DP,         SG03,   NG,       0.0,       1.0,           GEEN,
       SG03_3,  DP,         SG03,   NG,       3.0,       4.0,           AVR               + H2e,
       SG04_1,  DP,         SG04,  3.0,       3.0,       3.0,           BS_H5p,
       SG04_2,  DP,         SG04,   NG,       0.0,       1.0,           AVR               + H2e,
       SG05_1,  DP,         SG05,  3.0,       3.0,       3.0,           BS_H5p,
       SG05_2,  DP,         SG05,   NG,       0.0,       1.0,           AVR          + H1e,
       SG07_1,  DP,         SG07,  3.0,       3.0,       3.0,           BS_H5p,
       SG07_2,  DP,         SG07,   NG,       0.0,       1.0,           AVR          + H1e,
       SG08_1,  DP,         SG08,  3.0,       3.0,       3.0,           BS_H5p,
       SG08_2,  DP,         SG08,   NG,       0.0,       1.0,           AVR          + H1e,
       SG11_1,  DK,         SG11,   NG,        NG,        NG,           AVR,
       SG11_2,  DP,         SG11,  3.0,       3.0,       3.0,                           H1e,
       SG22_1,  DK,         SG22,   NG,        NG,        NG,           AVR,
       SG24_1,  DK,         SG24,   NG,        NG,        NG,           AVR,
       SG26_1,  DK,         SG26,   NG,        NG,        NG,           AVR,
       SG26_2,  DP,         SG26,   2.5,      2.5,       2.5,                           H1e,
       SG28_1,  DK,         SG28,   NG,        NG,        NG,           AVR,
       STOP};
```

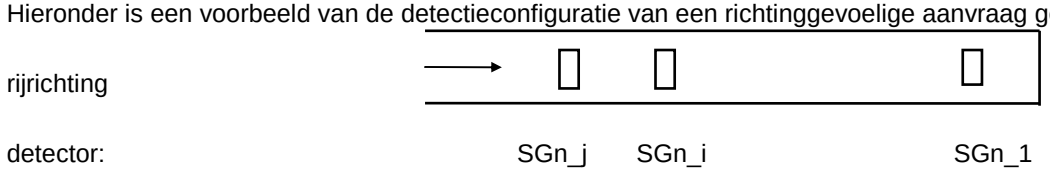
Figuur 3.15: Signaalgroepdetectietabel

Tabel voor signaalgroep met richtinggevoelige aanvraag

Via deze tabel is het mogelijk om richtinggevoelige aanvragen te definiëren, zoals omschreven in hoofdstuk 7 van de basisspecificatie [BS].

Eerst moet de eerste detector worden genoemd, daarna de tweede detector, en vervolgens de koplus; daarna de signaalgroep waarvoor de richtinggevoelige aanvraag gegenereerd moet worden en als laatste een paarnummer. Voor de opgegeven koplus geldt de bezettijd zoals voor deze lus opgegeven in de signaalgroepdetectietabel. Het starten van deze bezettijd wordt door de regelaar verzorgd.

Hieronder is een voorbeeld van de detectieconfiguratie van een richtinggevoelige aanvraag gegeven:



wordt als volgt in de tabel voor signaalgroepen met een richtinggevoelige aanvraag ingevuld:

1e detector : SGn_i
2e detector : SGn_j
koplus: SGn_1
signaalgroep: SGn

Per signaalgroep mogen er maximaal 4 paarnummers worden gebruikt, en wel de nummers 0, 1, 2 en 3. Wanneer er een richtinggevoelige aanvraag voor een richting wordt opgenomen, moet er geen AVR-functie in de signaalgroepdetectietabel voor de betreffende detector(en) worden opgenomen, maar moet er in de MA optie aan RGAH(SGn, i) worden gerefereerd (zie basisspecificatie hoofdstuk 7).

In figuur 3.16 is een voorbeeld van deze tabel opgenomen.

```
/*     tabel voor SG'en met RichtingGevoelige Aanvraag          */
struct SG_RGA     SG_RGA_tabel[] = {
/*     1e D():    2e D():    koplus D():  sg:   paarnr:  */
/*     SGn_i,     SGn_j,     SGn_k,       SGn,       x,  */
       SG02_1,    SG02_1,    SG02_1,      SG02,      0,
       SG03_1,    SG03_1,    SG03_1,      SG03,      0,
       SG04_1,    SG04_1,    SG04_1,      SG04,      0,
       SG05_1,    SG05_1,    SG05_1,      SG05,      0,
       SG07_1,    SG07_1,    SG07_1,      SG07,      0,
       SG08_1,    SG08_1,    SG08_1,      SG08,      0,
       SG26_2,    SG26_3,    SG26_1,      SG26,      0,
       STOP};
```

Figuur 3.16: Tabel voor richtinggevoelige aanvraag

Fiets- of voetgangersrichtingtabel
Normaliter is aan het volgnummer van een signaalgroep te zien of het een fiets- of voetgangersrichting betreft. De volgnummers van signaalgroepen zitten echter in de symbolische namen en niet in de (echte) namen, waarmee de RWS C-regelaar intern werkt (paragraaf 3.2.1): de informatie over de volgnummers gaat bij het compileren verloren.

Daarom is deze tabel aanwezig, waarin per regel één signaalgroep benoemd kan worden als fiets- of voetgangersrichting. In figuur 3.17 wordt de RWS C-regelaar meegedeeld dat signaalgroepen 22, 24, 26 en 28 fiets- of voetgangersrichtingen zijn, of preciezer: de signaalgroepen met de symbolische namen SG22, SG24, SG26 en SG28 (uit de CRAPDEF.H) zijn fiets- of voetgangersrichtingen.

In de basisspecificatie worden fiets- of voetgangersrichtingen afwijkend behandeld aan andere richtingen voor wat betreft het opzetten van de aanvraag op basis van detectormeldingen gedurende de garantieroodtijd (zie basisspecificatie hoofdstuk 5.4).

Deze tabel kan ook leeg zijn.

```
/*     FIETS of VOETGangersRichting tabel                       */
struct FIETS_VOETG_R  FIETS_VOETG_R_tabel[] = {
/*     fiets- of voetgangersrichting:                           */
/*     SGn,                                                     */
       SG22,
       SG24,
       SG26,
       SG28,
       STOP};
```

Figuur 3.17: Tabel met fiets- en voetgangersrichtingen

3.3.6 Declaratie ten behoeve van standaardapplicatie

Voor de of hulpvariabelen PPAH en VNMH uit hoofdstuk 7 van de basisspecificatie [BS] is één tabel gedefinieerd. In deze tabel kunnen de signaalgroepen worden gespecificeerd, die van deze hulpvariabelen gebruikmaken.

Signaalgroepen die gebruik maken van de hulpvariabelen PPAH en VNMH
In deze tabel bevat elke regel eerst de naam van een signaalgroep en vervolgens de naam van een bijbehorend tijdelement. Alleen voor de genoemde signaalgroepen worden de formules voor bepaling van de waarden van de hulpvariabelen PPAH en VNMH ingevuld. Voor de genoemde tijd in deze formules wordt het tijdelement gebruikt dat op dezelfde regel als de betreffende signaalgroep staat.

In figuur 3.18 is onder andere gespecificeerd dat signaalgroep 3 gebruikmaakt van de hulpvariabelen PPAH en VNMH. Als bij alternatieve realisaties wordt gewerkt met de PPAH(), moet de waarde van de tijd die hier is opgegeven (in figuur 3.18 is dat T_PPA03) worden opgenomen en ingesteld in de tijdentabel; in ons voorbeeld wordt T_PPA03 berekend aan de hand van de actuele maximum groentijd (zie figuur 3.19).

Het starten van deze tijden wordt door de regelaar verzorgd.

De tabel wordt ingevuld voor richtingen waaronder alternatief mag worden gerealiseerd (bijvoorbeeld SG02 mag alternatief realiseren tijdens een deel van de realisatie van SG03: dan moeten voor SG03 de betreffende regels worden ingevuld).

Deze tabel mag leeg zijn.

```
/*     tabel voor SG'en die gebruik maken van PPAH en VNMH      */
struct SG_PPAH  SG_PPAH_tabel[] = {
/*     PPAH() (en VNMH()):  TIJD():                             */
/*     SGn,                 SGn_i,                              */
       SG03,                T_PPA03,
       SG04,                T_PPA04,
       SG05,                T_PPA05,
       SG07,                T_PPA07,
       SG08,                T_PPA08,
       SG22,                T_PPA22,
       STOP};
```

Figuur 3.18: Tabel met betrekking tot de hulpvariabele PPAH en VNMH

3.3.7 Tijden, klokken, schakelaars, extra geheel getal parameters en extra parameters

De gebruiker kan tijden, klokken, schakelaars, extra geheel getal parameters en extra parameters definiëren. Deze staan in de volgende tabellen.

Tijdentabel
De tijdentabel bevat per regel de symbolische naam van een tijdelement gevolgd door de waarde waarop dat tijdelement wordt ingesteld. De waarde wordt opgegeven met één cijfer achter de decimale punt (zie ook paragraaf 3.3.1).

In figuur 3.19 is een voorbeeld van een tijdentabel gegeven.

De tijdentabel mag leeg zijn.

```
/*
 * ============================================================
 * tijden, klokken en schakelaars
 * ============================================================
 */

/*     TIJDENtabel                                             */
struct TIJDEN                     TIJDEN_tabel[] = {
/*     TIJD():                    instelling:                  */
/*     SGn_i,                      x.y,                        */
       T_BO02,                      5.0,  /* tbv BO_optie      */
       T_BO03,                      5.0,
       T_BO04,                      5.0,
       T_BO05,                      5.0,
       T_BO07,                      5.0,
       T_BO08,                      5.0,
       T_BO22,                      5.0,
       T_BO24,                      5.0,
       T_BO26,                      5.0,
       T_BO28,                      5.0,

       T_FB02,                    255.0,  /* tbv fasebewaking  */
       T_FB03,                    255.0,
       T_FB04,                    255.0,
       T_FB05,                    255.0,
       T_FB07,                    255.0,
       T_FB08,                    255.0,
       T_FB11,                    255.0,
       T_FB22,                    255.0,
       T_FB24,                    255.0,
       T_FB26,                    255.0,
       T_FB28,                    255.0,

       T_PPA03,                     0.0,  /* tbv alternatief real.       */
       T_PPA03P,                   10.0,  /* T_PPA.. worden berekend:    */
       T_PPA04,                     0.0,  /* T_PPA.. = MG - T_PPA..P     */
       T_PPA04P,                   10.0,
       T_PPA05,                     0.0,
       T_PPA05P,                   10.0,
       T_PPA07,                     0.0,
       T_PPA07P,                   10.0,
       T_PPA08,                     0.0,
       T_PPA08P,                   10.0,
       T_PPA22,                     3.0,  /* vaste waarde (geen MG_tijd) */

       T_RGA02,                     2.0,  /* bezettijden koplussen (RGA) */
       T_RGA03,                     2.0,
       T_RGA04,                     2.0,
       T_RGA05,                     2.0,
       T_RGA07,                     2.0,
       T_RGA08,                     2.0,
       T_RGA11,                     2.0,
       T_RGA26,                     2.5,
       STOP};
```

Figuur 3.19: Tijdentabel

Het starten van de tijden dient, voor zover dat niet door de regelaar zelf wordt verzorgd, elders in een functie te geschieden: bijv. in de functie ‘overige_voorwaarden()’ in de file CRAPCOD.C.

Klokkentabel
De klokkentabel bevat per regel de symbolische naam van een klokelement, gevolgd door twee kloktijdstippen voor respectievelijk de inschakeltijd en de uitschakeltijd. Een voorbeeld van de klokkentabel is opgenomen in figuur 3.20.

```
/*     KLOKKENtabel                                              */
struct KLOKKEN KLOKKEN_tabel[] = {
/*     KL():                    1e tijdstip:  2e tijdstip:       */
/*     NAAM,                     x.yz,         x.yz,             */
       OCHTEND,                  7.00,         9.30,
       AVOND,                   15.30,        18.00,
       STOP};
```

Figuur 3.20: Klokkentabel

Een klok KL is waar vanaf het eerste kloktijdstip tot aan het tweede kloktijdstip (ook over een dagwisseling heen). De kloktijdstippen worden éénmaal per seconde bijgewerkt, zodat het na het inschakelen van een regelaar of na het wijzigen van een kloktijdstip maximaal één seconde kan duren voordat een KL de juiste waarde aanneemt.

Let op:

a. In tegenstelling tot de overige tijden worden deze tijden weergegeven met twee cijfers achter de decimale punt. Het cijfer of de cijfers voor de decimale punt geven de uurstand aan, de cijfers achter de decimale punt de minutenstand. In figuur 3.22 geeft het klokelement OCHTEND dus aan dat de bijbehorende klok is ingeschakeld van 7.00 tot 9.30 uur.
b. In tegenstelling tot hetgeen in de CRAPDEF.H kan worden opgegeven, kunnen in deze tabel maximaal 20 klokken met bijbehorende tijdstippen worden opgegeven.
   - 0.00 - 24.00 continu aan
   - 0.00 - 23.59 continu aan
   - 0.00 - 0.00 continu uit
   - 7.00 - 7.00 continu uit
   - 7.00 - 7.01 een minuut aan
   - 7.00 - 9.00 twee uur aan
   - 9.00 - 7.00 tweeentwintig uur aan
   - 7.01 - 7.00 een minuut uit.
c. Hieronder wordt een aantal voorbeelden van mogelijke instellingen en bijbehorende consequenties gegeven.

Per klokelement kan één instelling opgegeven worden. De klokkentabel mag leeg zijn.

Schakelaartabel

In tabel 3.21 is een voorbeeld van de schakelaartabel gegeven. De schakelaartabel bevat per regel de symbolische naam van de schakelaar, gevolgd door de stand ‘AAN’ of ‘UIT’. Per schakelaar kan één instelling opgegeven worden.

```
/*     SCHAKELAARtabel                                           */
struct SCHAKELAAR SCHAKELAAR_tabel[] = {
/*     SCH():                   instelling:                      */
/*     NAAM,                    AAN (of UIT),                    */
       S_MA_04_03,              AAN, /* wel/geen mee-aanvraag 04 van 03  */
       S_MA_07_05,              AAN, /*  "    "    "    "     07 van 05  */
       S_WACHTST_ROOD,          UIT, /* wachtstand groen of rood         */
       STOP};
```

Figuur 3.21: Schakelaartabel

De schakelaartabel mag leeg zijn.

Extra geheel getal parameterstabel

De extra geheel getal parameterstabel bevat per regel de symbolische naam van een extra geheel getal parameter (EGGPARM) gevolgd door de waarde waarop deze extra geheel getal parameter wordt ingesteld. De waarde wordt opgegeven als geheel getal, dat wil zeggen zonder decimale punt (zie ook paragraaf 3.3.1).

In figuur 3.22 is een voorbeeld van de extra geheel getal parameterstabel opgenomen. De extra geheel getal parameterstabel mag leeg zijn. Er mogen maximaal 26.000 extra geheel getal parameters worden gedefinieerd.

```
/*
 * ================================================================
 * Extra geheel getal parameters
 * ================================================================
 */

/*     extra EGGPARM tabel                                       */
struct EGGPARMS  EGGPARMS_tabel[] = {
/*     EGGPARM(),                instelling                      */
       EG_JAAR,                  2003,
       STOP};
```

Figuur 3.22: Extra geheel getal parameterstabel

Aan de instelling van EGGPARM's kan worden gerefereerd of er kan mee worden gerekend in de applicatie (bijvoorbeeld in opties of in de functie ‘overige_voorwaarden()’ in de file CRAPCOD.C).

Extra parameterstabel
De extra parameterstabel bevat per regel de symbolische naam van een extra parameter (EPARM) gevolgd door de waarde waarop deze extra parameter wordt ingesteld. De waarde wordt opgegeven met één cijfer achter de decimale punt (zie ook paragraaf 3.3.1).
In figuur 3.23 is een voorbeeld van de extra parameterstabel opgenomen. De extra parameterstabel mag leeg zijn. Er mogen maximaal 26.000 extra parameters worden gedefinieerd.

```
/*
 * ================================================================
 * Extra parameters
 * ================================================================
 */

/*     extra PARMS tabel                                         */
struct EPARMS  EPARMS_tabel[] = {
/*     EPARM(),                  instelling                      */
       EP_MG,                    0.0,
       STOP};
```

Figuur 3.23: Extra parameterstabel

Aan de instelling van EPARM's kan worden gerefereerd of er kan mee worden gerekend in de applicatie (bijvoorbeeld in opties of in de functie ‘overige_voorwaarden()’ in de file CRAPCOD.C).

3.3.8 Opties

Algemeen
Voor iedere optie uit de basisspecificatie is een tabel aanwezig. Per regel van zo'n optietabel kan gespecificeerd worden onder welke voorwaarden de betreffende optie waar is voor een bepaalde signaalgroep en/of voor een bepaald blok. Dit wordt gedaan door het formuleren van een booleaanse expressie. Dat wil zeggen: de optie is waar juist dan als er aan een bepaalde verzameling voorwaarden (een 'booleaanse

expressie') is voldaan. In zo'n geval wordt in de optietabel dus die booleaanse expressie gespecificeerd. Op elk moment dat dan in de applicatiespecificatie bepaald moet worden of de betreffende optie waar is, wordt de bijbehorende booleaanse expressie geëvalueerd. Wanneer een optie altijd waar moet zijn, kan ‘WAAR’ worden ingevuld.

Het formuleren van een booleaanse expressie in C gebeurt met behulp van een zogenaamde functie. Een functie in C is een onafhankelijke verzameling van declaraties en formeel beschreven acties, waarmee op grond van bepaalde invoerparameters één uitvoerwaarde wordt berekend. Deze uitvoerwaarde wordt de terugkeerwaarde van de functie genoemd (wat de functie heeft 'teruggegeven' op de invoer) en moet van een bepaald type zijn, het zogenaamde terugkeertype.

Een declaratie van een functie voor een tabeloptie ziet er in C als volgt uit:

Bool naam van de functie (void)
{
statementlist; (optioneel)
return booleaanse expressie;
}

Bij tabelopties worden geen invoerparameters meegegeven. In de RWS C-regelaar moeten alle functies die in een optietabel genoemd worden, zijn gedefinieerd en uitgeschreven vóór die optietabel zelf. Hoewel de functienaam vrij te kiezen is (voor zover nog niet gebruikt voor iets anders), wordt aanbevolen om de naam van de optie hierin te gebruiken. In de volgende paragraaf over blokgebonden opties zullen we met behulp van het voorbeeld in figuur 3.24 een en ander toelichten.

Niet gespecificeerde opties zijn, conform de basisspecificatie, per definitie niet waar.

Blokgebonden opties

```
/*
 * ================================================================
 * opties voor het vasthouden van de blokactiviteit
 * ================================================================
 */

/*     BK gebonden optie: BO_OP(BKl)                              */
Bool alg_BO_funct (Int16 n, Int16 i) {
     return ROG(n) && A(n) || RVG(n) || G(n) && TIJD(i) && !MVG(n); }
Bool BO_funct_BK1 (void) {
     if (begin_G(SG02)) herstart_TIJD(T_BO02);
     if (begin_G(SG03)) herstart_TIJD(T_BO03);
     if (begin_G(SG04)) herstart_TIJD(T_BO04);
     if (begin_G(SG05)) herstart_TIJD(T_BO05);
     if (begin_G(SG07)) herstart_TIJD(T_BO07);
     if (begin_G(SG08)) herstart_TIJD(T_BO08);
     if (begin_G(SG22)) herstart_TIJD(T_BO22);
     if (begin_G(SG24)) herstart_TIJD(T_BO24);
     if (begin_G(SG26)) herstart_TIJD(T_BO26);
     if (begin_G(SG28)) herstart_TIJD(T_BO28);
     return alg_BO_funct(SG02, T_BO02) || alg_BO_funct(SG08, T_BO08) ||
            alg_BO_funct(SG24, T_BO24);    }
Bool BO_funct_BK2 (void) {
     return alg_BO_funct(SG03, T_BO03) || alg_BO_funct(SG04, T_BO04) ||
            alg_BO_funct(SG26, T_BO26) || alg_BO_funct(SG28, T_BO28);    }
Bool BO_funct_BK3 (void) {
     return alg_BO_funct(SG05, T_BO05) || alg_BO_funct(SG07, T_BO07) ||
            alg_BO_funct(SG22, T_BO22);    }

struct BK_OPTIE           BO_OP_tabel[] = {
/*     optie voor blok:   inhoud van de optie:                    */
```

```
/*     BKl,              functie,                              */
       BK1,              BO_funct_BK1,
       BK2,              BO_funct_BK2,
       BK3,              BO_funct_BK3,
       STOP};
```

Figuur 3.24: Optietabel voor de BO-optie

De blokophoudoptie (BO-optie), die de blokovergang kan tegenhouden, is de enige optie die alleen blokgebonden is (en niet ook signaalgroepgebonden). In de optietabel moet er een regel worden gedefinieerd voor ieder blok waarvoor deze optie moet gelden. Hierin wordt eerst het bloknummer van het betreffende blok genoemd, en daarna de naam van een vooraf gedeclareerde functie.

In figuur 3.24 zijn BO-opties gespecificeerd en wel voor blok 1, blok 2 en blok 3. Als voorbeeld nemen we blok 1: de waarde van de BO-optie voor dit blok op een bepaald moment is gelijk aan de terugkeerwaarde van de functie BO_funct_BK1 op dat moment, zoals is gespecificeerd door:

```
       BK1,              BO_funct_BK1,
```

Deze functie BO_funct_BK1 is direct vóór de BO_OP_tabel gedefinieerd, door

```
Bool BO_funct_BK1 (void) {
if (begin_G(SG02)) herstart_TIJD(T_BO02);
if (begin_G(SG03)) herstart_TIJD(T_BO03);
if (begin_G(SG04)) herstart_TIJD(T_BO04);
if (begin_G(SG05)) herstart_TIJD(T_BO05);
if (begin_G(SG07)) herstart_TIJD(T_BO07);
if (begin_G(SG08)) herstart_TIJD(T_BO08);
if (begin_G(SG22)) herstart_TIJD(T_BO22);
if (begin_G(SG24)) herstart_TIJD(T_BO24);
if (begin_G(SG26)) herstart_TIJD(T_BO26);
if (begin_G(SG28)) herstart_TIJD(T_BO28);
return  alg_BO_funct(SG02, T_BO02) || alg_BO_funct(SG08, T_BO08) ||
        alg_BO_funct(SG24, T_BO24);    }
```

Voor de functienaam BO_funct_BK1 staat het terugkeertype ‘Bool, dat wil zeggen dat de terugkeerwaarde van BO_funct_BK1 gelijk moet zijn aan een booleaanse waarde; er zijn twee booleaanse waarden, namelijk 0 (niet-waar) en 1 (waar). Er is één invoerparameter ‘void’ die aangeeft dat er geen (invoer)waarden zijn: binnen de accolades { en } staat immers dat de functie de logische uitkomst van ‘alg_BO_funct(SG02, T_BO02) ||’ etc. teruggeeft. De aanroep ‘alg_BO_funct(SG02, T_BO02)’ roept de betreffende functie aan en geeft de logische uitkomst van ‘ROG(SG02) && A(SG02) || RVG(SG02) || G(SG02) && TIJD(T_BO02) && !MVG(SG02)’ terug (de variabelen n en i worden vervangen door de functionele namen uit de functieaanroep). Voorafgaand aan het returnstatement is een statementlist opgenomen waarin de diverse hulptijden voor de blokophoudoptie worden gestart.

Signaalgroepgebonden opties
De volgende opties zijn alle uitsluitend signaalgroepgebonden: de MA-, TMVG-, TR-, VAA-, PPV-, PPB-, PPS-, AFK-, BL-, RWR-, VRVG-, VVAG2e-, RVAG2e- en VMVG-optie. In de optietabel moet er een regel worden gedefinieerd voor iedere signaalgroep waarvoor deze optie moet gelden. Hierin wordt eerst het signaalgroepnummer genoemd, en daarna de constante ‘WAAR’ of de naam van een functie.

Blok- en signaalgroepgebonden opties
Er zijn drie opties, namelijk de PPP-, PPA- en de VNM-optie, die zowel blok- als signaalgroepgebonden zijn. In de optietabel moet er een regel worden gedefinieerd voor iedere combinatie van blok en signaalgroep waarvoor deze optie moet gelden. Hierin wordt eerst het signaalgroepnummer genoemd, dan het bloknummer en daarna de constante ‘WAAR’ of de naam van een functie.

In het voorbeeld in figuur 3.25 op de volgende pagina's zijn de signaalgroepgebonden opties en de blok- en signaalgroepgebonden opties samen opgenomen. De betreffende optietabellen kunnen leeg zijn.

```
/*
 * ============================================================
 * opties voor aanvragen, meeverlengen en tegenhouden
 * ============================================================
 */

/*     SG gebonden optie: MA_OP(SGn)                          */
Bool alg_MA_funct (Int16 n) { return ROG(n) && A(n) || RVG(n); }
Bool MA_funct_02 (void) {
     return MAH7(SG02, 0) || alg_MA_funct(SG08); }
Bool MA_funct_03 (void) { return MAH7(SG03, 0); }
Bool MA_funct_04 (void) {
     return MAH7(SG04, 0) || alg_MA_funct(SG03) && SCH(S_MA_04_03); }
Bool MA_funct_05 (void) { return MAH7(SG05, 0); }
Bool MA_funct_07 (void) {
     return MAH7(SG07, 0) || alg_MA_funct(SG05) && SCH(S_MA_07_05); }
Bool MA_funct_08 (void) { return MAH7(SG08, 0); }
Bool MA_funct_11 (void) { return MAH7(SG11, 0); }
Bool MA_funct_26 (void) { return MAH7(SG26, 0); }
Bool MA_funct_28 (void) { return alg_MA_funct(SG02) || alg_MA_funct(SG08); }

struct SG_OPTIE           MA_OP_tabel[] = {
/*     optie voor sg:     inhoud van de optie:                */
/*     SGn,               functie,                            */
       SG02,              MA_funct_02,
       SG03,              MA_funct_03,
       SG04,              MA_funct_04,
       SG05,              MA_funct_05,
       SG07,              MA_funct_07,
       SG08,              MA_funct_08,
       SG11,              MA_funct_11,
       SG26,              MA_funct_26,
       SG28,              MA_funct_28,
       STOP};

/*     SG en BK gebonden optie:    PPP_OP(SGn, BKl)           */
struct SG_BK_OPTIE                 PPP_OP_tabel[] = {
/*     optie voor sg:  voor blok:  inhoud van de optie:       */
/*     SGn,            BKl,        functie,                   */
       STOP};

/*     SG gebonden optie: TMVG_OP(SGn)                        */
Bool TMVG_funct_alg (void) { return !SCH(S_WACHTST_ROOD) || WSRH; }

struct SG_OPTIE           TMVG_OP_tabel[] = {
/*     optie voor sg:     inhoud van de optie:                */
/*     SGn,               WAAR (of functie),                  */
       SG02,              TMVG_funct_alg,
       SG03,              TMVG_funct_alg,
       SG04,              TMVG_funct_alg,
       SG05,              TMVG_funct_alg,
       SG07,              TMVG_funct_alg,
       SG08,              TMVG_funct_alg,
       SG11,              TMVG_funct_alg,
       SG22,              TMVG_funct_alg,
       SG24,              TMVG_funct_alg,
       SG26,              TMVG_funct_alg,
       SG28,              TMVG_funct_alg,
       STOP};
```

```
/*     SG gebonden optie: TR_OP(SGn)                              */
struct SG_OPTIE           TR_OP_tabel[] = {
/*     optie voor sg:     inhoud van de optie:                    */
/*     SGn,               functie,                                */
       STOP};

/*
 * ============================================================
 * opties voor versnelde realisatie
 * ============================================================
 */

/*     SG gebonden optie: VAA_OP(SGn)                             */
struct SG_OPTIE           VAA_OP_tabel[] = {
/*     optie voor sg:     inhoud van de optie:                    */
/*     SGn,               WAAR (of functie),                      */
       SG02,              WAAR,
       SG03,              WAAR,
       SG04,              WAAR,
       SG05,              WAAR,
       SG07,              WAAR,
       SG08,              WAAR,
       SG11,              WAAR,
       SG22,              WAAR,
       SG24,              WAAR,
       SG26,              WAAR,
       SG28,              WAAR,
       STOP};

/*     SG gebonden optie: PPV_OP(SGn)                             */
struct SG_OPTIE           PPV_OP_tabel[] = {
/*     optie voor sg:     inhoud van de optie:                    */
/*     SGn,               WAAR (of functie),                      */
       SG02,              WAAR,
       SG03,              WAAR,
       SG04,              WAAR,
       SG05,              WAAR,
       SG07,              WAAR,
       SG08,              WAAR,
       SG11,              WAAR,
       SG22,              WAAR,
       SG24,              WAAR,
       SG26,              WAAR,
       SG28,              WAAR,
       STOP};

/*
 * ============================================================
 * optie voor bijzondere realisatie
 * ============================================================
 */

/*     SG gebonden optie: PPB_OP(SGn)                             */
struct SG_OPTIE           PPB_OP_tabel[] = {
/*     optie voor sg:     inhoud van de optie:                    */
/*     SGn,               functie,                                */
       STOP};

/*
 * ============================================================
 * optie voor speciale realisatie
 * ============================================================
 */

/*     SG gebonden optie: PPS_OP(SGn)                             */
struct SG_OPTIE           PPS_OP_tabel[] = {
/*     optie voor sg:     inhoud van de optie:                    */
/*     SGn,               functie,                                */
       STOP};
```

```
/*
 * ================================================================
 * opties voor alternatieve realisatie
 * ================================================================
 */

/*     SG en BK gebonden optie:    PPA_OP(SGn, BKl)             */
Bool PPA_funct_BK1       (void) { return PPAH(SG08); }
Bool PPA_funct_BK2       (void) { return PPAH(SG03) || PPAH(SG04); }
Bool PPA_funct_BK3_04_22 (void) { return PPAH(SG05); }
Bool PPA_funct_BK3_24_26 (void) { return PPAH(SG22); }
Bool PPA_funct_BK3_28    (void) { return PPAH(SG07); }

struct SG_BK_OPTIE                 PPA_OP_tabel[] = {
/*     optie voor sg:  voor blok:  inhoud van de optie:         */
/*     SGn,            BKl,        functie,                     */
       SG07,           BK1,        PPA_funct_BK1,
       SG24,           BK1,        PPA_funct_BK1,
       SG02,           BK2,        PPA_funct_BK2,
       SG26,           BK2,        PPA_funct_BK2,
       SG04,           BK3,        PPA_funct_BK3_04_22,
       SG22,           BK3,        PPA_funct_BK3_04_22,
       SG24,           BK3,        PPA_funct_BK3_24_26,
       SG26,           BK3,        PPA_funct_BK3_24_26,
       SG28,           BK3,        PPA_funct_BK3_28,
       STOP};

/*     SG en BK gebonden optie:    VNM_OP(SGn, BKl)             */
Bool VNM_funct_BK1       (void) { return VNMH(SG08); }
Bool VNM_funct_BK2       (void) { return VNMH(SG03) && VNMH(SG04); }
Bool VNM_funct_BK3_04_22 (void) { return VNMH(SG05); }
Bool VNM_funct_BK3_24_26 (void) { return VNMH(SG22); }
Bool VNM_funct_BK3_28    (void) { return VNMH(SG07); }

struct SG_BK_OPTIE                 VNM_OP_tabel[] = {
/*     optie voor sg:  voor blok:  inhoud van de optie:         */
/*     SGn,            BKl,        functie,                     */
       SG07,           BK1,        VNM_funct_BK1,
       SG24,           BK1,        VNM_funct_BK1,
       SG02,           BK2,        VNM_funct_BK2,
       SG26,           BK2,        VNM_funct_BK2,
       SG04,           BK3,        VNM_funct_BK3_04_22,
       SG22,           BK3,        VNM_funct_BK3_04_22,
       SG24,           BK3,        VNM_funct_BK3_24_26,
       SG26,           BK3,        VNM_funct_BK3_24_26,
       SG28,           BK3,        VNM_funct_BK3_28,
       STOP};
/*
 * ================================================================
 * opties voor beinvloeding van het verloop van de signaalgroep
 * ================================================================
 */

/*     SG gebonden optie: AFK_OP(SGn)                           */
struct SG_OPTIE           AFK_OP_tabel[] = {
/*     optie voor sg:     inhoud van de optie:                  */
/*     SGn,               functie,                              */
       STOP};

/*     SG gebonden optie: BL_OP(SGn)                            */
struct SG_OPTIE           BL_OP_tabel[] = {
/*     optie voor sg:     inhoud van de optie:                  */
/*     SGn,               functie,                              */
       STOP};

/*     SG gebonden optie: RWR_OP(SGn)                           */
struct SG_OPTIE           RWR_OP_tabel[] = {
/*     optie voor sg:     inhoud van de optie:                  */
/*     SGn,               functie,                              */
       STOP};
```

```
/*     SG gebonden optie: VRVG_OP(SGn)                          */
Bool VRVG_funct_02 (void) { return HF(H_RVG28); }
Bool VRVG_funct_08 (void) { return HF(H_RVG02) || HF(H_RVG28); }

struct SG_OPTIE           VRVG_OP_tabel[] = {
/*     optie voor sg:     inhoud van de optie:                  */
/*     SGn,               functie,                              */
       SG02,              VRVG_funct_02,
       SG08,              VRVG_funct_08,
       STOP};

/*     SG gebonden optie: VVAG2e_OP(SGn)                        */
Bool funct_FIX (void)    { return FIX; }

struct SG_OPTIE           VVAG2e_OP_tabel[] = {
/*     optie voor sg:     inhoud van de optie:                  */
/*     SGn,               functie,                              */
       SG02,              funct_FIX,
       SG03,              funct_FIX,
       SG04,              funct_FIX,
       SG05,              funct_FIX,
       SG07,              funct_FIX,
       SG08,              funct_FIX,
       SG11,              funct_FIX,
       SG22,              funct_FIX,
       SG24,              funct_FIX,
       SG26,              funct_FIX,
       SG28,              funct_FIX,
       STOP};

/*     SG gebonden optie: RVAG2e_OP(SGn)                        */
struct SG_OPTIE           RVAG2e_OP_tabel[] = {
/*     optie voor sg:     inhoud van de optie:                  */
/*     SGn,               functie,                              */
       STOP};

/*     SG gebonden optie: VMVG_OP(SGn)                          */
struct SG_OPTIE           VMVG_OP_tabel[] = {
/*     optie voor sg:     inhoud van de optie:                  */
/*     SGn,               functie,                              */
       SG02,              funct_FIX,
       SG03,              funct_FIX,
       SG04,              funct_FIX,
       SG05,              funct_FIX,
       SG07,              funct_FIX,
       SG08,              funct_FIX,
       SG11,              funct_FIX,
       SG22,              funct_FIX,
       SG24,              funct_FIX,
       SG26,              funct_FIX,
       SG28,              funct_FIX,
       STOP};
```

Figuur 3.25: Optietabellen voor overige opties

Koppeltabel

De koppeltabel bevat per regel de symbolische naam van een hulpvariabele die als koppelsignaal vanuit de regelaar naar buiten kan worden gebracht. Deze tabel wordt alleen gebruikt als de regeling aan een andere regeling moet worden gekoppeld. Binnen FLASH en in het verkeersregeltoestel worden de koppelsignalen tussen verschillende regelaars ook afgehandeld.

Eerst wordt in de tabel de HF genoemd; vervolgens wordt aangegeven of deze hulpvariabele een in- of een uitgangssignaal is en als laatste wordt voor de uitgaande koppelsignalen aangegeven of het koppelsignaal door de procesbesturing gereset (KSU) of gehandhaafd (KSH) moet blijven als de regelaar niet regelt. Als

GEEN wordt ingevuld, wordt het betreffende uitgaande koppelsignaal door de procesbesturing gereset als de regelaar niet regelt. De koppeltabel mag leeg zijn. De in- en uitgangssignalen worden vanaf 1 genummerd. Zie ook paragraaf 4.2.6.

Alle koppelsignalen die tussen de aanroepen “START_PTP_KOPPELING, GEEN, GEEN, GEEN,” en “EINDE_PTP_KOPPELING, GEEN, GEEN,GEEN,” staan zijn onderdeel van een PTP-koppeling. Als een regelaar met meerdere regelaars middels PTP gekoppeld is, worden per regelaar de koppelsignalen voorafgegaan door “START_PTP_KOPPELING, GEEN, GEEN, GEEN,” en afgelsoten met “EINDE_PTP_KOPPELING, GEEN, GEEN,GEEN,”.

```
/*
 * ============================================================
 * tabel met koppelsignalen
 * ============================================================
 */

/*     KOPPELtabel                                                          */
struct KOPPEL   KOPPEL_tabel[] = {
/*     hulpfunctie:   ingangssignaal: uitgangssignaal: KSU of KSH:        */
/*     NAAM,          x (of GEEN),    x (of GEEN),     KSU, KSH (of GEEN) */

       KU_MG_OCHTEND,        GEEN,    1,               KSU,  /* Koppeling met VRI 200 */
       KU_MG_AVOND,          GEEN,    2,               KSU,

       #ifndef pc_omgeving
       START_PTP_KOPPELING,  GEEN,  GEEN,             GEEN, /* PTP-koppeling van/naar VRI 300 */
       #endif

       KI_FILE_02_300,          3,  GEEN,             GEEN,

       #ifndef pc_omgeving
       EINDE_PTP_KOPPELING,  GEEN,  GEEN,             GEEN, /* PTP-koppeling van/naar VRI 300 */
       #endif

       STOP};
```

Figuur 3.26: Koppeltabel

Regelingentabel
Deze tabel dient de applicatiebouwer te gebruiken om (gekoppelde) regelingen kenbaar te maken aan het programma, aan FLASH en aan het verkeersregeltoestel. De tabel bevat per regel maar één element, namelijk een kruispuntnummer. Het eerste element is het eigen kruispuntnummer (verplicht), de tweede en volgende elementen bevatten de kruispuntnummers van de gekoppelde regelingen.

In het voorbeeld van figuur 3.27 is 100 het kruispuntnummer van deze regeling en 200 het kruispuntnummer van de regeling waarmee gekoppeld wordt. Het kruispuntnummer mag maximaal de waarde 32767 hebben.

```
/*
 * ============================================================
 * tabel met gekoppelde regelingen
 * ============================================================
 */
/*     REGELINGtabel                                            */
struct REGELING   REGELING_tabel[] = {
/*     Nummer                                                   */
       100, /* Deze regeling */
       200, /* VRI 200 */
       STOP};
```

Figuur 3.27: Regelingentabel

3.4 De overige voorwaarden in CRAPCOD.C

3.4.1 Algemeen

In de file CRAPCOD.C kunnen de eventuele acties worden gespecificeerd die niet in de tabellen (CRAPTAB.C) kunnen worden ondergebracht: de zgn. overige voorwaarden.

3.4.2 Het kopje van CRAPCOD.C

Net als bij de CRAPDEF.H en de CRAPTAB.C is het gebruikelijk de file te laten beginnen met wat commentaar om aan te geven wat de inhoud van de file is en wie er verantwoordelijk is voor die inhoud. Dit commentaar is niet verplicht maar wel handig ter identificatie van de file. Ook kunnen wijzigingen in dit commentaar worden bijgehouden.

```
/*
 * Copyright(c)  2014  Rijkswaterstaat
 *                     WVL
 * File: CRAPCOD.C
 *
 * Nummer kruispunt:        1
 * Regelprogramma vri:      Demoregeling CRW
 *
 * Dienst:                  Goudappel Coffeng BV
 * Ontwerper:               Theo Dijkshoorn
 *
 * datum:                   14-02-2014
 * versie:                  1
 */

#include "crtypes.h"
#include "crapcod.h"
#include "craptab.h"
#include "ccie_int.h"
#include "crapdef.h"

#include "crvlog.h"    /* tbv VLOG             */
#include "aftr_td.inc" /* include files        */
#include "fasebew.inc"
#include "hfrvg.inc"
#include "kiezen_m.inc"

#ifdef pc_omgeving
#include "crfutils.h"  /* tbv printxy in flash */
#endif
```

Figuur 3.28: Het kopje van CRAPCOD.C

De vijf include-regels uit figuur 3.28 die op het commentaar volgen, zijn verplicht. De vijf include-regels die daarop volgen, zijn afhankelijk van de in externe bestanden gespecificeerde functionaliteiten die men in de CRAPCOD.C wil gebruiken. In dit geval worden de standaard functionaliteiten voor VLOG, aftrekken tijden, fasebewaking, hulpvariabele rood-voor-groen en kiezen maximum groentijden gebruikt.
In de drie regels daarna wordt de printxy-functionaliteit opgenomen voor gebruik op de pc (binnen FLASH). Met deze functionaliteit kan extra informatie over de verkeersregeling naar het scherm worden geschreven. Code tussen de regels ‘#ifdef pc_omgeving’ en ‘#endif ‘ wordt niet meegenomen bij de implementatie van de verkeersregeling in het verkeersregeltoestel. De naam ‘pc_omgeving’ is namelijk alleen in FLASH bekend en niet bij de fabrikant.

3.4.3 Overige voorwaarden

De overige voorwaarden in de CRAPCOD.C bestaan uit drie onderdelen die ieder in een functie zijn opgenomen, namelijk:
- initialisatie in de functie ‘init_bij_inschakelen()’;
- overige voorwaarden die één keer per seconde worden doorlopen in de functie ‘voorwaarden _per_seconde()’;
- overige voorwaarden die iedere systeemronde worden doorlopen in de functie ‘overige_voorwaarden()’.

Binnen deze functies kan de gebruiker de acties specificeren, die hij niet kwijt kan in de tabellen en de bijbehorende functies.

3.4.4 Initialisatie

Specifieke acties die de gebruiker tijdens de initialisatie wil laten uitvoeren, moet hij onderbrengen in de functie, ‘init_bij_inschakelen()’. De acties moeten uiteraard worden beschreven volgens de C-syntaxis met behulp van formules en expressies. In die formules en expressies wordt gebruik gemaakt van de eigenschappen van basisspecificatiefuncties (zie hoofdstuk 4 en A1.2 uit de basisspecificatie [BS]), zoals geïmplementeerd in de RWS C-regelaar. Een voorbeeld van de functie ‘init_bij _inschakelen()’ is in figuur 3.29 opgenomen.

```
/*
 * ============================================================
 * overige voorwaarden
 * ============================================================
 */

void init_bij_inschakelen(void)
{

setVlogVRIid("Demo_CR");          /* Kruispuntnaam VLOG-bestanden                              */
setVlogMode(VLOGMODE_FILE_ASCII); /* wijze van genereren VLOG-bestanden                        */
                                  /* mogelijke opties:                                         */
                                  /* VLOGMODE_NONE             = niks doen                     */
                                  /* VLOGMODE_REALTIME_BINAIR  = realtime stream, binair formaat */
                                  /* VLOGMODE_FILE_BINAIR      = naar file, binair formaat     */
                                  /* VLOGMODE_REALTIME_ASCII   = realtime stream, ASCII formaat  */
                                  /* VLOGMODE_FILE_ASCII       = naar file, ASCII formaat      */
                                   /* Let op: voor toepassing VLOG ook #include "crvlog.h" in     */
                                  /* crapcod.c opnemen !!!                                     */

/* ============================
 * kiezen maximum groentijden
 * ============================
 */
 kiezen_MG_TIJDEN (OCHTEND, AVOND, EP_MG, EU);

/* ================================
 * Aftrekken tijden tbv alternatief
 * ================================
 */
 aftrekken_TIJDEN(T_PPA03P,  T_PPA03, SG03);
 aftrekken_TIJDEN(T_PPA04P,  T_PPA04, SG04);
 aftrekken_TIJDEN(T_PPA05P,  T_PPA05, SG05);
 aftrekken_TIJDEN(T_PPA07P,  T_PPA07, SG07);
 aftrekken_TIJDEN(T_PPA08P,  T_PPA08, SG08);
```

```
/* ================================
 * schakelen wachtstand rood/groen
 * ================================
 */
 if (SCH(S_WACHTST_ROOD)) { reset_WSGR_toewijzing(SG02);
                            reset_WSGR_toewijzing(SG08);
                            reset_WSGR_toewijzing(SG28); }
 else                     {   set_WSGR_toewijzing(SG02);
                              set_WSGR_toewijzing(SG08);
                              set_WSGR_toewijzing(SG28); }
} /* einde init_bij_inschakelen */
```

Figuur 3.29: De functie ‘init_bij_inschakelen()’

In dit voorbeeld wordt eerst VLOG ingeschakeld, waarbij de kruispuntnaam ‘Demo_CR’ is en de VLOG-data in ASCII-formaat naar bestanden wordt weggeschreven. Vervolgens worden de actuele maximumgroentijden gekozen, de actuele tijden voor alternatieve realisaties berekend en de actuele wachtstand ingesteld.

3.4.5 Specifieke acties voor een regeling per seconde

In de functie "voorwaarden_per_seconde" kunnen de voorwaarden c.q. instructies worden opgenomen, die niet elke systeemronde van de regelaar behoeven te worden geëvalueerd; bijv. de kiesinstructies. Wel moet er bij de formulering van deze voorwaarden rekening mee worden gehouden, dat deze functie slechts éénmaal per seconde wordt aangeroepen. Zo mogen er bijvoorbeeld geen begin- en eindmeldingen (begin_HF(), einde_G(), etc.) in deze voorwaarden worden opgenomen, omdat niet gegarandeerd is dat deze ook bij de afhandeling van deze functie worden ‘gezien’ door de regelaar. Alleen de begin- en eindemeldingen van detectiefouten worden dusdanig afgehandeld dat ze in deze functie kunnen worden opgenomen. Een voorbeeld van de functie ‘voorwaarden_per_seconde’ is opgenomen in figuur 3.30.

```
void voorwaarden_per_seconde(void)
{
/* ================================
 * Aftrekken tijden tbv alternatief
 * ================================
 */
 aftrekken_TIJDEN(T_PPA03P,  T_PPA03, SG03);
 aftrekken_TIJDEN(T_PPA04P,  T_PPA04, SG04);
 aftrekken_TIJDEN(T_PPA05P,  T_PPA05, SG05);
 aftrekken_TIJDEN(T_PPA07P,  T_PPA07, SG07);
 aftrekken_TIJDEN(T_PPA08P,  T_PPA08, SG08);
}  /* einde voorwaarden per seconde */
```

Figuur 3.30: De functie ‘voorwaarden_per_seconde()’

In dit voorbeeld worden de tijden voor de alternatieve realisatie in de functie ‘voorwaarden_per_seconde()’ bijgewerkt.

3.4.6 Overige specifieke acties voor een regeling

In de functie ‘overige_voorwaarden()’ worden de overige specifieke acties voor een regeling opgenomen. Deze acties worden iedere systeemronde doorlopen. Een voorbeeld van de functie ‘overige_voorwaarden()’ is opgenomen in figuur 3.31.

void overige_voorwaarden(void)

```
{
/* ===================================
 * afhandeling deelconflicten 02-08-28
 * ===================================
 */
 HFRVG(SG02, H_RVG02);
 HFRVG(SG28, H_RVG28);

/* ============================
 * maatregelen bij fasebewaking
 * ============================
 */
 fasebewaking (T_FB02);


/* ================================
 * schakelen wachtstand rood/groen
 * ================================
 */
 if (begin_SCH(S_WACHTST_ROOD)) { reset_WSGR_toewijzing(SG02);
                                  reset_WSGR_toewijzing(SG03);
                                  reset_WSGR_toewijzing(SG04);
                                  reset_WSGR_toewijzing(SG28); }
 if (einde_SCH(S_WACHTST_ROOD)) {   set_WSGR_toewijzing(SG02);
                                    set_WSGR_toewijzing(SG03);
                                    set_WSGR_toewijzing(SG04);
                                    set_WSGR_toewijzing(SG28); }

/* ============================
 * Signalen van/naar interface
 * ============================
 *
 *   1. Ochtendprogramma
 *   2. Dalprogramma
 *   3. Avondprogramma
 */
 if (inst_EPARM(EP_MG)==1)
 {  plaats_in_INT (eerste_plaats_positie+OVU_MG_OCHTEND, 0);   /* OCHTEND  */
    plaats_in_INT (eerste_plaats_positie+OVU_MG_DAL,     1);   /* DAL      */
    plaats_in_INT (eerste_plaats_positie+OVU_MG_AVOND,   0); } /* AVOND    */
 if (inst_EPARM(EP_MG)==2)
 {  plaats_in_INT (eerste_plaats_positie+OVU_MG_OCHTEND, 1;   /* OCHTEND  */
    plaats_in_INT (eerste_plaats_positie+OVU_MG_DAL,     0;   /* DAL      */
    plaats_in_INT (eerste_plaats_positie+OVU_MG_AVOND,   0); } /* AVOND    */
 if (inst_EPARM(EP_MG)==3)
 {  plaats_in_INT (eerste_plaats_positie+OVU_MG_OCHTEND, 0);   /* OCHTEND  */
    plaats_in_INT (eerste_plaats_positie+OVU_MG_DAL,     0;   /* DAL      */
    plaats_in_INT (eerste_plaats_positie+OVU_MG_AVOND,   1; } /* AVOND    */

/* ================================
 * printstatements e.d. tbv testen
 * ================================
 */
#ifdef pc_omgeving
if (inst_EPARM(EP_MG)==1)  printxy (0, 0, "DALPERIODE,   MG_1");
if (inst_EPARM(EP_MG)==2)  printxy (0, 0, "OCHTENDSPITS, MG_2");
if (inst_EPARM(EP_MG)==3)  printxy (0, 0, "AVONDSPITS,   MG_3");
#endif

/* ==========================
 * kiezen maximum groentijden
 * ==========================
 */
 kiezen_MG_TIJDEN (OCHTEND, AVOND, EP_MG, SU);

} /* einde overige voorwaarden */
```

Figuur 3.31: De functie ‘overige_voorwaarden()’

In dit voorbeeld zijn de volgende acties opgenomen:
- afhandeling deelconflicten
- fasebewaking
- schakelen wachtstand
- aansturen signalen voor de handbediening
- aansturen signalen binnen de testomgeving.

Als voorbeeld nemen we het stukje code voor het schakelen van de wachtstand binnen ‘overige_voorwaarden’: daar wordt gespecificeerd, dat als de schakelaar ‘SCH(S_WACHTST_ROOD)’ waar wordt, de wachtstand groen toewijzingen van de richtingen 2, 3 en 4 worden ingetrokken en ontstaat een wachtstand rood regeling. Bij het afvallen van de schakelaar worden de wachtstand groen toewijzigingen weer toegekend.

In de formules en expressies die worden gebruikt om de acties te beschrijven, kan gebruik worden gemaakt van de eigenschappen van basisspecificatiefuncties.
Voorbeelden van die eigenschappen zijn:
- de waarde van een teller TL behorend bij signaalgroep n kan met 1 verhoogd of verlaagd worden door de instructie ‘incr_TL(naam)’ respectievelijk ‘decr_TL(naam)’
- er kan gerefereerd worden aan een datum: bijvoorbeeld aan 2 februari 2001 door de referentie ‘datum(2001,2,22)’. Als 2 februari 2001 inderdaad de datum in de RWS C-regelaar is, geeft de functie datum de waarde ‘WAAR’ terug, en in het andere geval de waarde ‘NIET-WAAR’.

Bijlage 2 geeft een limitatieve opsomming van alle mogelijke referenties en instructies van de RWS C-regelaar.

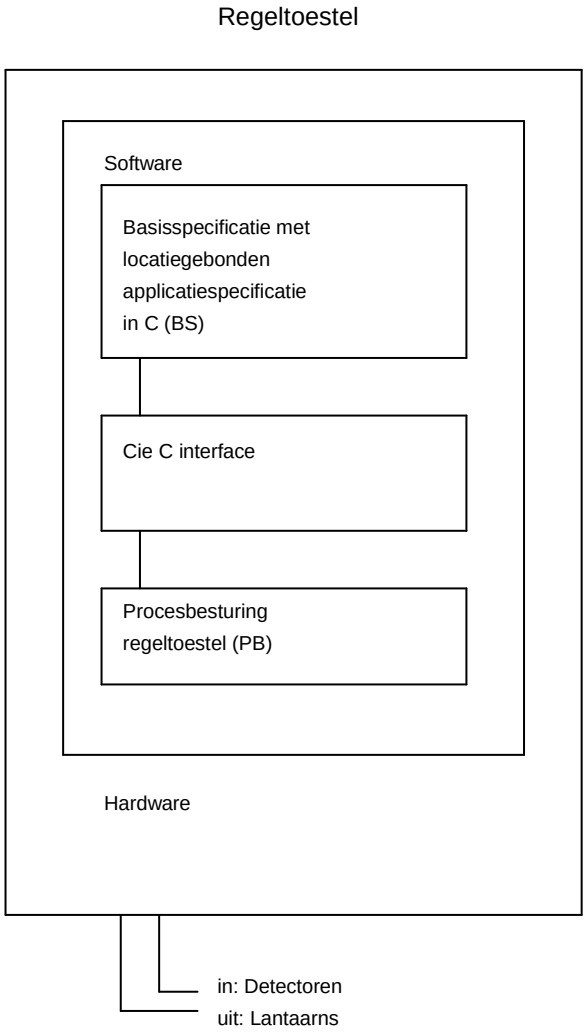
De legenda van de basisspecificatie is gegeven in hoofdstuk 2 van [BS]: daar is de betekenis van de symbolen met hun eventuele indices gegeven.

Voor de betekenis van sleutelwoorden in C, zoals ‘if’, wordt verwezen naar de handleiding van de compiler. Met nadruk wordt hier vermeld dat iedere conditionele expressie in C tussen haakjes moet staan. Dus iedere expressie bij een if-, while-, en for-constructie dient expliciet tussen haakjes gezet te worden. Conditionele expressies, die niet tussen haakjes staan, worden soms door de C-compiler als syntactisch correct beschouwd, terwijl de uitkomst onvoorspelbaar blijkt te zijn. Een uitdrukking die niet correct wordt geïnterpreteerd, is bijvoorbeeld ‘if !G(SG02)’; dit moet worden geformuleerd als ‘if (!G(SG02))’.

4 Bijzondere aspecten bij het specificeren

4.1 Opbouw van de software, volgordes

4.1.1 Applicatiespecificaties in het regeltoestel en in de pc

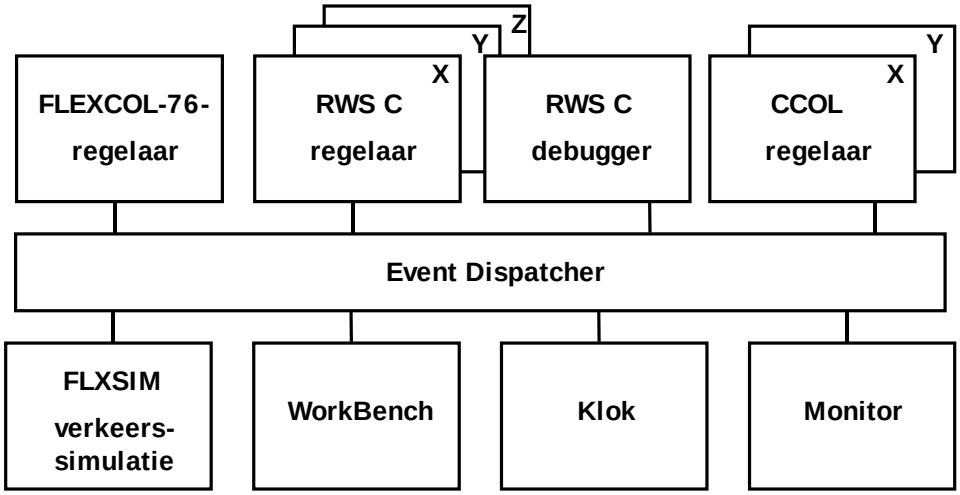


Figuur 4.1: Applicatiespecificatie in C geïmplementeerd in het regeltoestel

De implementatie van een applicatiespecificatie in een verkeersregeltoestel is schematisch weergegeven in figuur 4.1. Tussen de applicatiespecificatie en de basisspecificatie in C en de besturingssoftware van het regeltoestel zorgt de interface van de CVN commissie C voor de communicatie (zie ook paragraaf 1.3).

Als de applicatiespecificatie in C met behulp van de RWS C-regelaar wordt getest op de pc, moet de ProcesBesturing van het regeltoestel (PB) uiteraard worden gesimuleerd (voorzover relevant). In de FLASH-

handleiding (paragraaf 6.1) is de samenhang aangegeven tussen de regelaar en het simulatieproces van FLEXSYT. In figuur 4.2 is de positie van de RWS C-regelaar binnen FLASH weergegeven.



Figuur 4.2: Positie RWS C-regelaar binnen FLASH

Het totale simulatieproces wordt bestuurd vanuit FLASH. De RWS C-regelaar wordt 10 keer per (simulatie)seconde door FLASH aangeroepen, waarbij de benodigde formules worden geëvalueerd. Initialisatie vindt ook plaats in opdracht van FLASH.

Een systeemronde is in dit kader de tijd tussen twee opeenvolgende momenten waarop FLASH de controle over de pc overneemt van de procesbesturing van de C-regelaar.

4.1.2 Volgorde van uitvoering van formules

Algemeen
In de figuur 4.3 is de volgorde gegeven waarin de formules uit de basisspecificatie in het deel ‘Basisspecificatie met locatiegebonden applicatiespecificatie in C’ worden uitgevoerd. Vervolgens is in figuur 4.4 een en ander nog eens schematisch weergegeven.

Bij het inschakelen van de regelaar
- ontstaat ‘begin_R()’ en ‘begin_WR()’ voor alle signaalgroepen en als gevolg daarvan lopen alle ontruimingstijden af
- wordt het eerste blok actief en
- wordt de functie init_bij_inschakelen() aangeroepen.

Int16 applicatieprogramma (Int16)
```
{
...  /* hier niet weergegeven kode */
    /* haal eventueel nieuwe parameters op */
    /* haal nieuwe detectorstanden op */
    /* Werk de tijden bij */

    /* handel RGA af */
    standaard_applicatie_RGA();
```

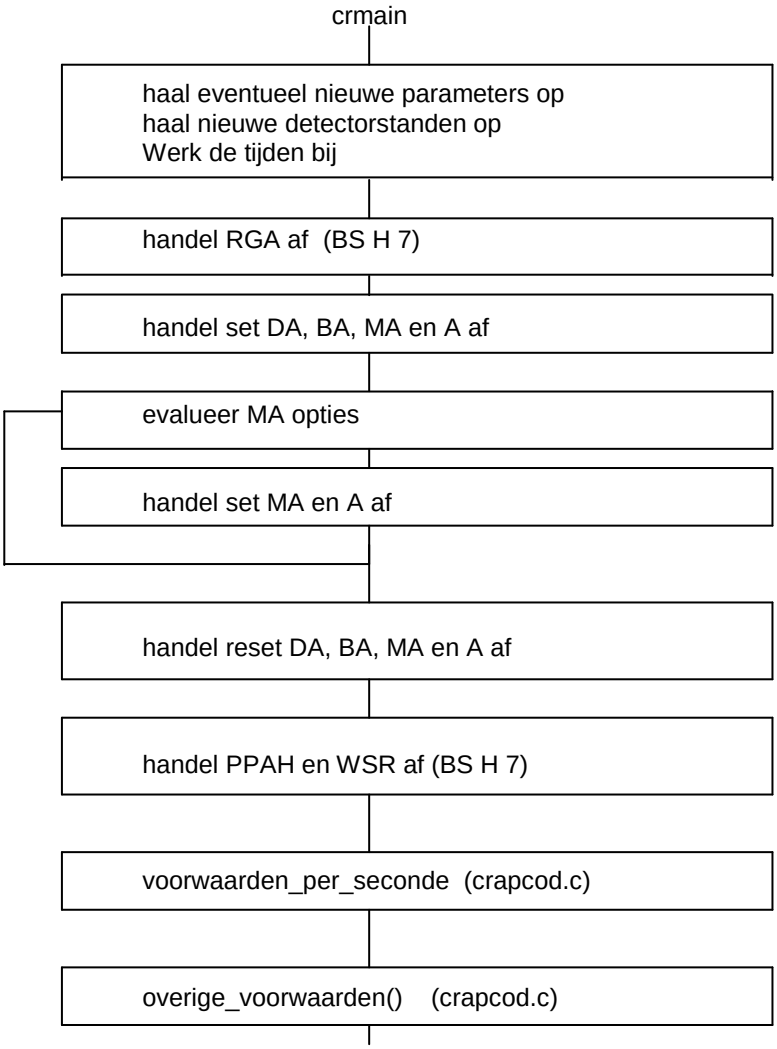
```
    /* handel set BA, DA, MA en A af */
    afhandeling_set_BA();
    afhandeling_set_DA();
    afhandeling_set_MA_A();

    /* evalueer MA optie */
    do {
        /* voer de MA_optie zo lang
         * uit tot er geen aanvragen meer gegenereerd
         * worden tijdens ROG van een willekeurige sg
         */
            Ops_MA_eval();
        } while ( afhandeling_set_MA_A() == TRUE );

    /* handel reset DA, BA, MA en A af */
    reset_afhandeling_DA_BA_MA_A();

    /* handel PPAH en WSR af */
         standaard_applicatie_PPAH_WSR();

    /*
     * evalueer door de gebruiker gedefinieerde
     * overige voorwaarden (crapcod.c )
     */
    if (seconde_nr != CIF_KLOK[CIF_SECONDE])
       voorwaarden_per_seconde();
    regeling_update();
    overige_voorwaarden();

    /*
     * Evalueer opties voor de nieuwe ronde
     */
    Ops_eval();
    Opb_eval();

    /* werk een aantal gelijknamige hulpvariabelen bij */
    diverse_opties_H5_4();

    /* signaalgroep procedure */
    for (sg=0; sg<sgn_max; sg++)
    signaalgroep_procedure(sg);


    /* blok procedure */
    blok_procedure();

    /* berichtenbuffer, bijwerken objecten */
... /* hier niet weergegeven kode */
}
```

Hiernavolgend is de blokprocedure nader uitgewerkt.

```
void blok_procedure()
{
    /*
     * set en reset PPP en PPA
     */
    privilege_periode();

     /*
     * Realisatie primaire SG'n van het actieve blok (BS H 6.3.1)
     */
    real_prm_act_blok ();

    /*
     * reset PR en BR
     */
    reset_alle_prs();
```

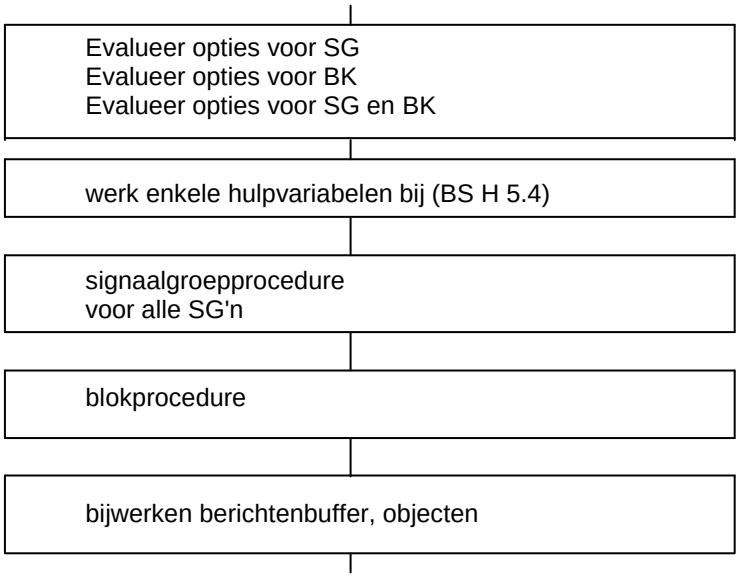
```
/*
 * Overname van primair naar primair (BS H 6.3.4).
 */
ovnm_prm_l_prm_m();
ovnm_prm_m_prm_m();

/*
 * Realisatie alternatieve SG'n van het actieve blok
 * (BS H 6.5.1)
 */
 real_alt_act_blok();

/*
 * reset AR
 */
reset_alle_ars();

/*
 * Overname van alternatief naar primair en van alternatief
 * naar alternatief (BS H 6.5.4 en 6.5.5).
 */
ovnm_alt_l_prm_m();
ovnm_alt_l_alt_m();

/*
 * Realisatie primaire SG'n van een niet actief blok
 * (BS H 6.4.1)
 */
real_prm_niet_act_blok();

/*
 * Overname van alternatief naar primair in blok m+b
 * (BS H 6.5.6).
 */
ovnm_alt_l_prm_mb();

/*
 * versneld naar MVG
 * (BS H 6.5.7)
 */
versneld_MVG_WR();

/*
 * de BR
 * Bijzondere realisatie buiten het blokkenschema
 * (BS H 6.7.1)
 */
bijzondere_realisatie();

/*
 * Overname van bijzonder naar primair (BS H 6.7.2).
 */
ovnm_br_prm_m();

/*
 * Overname van bijzonder naar primair in blok
 * m+b (BS H 6.7.3).
 */
ovnm_br_l_prm_mb();

   /*
    * tot slot de SR
    * Speciale realisatie
    * (BS H 6.8.1)
    */
   speciale_realisatie();

   /*
    * Overname van speciaal naar primair (BS H 6.8.2).
    */
   ovnm_sr_prm_m();
```

```
        /*
         * Overname van speciaal naar primair in blok
         * m+b (BS H 6.8.2).
         */
        ovnm_sr_l_prm_mb();

    /*
     * evalueer mogelijkheden tot blokwisseling
     * ( bs 6.2 )
     */
    blok_wisseling();
}
```

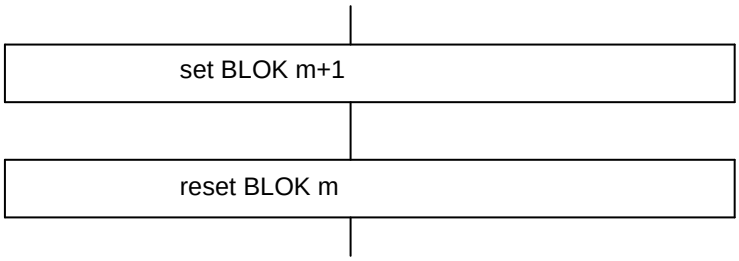


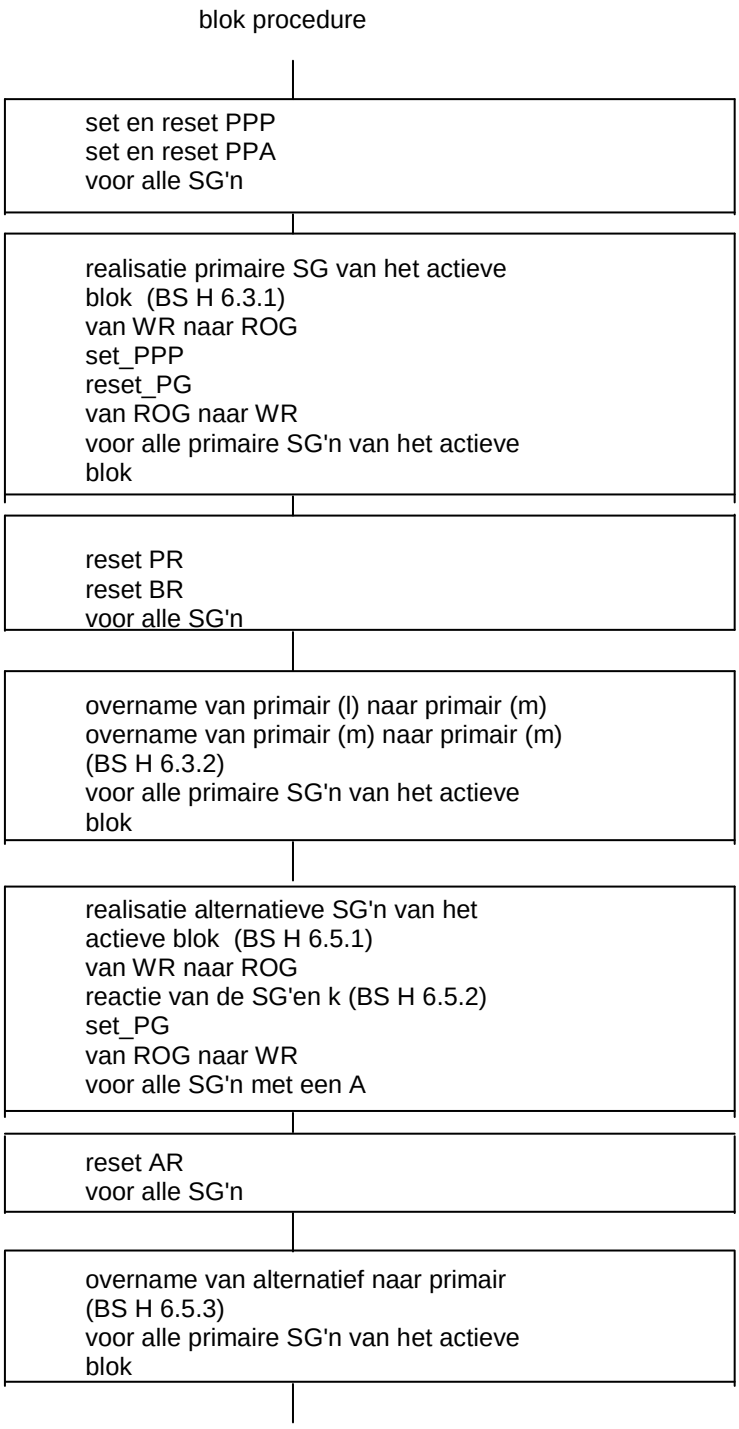
Na ‘handel set MA en A af" kan teruggesprongen worden naar ‘evalueer MA opties’. Dit gebeurt zolang tijdens ‘handel set MA en A af’ een A wordt gegenereerd voor een signaalgroep die op dat moment in ROG staat.

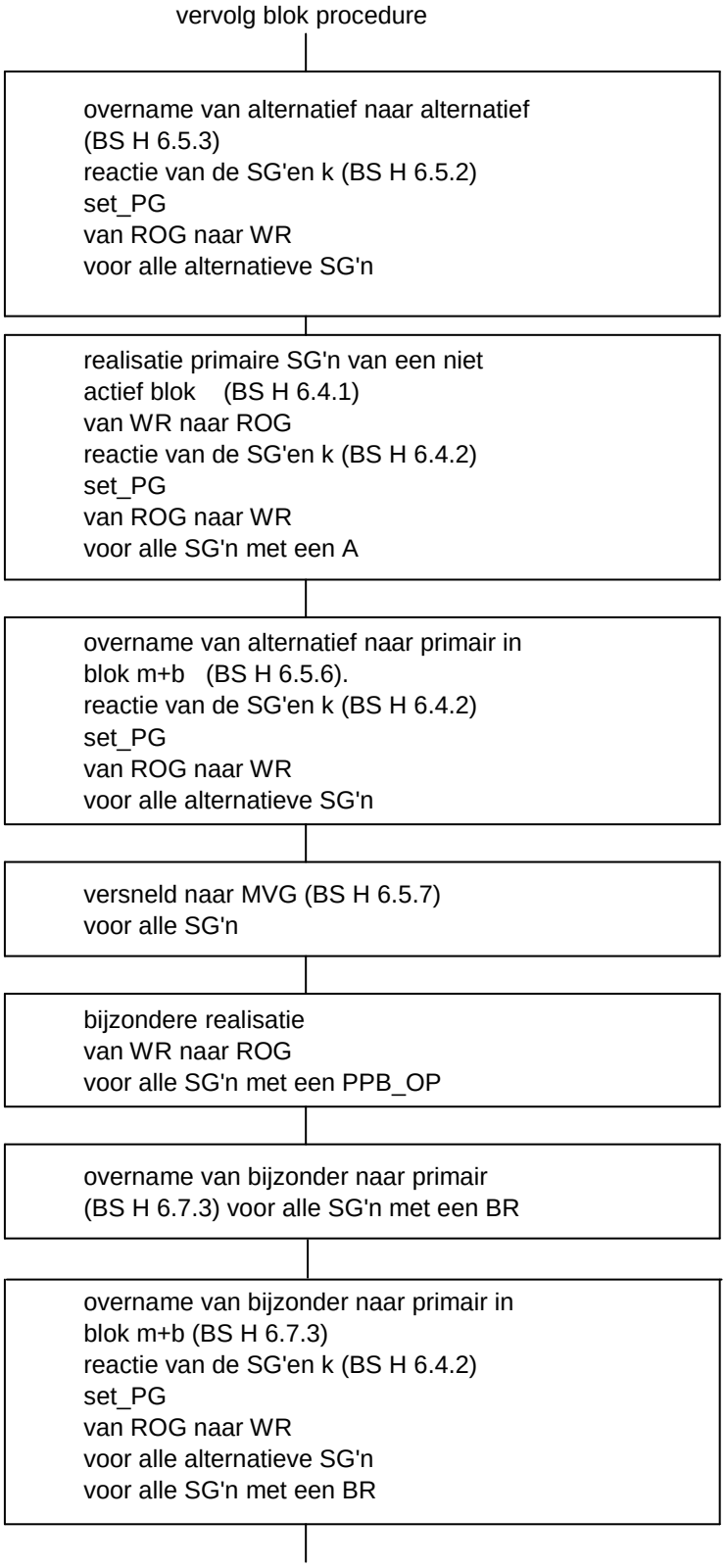
vervolg crmain



blokwisseling




blok procedure
set en reset PPP
set en reset PPA
voor alle SG'n
realisatie primaire SG van het actieve
blok (BS H 6.3.1)
van WR naar ROG
set_PPP
reset_PG
van ROG naar WR
voor alle primaire SG'n van het actieve
blok
reset PR
reset BR
voor alle SG'n
overname van primair (l) naar primair (m)
overname van primair (m) naar primair (m)
(BS H 6.3.2)
voor alle primaire SG'n van het actieve
blok
realisatie alternatieve SG'n van het
actieve blok (BS H 6.5.1)
van WR naar ROG
reactie van de SG'en k (BS H 6.5.2)
set_PG
van ROG naar WR
voor alle SG'n met een A
reset AR
voor alle SG'n
overname van alternatief naar primair
(BS H 6.5.3)
voor alle primaire SG'n van het actieve
blok


vervolg blok procedure
overname van alternatief naar alternatief
(BS H 6.5.3)
reactie van de SG'en k (BS H 6.5.2)
set_PG
van ROG naar WR
voor alle alternatieve SG'n
realisatie primaire SG'n van een niet
actief blok (BS H 6.4.1)
van WR naar ROG
reactie van de SG'en k (BS H 6.4.2)
set_PG
van ROG naar WR
voor alle SG'n met een A
overname van alternatief naar primair in
blok m+b (BS H 6.5.6).
reactie van de SG'en k (BS H 6.4.2)
set_PG
van ROG naar WR
voor alle alternatieve SG'n
versneld naar MVG (BS H 6.5.7)
voor alle SG'n
bijzondere realisatie
van WR naar ROG
voor alle SG'n met een PPB_OP
overname van bijzonder naar primair
(BS H 6.7.3) voor alle SG'n met een BR
overname van bijzonder naar primair in
blok m+b (BS H 6.7.3)
reactie van de SG'en k (BS H 6.4.2)
set_PG
van ROG naar WR
voor alle alternatieve SG'n
voor alle SG'n met een BR

Figuur 4.3: Volgorde van uitvoering van formules

Begin- en eindmeldingen
Begin- en eindmeldingen, zoals bijvoorbeeld ‘begin_HF()’ en ‘einde_G()’, blijven vanaf het moment van ontstaan in de regelaar aanwezig tot aan het einde van de volgende systeemronde (bijwerken objecten). Hierdoor kan er in de gehele regelaar altijd aan de desbetreffende melding worden gerefereerd. Dit betekent wel, dat een dergelijke melding vanaf haar ontstaan tot aan het einde van de systeemronde twee ronden achter elkaar waar is.

In het voorbeeld in figuur 4.4 is de melding ‘begin_HF(H_RVG02)’ waar vanaf het uitvoeren van de instructie ‘set_HF(H_RVG02)’ in ronde x, tot aan het bijwerken van de objecten aan het einde van ronde x+1.

	ronde x	ronde x+1	ronde x+2
overige_voorwaarden	set_HF(H_RVG02) HF(H_RVG02) begin_HF(H_RVG02)	HF(H_RVG02) begin_HF(H_RVG02)	HF(H_RVG02)
bijwerken objecten			

Figuur 4.4: Voorbeeld van de geldigheidsduur van begin- en eindmeldingen

Het hierbovengestelde geldt niet voor de begin- en eindmeldingen van detectoren.
Dit betekent het volgende:
- begin_D(SGn_i) en
- einde_D(SGn_i)

zijn in de regelaar na opkomen resp. afvallen van de detector D(SGn_i) in één systeemronde waar.

Voorbeeld:

Als gevolg van het bovenstaande kan er o.a. op begin_D(SGn_i) worden geteld.
De volgende combinatie van instructies levert na 5 detectiemeldingen van D(SGn_i) de set van de HF(naam) op.

```
if (TL(naam) >= 5 )      maak_TL(naam, 0)
if (begin_D(SGn_i) )     incr_TL(naam)
if (TL(naam)==5 )  set_HF(naam)
```

Gebeurtenissen-systeemronde-matrix
Een vraag met betrekking tot de volgorde van afhandeling van formules kan worden beantwoord door een gebeurtenissen-systeemronde-matrix op te zetten, waarin de relevante gebeurtenissen worden opgenomen in relatie tot de systeemronden.

Op deze wijze kan bijvoorbeeld de vraag worden beantwoord, of de toestand ROG van een signaalgroep bij de behandeling van de MA_OP altijd wordt ‘gezien’.
De volgorde waarin de desbetreffende relevante gebeurtenissen in de regelaar worden geëvalueerd is:

BA
DA
MA
A
primaire realisatie
beëindiging ROG
versnelde realisatie
beëindiging ROG van de conflictrichtingen

Uitgezet in een matrix met systeemronden levert dit het volgende beeld op van een mogelijke situatie:

	ronde x	ronde x+1
BA		BA(SGn)
DA		DA(SGn)
MA		
A		A(SGn)
pr. real. SGk reset ROG SGK	WR(SGk)	ROG(SGk) && !A(SGk)
versn. r. SGn	WR(SGn) && !A (SGn)	ROG(SGn) && A(SGn)
reset ROG SGk		reset_ROG(SGk)

Figuur 4.5: Voorbeeld van gebeurtenissen-systeemronde-matrix

In dit voorbeeld is n een conflictrichting van k, zoals bedoeld bij de versnelde realisatie (zie [BS], Hoofdstuk 6.4), en wordt richting k primair gerealiseerd.

Uit het bovenstaande blijkt dat er geen garantie is, dat de toestand ROG van een signaalgroep bij de behandeling van de MA_OP altijd wordt gezien. Dit is een gevolg van het gegeven dat de versnelde realisatie wordt behandeld na de primaire realisatie. Als een richting in een systeemronde naar ROG is gegaan dan kan dat ROG in dezelfde systeemronde door een versnelde realisatie van een conflictrichting worden beëindigd.

Een ander voorbeeld is het refereren aan een tijd in de functie ‘overige_voorwaarden’ in relatie tot de signaalgroeptoestand: stel dat in de functie overige_voorwaarden de volgende voorwaarden opgenomen worden:

```
if ( RVG(SG22) || G(SG22) && TIJD(T_NLP21)
      set_HF(H_NALOOP21);
else
      reset_HF(H_NALOOP21);  /* en */
if ( begin_G(SG22) )
      start_TIJD(T_NLP21);
```

Deze gebeurtenissen worden in de regelaar geëvalueerd in de volgorde waarin ze zijn gespecificeerd.

Uitgezet in een matrix met systeemronden levert dit het beeld uit figuur 4.6 op.

Doordat het starten van de tijd gespecificeerd is na de HF, is de HF niet waar in de systeemronde waarin "begin_G" waar is (ronde x+2) en de tijd wordt gestart. Immers bij de evaluatie van de HF in ronde x+2 is de tijd nog niet gestart (dat gebeurt direct daarna) en is "G(SG22) && TIJD(T_…..)" niet waar.

Over het algemeen zal het bovenstaande niet de bedoeling zijn en kan dit probleem op eenvoudige wijze worden opgelost door het starten van de tijd te specificeren vóór de HF.

	ronde x	ronde x+1	ronde x+2	ronde x+3
overige_ww.	HF(H_NALOOP21) !TIJD(T_NLP21) TIJD(T_NALOOP21)	HF(H_NALOOP21) !TIJD(T_NLP21)	!HF(H_NALOOP21) TIJD(T_NLP21)	HF(H_NALOOP21)
SG proc.	RVG(SG22)	G(SG22) begin_G(SG22)	G(SG22) begin_G(SG22)	G(SG22)

Figuur 4.6: Voorbeeld van gebeurtenissen-systeemronde-matrix

4.2 Andere aspecten

4.2.1 Systeemgrenzen van de RWS C-regelaar

De volgende systeemgrenzen zijn in één RWS C-regelaar van kracht.

aantal	maximaal
blokken	10
signaalgroepen	48
detectoren (D)	256
hulpvariabelen (HF)	256
tellers (TL)	256
niet signaalgroep gebonden tijden (TIJD)	256
extra geheel getal parameters (EGGPARM)	26.000
extra parameters (EPARM)	26.000
schakelaars (SCH)	256
klokken (KL)	20
in de klokkentabel op te geven klokken (met bijbehorende tijdstippen)	20

aantal	maximaal
inkomende koppelsignalen	128
uitgaande koppelsignalen	128
posities voor extra inkomende informatie in de interface	127
posities voor extra uitgaande informatie in de interface	127

4.2.2 Formules uit de basisspecificatie

Het is mogelijk de formules uit de basisspecificatie [BS] in de applicatiespecificatie te gebruiken. Er moeten dan include-files extra worden opgenomen:
#include ‘crsg.h’ (voor de formules uit de signaalgroepprocedure) en #include ‘crbk.h’ (voor de formules uit te blokprocedure).

Vervolgens kan aan de formules van de basisspecificatie worden gerefereerd door ze als volgt te omschrijven:
- voor de overgangsformules uit hoofdstuk 5.2 wordt ov_ toegevoegd
  bijvoorbeeld: formule b wordt hier ov_b
- voor de diverse formules uit de hoofdstuk 5.4 wordt dv_ toegevoegd
  bijvoorbeeld: formule aq wordt hier dv_aq
- voor de formules van de blokprocedure uit hoofdstuk 6 wordt blk_ toegevoegd
  bijvoorbeeld: formule f wordt hier blk_f.

Tenslotte dient de RWS C-regelaar met betrekking tot deze formules te weten welke signaalgroep het betreft. Dit wordt bereikt door het statement
- sgn_n=SG??

voor het gebruik van de formules te plaatsen. Hierin staat ?? voor het nummer van de signaalgroep.

Voorbeelden hiervan zijn:
- if (BLOK(BK2) && blk_g)
  { statement(s); }

Het bovenstaande statement wordt uitgevoerd als voorwaarde g uit hoofdstuk 6 van de basisspecificatie waar is voor het actieve blok 2;

- sgn_n=SG11;
  if (ov_e)
  { statement(s); }

Het bovenstaande statement wordt uitgevoerd als voorwaarde e uit hoofdstuk 5.2 van de basisspecificatie waar is voor SG11.

De uitdrukking
- sgn_n=SG02;
  if (blk_r)
  { statement(s); }

Waarin de formule r uit hoofdstuk 6.3.1 voorkomt (r=ROG(SGn)) is dus equivalent aan
- if (ROG(SG02))
  { statement(s); }

4.2.3 Datatypen

Binnen een bepaald programma kunnen eigen namen worden gekozen voor de te gebruiken types. Dit is ook bij de RWS C-regelaar het geval.
De voor de gebruiker belangrijkste typen die in de RWS C-regelaar gebruikt worden zijn:
- Int16 signed short
- Uint16 unsigned short
- Uint32 unsigned long
- Bool unsigned short.

Bij het specificeren is het soms noodzakelijk te weten van welk type een bepaalde variabele van de regelaar is. Dit geldt te meer daar binnen de regelaar eigen type-definities zijn doorgevoerd. Hiernavolgend volgen daarom de types van enkele relevante variabelen.

de variabele:	is van het type:
inst_TIJD(naam)	Uint32
waarde_TIJD(naam)	Uint32
inst_VG_TIJD(SGn)	Uint32
waarde_VG_TIJD(SGn)	Uint32
TL(naam)	Uint16
inst_EGGPARM(naam)	Int32
inst_EPARM(naam)	Int32
BKl (symbolisch BK nummer)	Int16
SGn (symbolisch SG nummer)	Int16
SGn_i (symbolisch HF nummer)	Int16

(VG_TIJD geldt hierboven als voorbeeld)

Een Bool wordt gebruikt als terugkeertype bij een functie die een boolean terug moet geven, zoals bijvoorbeeld bij een tabeloptie.

Wanneer er in het applicatieprogramma elementen van een verschillend type met elkaar vergeleken worden of gebruikt worden als operanten in een formule (bijvoorbeeld een EPARM en een TIJD) moet type-casting worden gebruikt. Dit gaat als volgt:

- Vóór de variabele moet tussen haakjes het gewenste type geplaatst worden, bijvoorbeeld als waarde van het type Uint32 is, zal het volgende C-statement een warning van de compiler opleveren:

```
waarde = inst_EPARM (SGn_i);
```

- Door de inst_EPARM te 'casten' naar het type van waarde (Uint32) vervalt deze warning, het 'casten' gebeurt als volgt:

```
waarde = (Uint32) inst_EPARM (SGn_i)
```

- Het casten naar een ander type variabele kan tot gevolg hebben voor de waarde van de variabele. Stel de waarde van i = -10. De instructie (Uint32): kan dan foute resultaten geven, omdat Uint32 geen negatieve getallen aan kan!

4.2.4 Tijden en extra parameters

In de CRAPTAB.C file worden instellingen van tijden en extra parameters opgegeven in de vorm van floating point getallen.
Bij het initialiseren van de regeling worden deze getallen eerst met tien vermenigvuldigd. Tijden worden daarna omgezet naar niet negatieve gehele getallen volgens het type Uint32, extra parameters naar gehele getallen van het type Int32.
Deze constructie is gekozen omdat:

- het werken met longs in de regelaar veel minder tijd kost dan het werken met floating point getallen
- tijden in tienden van seconden in de CRAPTAB.C file moeten kunnen worden opgegeven en
- voor een wijze van invullen van de CRAPTAB.C file is gekozen die zo dicht mogelijk ligt bij de wijze van invullen van een klassieke applicatiespecificatie.

Het bovenstaande betekent dat:

- na het 'opvragen' van een tijds- of extra parameterinstelling in het programma (bijvoorbeeld in de CRAPCOD.C) met behulp van bijvoorbeeld de referentie inst_TIJD() of inst_EPARM() of
- na het opvragen van een tijdsinstelling in de debugger
  door de regelaar een getal wordt afgegeven waarin het tienvoud staat van de in de CRAPTAB.C ingestelde waarde van die tijd of extra parameter.

Het bovenstaande geldt op overeenkomstige wijze voor de referentie waarde_TIJD(). Na het in het programma 'opvragen' van de waarde van een lopende tijd (bijvoorbeeld in de CRAPCOD.C) met behulp van de referentie waarde_TIJD() wordt door de regelaar een getal afgegeven waarin staat hoever de tijd op het moment van 'opvragen' is afgelopen. De inhoud van dit getal is de 'werkelijke' verstreken tijd vermenigvuldigd met tien.

Voor een aantal tijden is de mogelijkheid gecreëerd om ze programmatisch te kunnen wijzigen met bijvoorbeeld de instructie wijzig_TIJD(). Voor de extra parameters kan dit met bijvoorbeeld de instructie wijzig_EPARM().
Bij de wijzigingsinstructie voor tijden kan gebruik worden gemaakt van de ingestelde waarde van andere tijden, voor de extra parameters met de ingestelde waarde van andere extra parameters. Met deze ingestelde waarden kan volgens de regels van de taal C worden gemanipuleerd. Zolang daarbij met symbolische namen in de statements wordt gewerkt, zal weinig hinder worden ondervonden van het feit dat deze waarden met tien zijn vermenigvuldigd. Alleen bij het werken met absolute getallen in de statements moet uiteraard wel met dit feit rekening worden gehouden.

Het werken met absolute getallen is echter vanuit een ander standpunt bezien niet aan te bevelen. Telkens wanneer een absoluut getal in het programma wordt opgenomen kan dit later tijdens de operationele situatie op straat niet meer worden gewijzigd (het is geen parameter van de regeling). Wordt echter gewerkt met tijden of extra parameters die in de CRAPTAB.C file zijn opgegeven, dan worden deze tijden automatisch ook als parameter in de Cie C-interface naar de PB opgenomen en kunnen dan in een operationele situatie met behulp van het instelapparaat van de fabrikant van een andere instelling worden voorzien.
Tevens wordt hier nog gewezen op het volgende.
Wanneer bij het hierboven bedoelde manipuleren tijdsinstellingen van elkaar worden afgetrokken moet rekening worden gehouden met het feit dat deze instellingen van het type Uint32 zijn. Als
inst_TIJD(T_PPA03) == 300 en
inst_TIJD(T_PPA03P) == 400, dan levert
inst_TIJD(T_PPA03) - inst_TIJD(T_PPA03P) een waarde op van 4294967196.
(voor de verklaring daarvan wordt verwezen naar een leerboek over C).

Verder is nog van belang dat de PB gewijzigde tijden van de regelaar uit de Cie C-interface ophaalt, ten behoeve van de communicatie met de buitenwereld. Dit wordt op twee manieren gedaan. Als er één tijd is gewijzigd, dan wordt die ene tijd opgehaald. Is er meer dan één tijd gewijzigd, dan worden alle tijden (dus ook de niet gewijzigde) uit de Cie C-interface opgehaald.
Na bijvoorbeeld de instructie kies_MG_TIJD1() worden meerdere tijden in de regelaar gewijzigd (de maximumgroentijden van alle signaalgroepen) en worden als gevolg daarvan alle tijden door de PB uit de Cie C-interface opgehaald.
Om nu niet teveel tijd kwijt te zijn aan het voortdurend ophalen van tijden uit de Cie C-interface, is het verstandig dergelijke wijzigingen niet cyclisch te laten plaatsvinden, maar alleen bij veranderingen van toestanden.

Voorbeeld van een instelling en een waarde van een tijd:
TIJD(T_BO02) heeft in de CRAPTAB.C een instelling van 5.0s.
Dit heeft in de regelaar de volgende effecten:

referentie:	regelaar geeft:	onder de omstandigheid:
inst_TIJD(T_BO02)	50	
waarde_TIJD(T_BO02)	0	als de tijd niet loopt
waarde_TIJD(T_BO02)	0	bij de start van de tijd
waarde_TIJD(T_BO02)	0	bij de herstart van de tijd
waarde_TIJD(T_BO02)	40	4 seconden na de (her)start van de tijd

4.2.5 Maximumgroen- en hiaattijden

Hiernavolgend volgt een overzicht van de benamingen die in de regelaar en de debugger gehanteerd worden voor de instellingen van de maximumgroen- en hiaatijden.

Benamingen ten behoeve van de maximumgroentijden per SG:

in:	CRAPTAB.C de waarde die staat op de positie van:	REGELAAR	DEBUGGER
	MG_TIJD	inst_MG_TIJD	inst_MG_TIJD
	MG_TIJD1	inst_MG_TIJD1	inst_MG_TIJD1
	MG_TIJD2	inst_MG_TIJD2	inst_MG_TIJD2
	MG_TIJD3	inst_MG_TIJD3	inst_MG_TIJD3
	MG_TIJD4	inst_MG_TIJD4	inst_MG_TIJD4
	MG_TIJD..	inst_MG_TIJD..	inst_MG_TIJD..
	MG_TIJD12	inst_MG_TIJD12	inst_MG_TIJD12

Bij initialisatie worden voor elke signaalgroep eerst de tijdsinstelling volgens de kolom CRAPTAB.C horizontaal overgezet in de kolom REGELAAR.
Indien inst_MG_TIJD "NG" is wordt inst_MG_TIJD (in de kolom REGELAAR) gevuld met inst_MG_TIJD1.
Indien inst_MG_TIJD én inst_MG_TIJD1 beide "NG" zijn, wordt inst_MG_TIJD (in de kolom REGELAAR) gevuld met 0.
Tijdens het regelen wordt voor het bepalen van de actuele waarde van de maximumgroentijd van een signaalgroep gebruik gemaakt van de waarde van inst_MG_TIJD uit de kolom REGELAAR.
De instructie kies_MG_TIJDj() zet de tijdsinstelling volgens inst_MG_TIJDj uit de kolom REGELAAR in inst_MG_TIJD in de kolom REGELAAR. Een nieuwe waarde van inst_MG_TIJD uit de kolom REGELAAR wordt voor de signaalgroep actueel bij de eerstvolgende start of herstart van de maximumgroentijd.
Bij een herstart van de regeling wordt de laatst gekozen MG_TIJDj gehandhaafd.

De tijden die via de debugger gewijzigd kunnen worden, kunnen door de fabrikant eveneens vanuit de procesbesturing (PB) via de Cie C-interface gewijzigd worden.

Het tijdens regelen wijzigen van een instelling van een maximumgroentijd
- door het programma als gevolg van bijvoorbeeld de instructie wijzig_MG_TIJD,
- door de debugger of
- door de fabrikant met behulp van het instelapparaat (via de PB),

heeft wijziging van de instelling van de gelijknamige maximumgroentijd in kolom REGELAAR tot gevolg.
Vanaf het moment dat een bepaalde maximumgroentijd is gekozen wordt dat door de regelaar onthouden en zorgt de regelaar er voor dat bij een wijziging de "bijbehorende" maximumgroentijd mee wordt gewijzigd. Dit werkt dus niet zolang er nog geen maximumgroentijd is gekozen. Het is daarom aan te bevelen in de functie init_bij_inschakelen() direct de juiste maximumgroentijd te kiezen.
Let op:
Door het geven van de instructie ‘kies_MG_TIJDj()’ worden voor alle signaalgroepen de in deze tabel opgegeven waarden voor de MG_TIJDj als maximumgroentijden ingesteld. Omdat deze instructies niet signaalgroepgebonden of blokgebonden zijn, kan gemakkelijk de fout worden gemaakt de laatste twee ronde haakjes te vergeten. Deze haakjes moeten er echter wel achter, aangezien deze instructie in C een functieaanroep is.
Hetzelfde geldt voor de extra maximumgroentijden.

Benamingen ten behoeve van de hiaattijden per D:

in:	CRAPTAB.C de waarde die staat op de positie van:	REGELAAR	DEBUGGER
	H_TIJD	inst_H_TIJD	inst_H_TIJD
	E_H_TIJD	inst_H_TIJD1	inst_H_TIJD1
		inst_H_TIJD2	inst_H_TIJD2

Bij initialisatie worden per detector eerst de tijdsinstellingen uit de kolom CRAPTAB.C horizontaal overgezet in de kolom REGELAAR.
Vervolgens wordt inst_H_TIJD (in de kolom REGELAAR) gevuld met inst_H_TIJD1.
Tijdens het regelen wordt voor het bepalen van de actuele waarden van de hiaattijden van een signaalgroep gebruikgemaakt van de waarden van inst_H_TIJD uit de kolom REGELAAR.
De instructie kies_H_TIJD(SGn) zet de tijdsinstelling volgens inst_H_TIJD1 uit de kolom REGELAAR in inst_H_TIJD in de kolom REGELAAR. De instructie kies_E_H_TIJD(SGn) zet de tijdsinstelling volgens inst_H_TIJD2 uit de kolom REGELAAR in inst_H_TIJD in de kolom REGELAAR. Een nieuwe waarde van inst_H_TIJD en uit de kolom REGELAAR wordt voor de detector actueel bij de eerstvolgende start of herstart van de betreffende hiaattijd.
Bij herstart van de regelaar wordt altijd de inst_H_TIJD1 gezet in de kolom REGELAAR en gaat de regelaar er van uit dat de gewone (dus niet de extra) hiaattijd is gekozen.

De tijden die via de debugger gewijzigd kunnen worden, kunnen door de fabrikant eveneens vanuit de PB via de Cie C-interface gewijzigd worden.

Het tijdens regelen wijzigen van een instelling van een hiaattijd
- door het programma als gevolg van b.v. de instructie wijzig_H_TIJD,
- door de debugger of
- door de fabrikant met behulp van het instelapparaat (via de PB)

heeft wijziging van de instelling van de gelijknamige hiaattijd in kolom REGELAAR tot gevolg. De regelaar houdt bij welke hiaattijd is gekozen en zorgt er voor dat bij een wijziging de ‘bijbehorende’ hiaattijd mee wordt gewijzigd.

4.2.6 Koppelsignalen

Met de RWS C-regelaar is het mogelijk meerdere werkende regelaars aan elkaar te koppelen.
In de CRAPTAB.C file is daartoe een structure opgenomen waarin koppelsignalen kunnen worden gedefinieerd (zie figuur 4.7).

```
/*     KOPPEL signalen tabel                                  */
struct KOPPEL             KOPPEL_tabel[] = {
/*     hulpfunctie:   ingangssignaal:   uitgangssignaal: KSU of KSH:        */
/*     NAAM,          x (of GEEN),      x (of GEEN),     KSU, KSH (of GEEN) */
       file_212,                1,             GEEN,     GEEN,
       leven_200,               2,             GEEN,     GEEN,
       va_groen_212,            3,             GEEN,     GEEN,
       tegenhouden_102,         4,             GEEN,     GEEN,
       naloop_208_108,          5,             GEEN,     GEEN,
       file_112,             GEEN,                6,      KSH,
       leven_100,            GEEN,                7,      KSU,
       kopp_actief,          GEEN,                8,      KSU,
       va_groen_112,         GEEN,                9,      KSU,
       groen_102,            GEEN,               10,      KSU,
       naloop_102_202,       GEEN,               11,      KSU,
       STOP};
```

Figuur 4.7: Voorbeeld van een koppelsignalentabel regelaar A

De regelaar gaat op de volgende wijze met deze koppelsignalen om.

Allereerst de uitgaande koppelsignalen.
In de structure zoals weergegeven in figuur 4.7 moet de symbolische naam van een hulpvariabele worden opgegeven en het nummer van het bijbehorende uitgaande koppelsignaal. Deze hulpvariabele kan op normale wijze in de regelaar worden gemanipuleerd. Vlak voordat de regelaar in een systeemronde terugkeert naar de

procesbesturing, wordt van alle uitgaande koppelsignalen bekeken of de betreffende HF waar resp. niet waar is, en schrijft de regelaar het waar zijn resp. het niet waar zijn van het bijbehorende koppelsignaal in de Cie C-interface. De procesbesturing zorgt er dan voor dat dit koppelsignaal wordt verzonden naar de andere regelaar. Afhankelijk van de toewijzing ‘KSU’ of ‘KSH’ bepaalt de procesbesturing of een uitgaand koppelsignaal gereset of gehandhaafd moet blijven als de regelaar niet regelt. Uitgaande koppelsignalen met een KSU-toewijzing (of ‘GEEN’) worden door de procesbesturing gereset als de regelaar niet regelt. Bij een KSH-toewijzing worden uitgaande koppelsignalen als de regelaar niet regelt gehandhaafd door de procesbesturing.

Het bovenstaande betekent het volgende:
- bij een gewijzigde status van de uitgaande koppelsignalen worden deze verzonden,
- het nummer van het koppelsignaal komt in de interface te staan en niet het nummer (symbolische naam) van de HF,
- het nummer van het koppelsignaal wordt dan ook, samen met de waar zijn of niet waar zijn informatie, naar de andere regelaar verzonden.

In de andere regelaar wordt het koppelsignaal met het desbetreffende nummer als inkomend koppelsignaal binnengehaald. In de structures voor de koppelsignalen in de beide regelaars moet dus het nummer van het uitgaande koppelsignaal in de ene regelaar gelijk zijn aan het nummer van het corresponderende, inkomende koppelsignaal in de andere regelaar. De benaming van de bijbehorende HF's is niet relevant en kan vrij worden gekozen. Omwille van de leesbaarheid is het wel aan te bevelen deze benaming gelijk te houden. In- en uitgaande koppelsignalen behoeven niet achter elkaar door te worden genummerd. Er kan dus in dezelfde regelaar een uitgaand koppelsignaal met het nummer 1 zijn als een inkomend koppelsignaal met het nummer 1 (zie figuur 4.8). Dit kan echter verwarrend zijn.

```
/*     KOPPEL signalen tabel                                  */
struct KOPPEL            KOPPEL_tabel[] = {
/*     hulpfunctie:   ingangssignaal:   uitgangssignaal: KSU of KSH:        */
/*     NAAM,          x (of GEEN),      x (of GEEN),     KSU, KSH (of GEEN) */
       file_212,             GEEN,               1,     KSH,
       leven_200,            GEEN,               2,     KSU,
       va_groen_212,         GEEN,               3,     KSU,
       tegenhouden_102,      GEEN,               4,     KSU,
       naloop_208_108,       GEEN,               5,     KSU,
       file_112,                6,            GEEN,    GEEN,
       leven_100,               7,            GEEN,    GEEN,
       kopp_actief,             8,            GEEN,    GEEN,
       va_groen_112,            9,            GEEN,    GEEN,
       groen_102,              10,            GEEN,    GEEN,
       naloop_102_202,         11,            GEEN,    GEEN,
       STOP};
```

Figuur 4.8: Voorbeeld van een koppelsignalentabel regelaar B (gekoppeld met regelaar A)

De inkomende koppelsignalen
In de structures zoals weergegeven in figuren 4.7 en 4.8 moet de symbolische naam van een hulpvariabele worden opgegeven en het nummer van het bijbehorende inkomende koppelsignaal. Deze hulpvariabele wordt door de regelaar gemanipuleerd.
Het binnen halen van de koppelsignalen wordt door de procesbesturing verzorgd. Direct nadat de regelaar in een systeemronde de applicatie heeft aangeroepen, wordt bekeken of er door de procesbesturing koppelsignalen in de Cie C-interface zijn geplaatst. Is dat het geval dan maakt de regelaar de bijbehorende HF, afhankelijk van de waarde van het koppelsignaal, waar resp. niet waar.
Hierbij dient met het volgende rekening te worden gehouden.
Indien het waar en het niet waar zijn van een koppelsignaal zo snel achter elkaar door de andere machine wordt verzonden dat bij het verwerken van de inkomende koppelsignalen door de ontvangende machine beide standen van het koppelsignaal worden gezien, worden beide standen weliswaar verwerkt, maar omdat dit

direct achter elkaar gebeurt, wordt in de applicatie slechts de laatst binnengekomen stand ‘gezien’; het is daarom verstandig koppelsignalen niet korter te laten bestaan dan tenminste 2 systeemronden van de ontvangende machine.

4.2.7 Regelingen

In de regelingenstructure in de CRAPTAB.C worden gekoppelde regelingen kenbaar gemaakt aan het programma. Om in een applicatie de status van gekoppelde regelingen te kunnen bepalen, is de volgende referentie beschikbaar:

status_regelen(nr)
Tussen de haken wordt het nummer van de gewenste regeling ingevuld.

De referentie aan status_regelen(nr) retourneert de waarde ‘TRUE’ (=1) als de regeling met nummer nr in de stand regelen (RWS_STATUS_REGELEN) staat. Dit houdt tevens in dat de verbinding met de gekoppelde regelaar in orde is. Is nr onbekend, danwel staat de bedoelde regeling niet in de stand regelen of is de verbinding verbroken, zal de waarde ‘FALSE’ (=0) worden geretourneerd.

4.2.8 Gekoppelde signaalgroepen

Indien signaalgroepen worden gekoppeld volgens paragraaf 6.6 van de Basisspecificatie dient het onderstaande in acht te worden genomen.

a. De beide te koppelen signaalgroepen moeten in hetzelfde blok primair of alternatief worden gespecificeerd.
b. De te koppelen signaalgroepen dienen beide dezelfde (fictieve) conflicten te hebben. Daar waar geen conflict nodig is dient een fictief-conflict te worden gedefinieerd.
c. De stucture voor ‘gekoppelde SG'en volgens BS_H66’ moet uiteraard worden ingevuld.
d. Indien voor de te koppelen signaalgroepen de MVG_OP WAAR is of kan worden, moeten de beide signaalgroepen elkaar mee-aanvragen anders kan de regeling vastlopen!
e. Indien de te koppelen signaalgroepen elkaar niet mee-aanvragen is het verstandig de stuctures voor de ‘extra op te geven SG'en volgens BS_H5 formule t en ac’ in te vullen met de hierboven bedoelde fictieve conflicten.

4.2.9 Diversen

Fasebewaking zal in de applicatiespecificatie moeten worden gespecificeerd met behulp van de instructie set_FB().
Let op:
Door het geven van de instructie ‘set_FB()’ worden in de Cie C-interface het fasebewakingsbit ten behoeve van de PB opgezet. Omdat deze instructie niet signaalgroepgebonden of blokgebonden is, kan gemakkelijk de fout worden gemaakt de laatste twee ronde haakjes te vergeten. Deze haakjes moeten er echter wel achter, aangezien deze instructie in C een functieaanroep is.
Fixatie zal in de applicatiespecificatie moeten worden gespecificeerd met behulp van de referentie FIX.

Indien bepaalde handelingen een directe relatie hebben met hetgeen door de PB in de Cie C-interface wordt geschreven kan het vanuit het oogpunt van beperking van de procestijd gewenst zijn deze handelingen alleen te laten uitvoeren als de PB daadwerkelijk in de Cie C-interface heeft geschreven. De referentie nw_info_in_INT geeft dat aan.

De referentie W_REG_VRI geeft aan of het verkeersregeltoestel daadwerkelijk regelt.

De afhandeling van selectieve detectie is vastgelegd in de Cie C-interface. De fabrikant dient de benodigde informatie op de daarvoor bestemde plaatsen te schrijven. Ten behoeve van de behandeling van selectieve detectie is een aantal referenties beschikbaar, allen beginnend met DSI_. Deze referenties zijn terug te vinden in bijlage 2.

Het blokkenschema kan worden gewijzigd met de toewijzingsinstructies voor realisaties. Om de omschakelingen op een veilige manier uit te voeren, zijn er ook ondersteunende instructies beschikbaar. Deze functies zijn uitgebreid beschreven in de [BS] en terug te vinden in bijlage 2.

Extra informatie kan in de Cie C-interface worden geplaatst door in de applicatiespecificatie gebruik te maken van de instructie plaats_in_INT(x, a). Hierbij geldt dat x de positie in de interface is en dat a de te plaatsen waarde is waarvoor geldt -32.768<=a<=32.767. De applicatieschrijver zal zelf de waarde van x moeten bijhouden indien op meerdere plaatsen moet worden geschreven. Hierbij kan gebruik worden gemaakt van de referentie eerste_plaats_positie+i. Deze referentie bepaalt het begin waar vanaf geplaatst kan worden. De toevoeging i bepaalt de positie (0<=i<=126) van elke volgende plaats.
Tevens zal hij er voor moeten zorgen dat de fabrikant van het verkeersregeltoestel weet wat met de betreffende informatie moet worden gedaan.

Extra informatie kan uit de Cie C-interface worden gehaald door in de applicatiespecificatie gebruik te maken van de refentie haal_uit_INT(y). Hierbij geldt dat y de positie in de interface is. Voor de waarde b die wordt opgehaald geldt –32.768<=b<=32.767. De applicatieschrijver zal zelf de waarde van y moeten bijhouden indien op meerdere plaatsen moet worden gelezen. Hierbij kan gebruik worden gemaakt van de referentie eerste_ophaal_positie+i. Deze referentie bepaalt het begin waar vanaf opgehaald kan worden. De toevoeging i bepaalt de positie (0<=i<=126) van elke volgende plaats.
Tevens zal hij er voor moeten zorgen dat de fabrikant van het verkeersregeltoestel weet wat op de betreffende plaatsen in de Cie C-interface moet worden geschreven.

Een andere manier om informatie naar de Cie C-interface te schrijven om of informatie hieruit te lezen is via de CIF_MON_UBER en CIF_MON_IBER buffers. Voor het gebruik van deze twee buffers zijn twee nieuwe functies beschikbaar. Afgesproken is dat boodschappen in deze buffers van elkaar worden gescheiden door het newline scheidingsteken \n. Zie verder voor nadere info.

Functie: Bool schrijf_MON_UBER(char *s)

Schrijft de tekst die in de string s staat in het CIF_MON_UBER buffer. Het scheidingsteken \n wordt automatisch toegevoegd als het nog niet in de string aanwezig is.

Functie retourneert:
TRUE de string s kon succesvol in het buffer geplaatst worden
FALSE plaatsen niet succesvol (te lange tekst, buffer te klein, buffer te vol, etc.)
Toepassing in broncode applicatiebouwer:

```
if ( !schrijf_MON_UBER( "boodschap1" ) )
{
  /* niet goed gegaan, reageren? */
}

/* maar het mag ook zo */
```

```
schrijf_MON_UBER( "boodschap2\n" );
schrijf_MON_UBER( "boodschap 3" );

/* of zo */
/* CIF_MAX_MON_UBER = 1024 is maximum grootte */
char txt[CIF_MAX_MON_UBER];
strcpy( txt, "boodschap 4\n" );
schrijf_MON_UBER( txt );
```

Functie: Bool lees_MON_IBER(char *s)

Leest een bericht uit het CIF_MON_IBER buffer en plaatst dat in de string s. Het bericht wordt gelezen totdat de buffer leeg is, tot de maximale lengte (CIF_MAX_MON_IBER = 1024) is bereikt of totdat het scheidingsteken wordt gevonden.

Functie retourneert:

TRUE	er kon een bericht in de string s geplaatst worden
FALSE	er was geen bericht aanwezig in het buffer

Toepassing in broncode applicatiebouwer:

```
char txti[CIF_MAX_MON_IBER];
while ( lees_MON_IBER(txti) )
{
  printxy (0,4,"MON_IBER: %s", txti);
}
```

5 FLASH/FLEXSYT-II

5.1 Inleiding

FLASH (FLEXSYT Application Shell)) is een ontwikkelomgeving waarmee FLEXSYT-II simulaties kunnen worden uitgevoerd. De FLEXSYT-verkeersimulator simuleert het verkeer volgens het model van FLEXSYT-II. Met FLEXSYT-II kunnen simulaties worden uitgevoerd met Flexcol-76-regelingen, met RWS C-regelingen en met CCOL-regelingen. De verschillende typen regelaars kunnen in één simulatie voorkomen.

FLASH bestaat uit de volgende hoofdonderdelen:
- management tool
- bestandsorganisatie van de verschillende bestanden die worden gebruikt en aangemaakt bij het uitvoeren van verkeerssimulaties
- netwerk editor
- een grafische omgeving voor het ontwerpen en tekenen van verkeerskundige netwerken
- workbench
- besturingsomgeving voor verkeerssimulaties
- monitor
- visualisatieomgeving voor verkeerssimulaties.

FLASH draait onder Microsoft Windows 95 en hoger. Er is een handleiding van FLASH [FLASH] beschikbaar. In dit hoofdstuk worden enkele praktische tips gegeven voor het gebruiken van FLASH voor het uitvoeren van simulaties met de RWS C-regelaar.

5.2 Installatie

Voordat FLASH wordt opgestart, dienen enkele settings voor de compiler gecontroleerd te worden. Indien deze settings nog niet bestaan, moeten ze worden toegevoegd. Dat gaat via de omgevingsvariabelen van Windows (te bereiken via Configuratiescherm, Systeem, Geavanceerd, Omgevingsvariabelen). Het betreft het volgende (uitgaande van installatie van de compiler in de directory C:\Program Files\Microsoft Visual Studio):

Voor Microsoft Visual C++ Express / Microsoft Visual Studio 2010 dienen de volgende instellingen te worden toegevoegd:
Bij SET INCLUDE=
C:\Program Files\Microsoft Visual Studio 10.0\VC\INCLUDE;
C:\Program Files\Microsoft SDKs\Windows\v7.0A\include;

Bij SET LIB=
C:\Program Files\Microsoft Visual Studio 10.0\VC\LIB;
C:\Program Files\Microsoft SDKs\Windows\v7.0A\lib;

Bij SET PATH=
C:\Program Files\Microsoft Visual Studio 10.0\Vc\bin;
C:\Program Files\Microsoft Visual Studio 10.0\Common7\ide;

Het is mogelijk dat voor een correcte werking de paden omgewisseld geplaatst moeten worden.

Bij het compileren wordt gebruikgemaakt van de bestanden MSPDB100.DLL en MSOBJ100.DLL. Deze bestanden worden bij het installeren van de Microsoft compiler in de verkeerde directory gezet. Daarom moeten deze bestanden gekopieerd worden van ..\Common7\Ide naar ..\VC\Bin.

Na het opstarten van flash moet onder het pull-down-menu 'Instellingen' het submenu 'Locaties bestanden...' aangeklikt worden om de juiste paden voor de C-regelaar te kunnen instellen. Uitgaande van installatie van de RWS C-regelaar in de standaarddirectory C:\CR moeten de instellingen als volgt worden ingevuld:

Pad RWS-C code C:\CR\STANDARD
Pad nieuwe RWS-C C:\CR\REG\DEFAULT

Ook moet het pad voor de compiler aangegeven worden. Hierbij moet verwezen worden naar de subdirectory BIN van de betreffende compiler. Is bijvoorbeeld de compiler in de directory C:\Program Files\Microsoft Visual Studio 10.0\Vc\bin geïnstalleerd, dan wordt het pad als volgt ingesteld:
Pad RWS C-compiler C:\Program Files\Microsoft Visual Studio 10.0\Vc\bin

Na het opstarten van FLASH moet onder het pull-down-menu 'Instellingen' het submenu 'Locaties bestanden...' aangeklikt worden om de juiste paden voor de C-regelaar te kunnen instellen. Uitgaande van installatie van de RWS C-regelaar in de standaarddirectory C:\CR moeten de instellingen als volgt worden ingevuld:

Pad RWS-C code C:\CR\STANDARD
Pad nieuwe RWS-C C:\CR\REG\DEFAULT

Ook moet het pad voor de compiler aangegeven worden. Hierbij moet verwezen worden naar de subdirectory BIN van de betreffende compiler. Is bijvoorbeeld de compiler in de directory C:\PROGRAM FILES\DEVSTUDIO\VC98 geïnstalleerd, dan wordt het pad als volgt ingesteld:
Pad RWS C-compiler C:\PROGRA~1\DEVSTU~1\VC98\BIN

5.3 Gebruik RWS C-regelaar binnen FLASH

5.3.1 Koppeling RWS C-regelaar aan FLASH

Om de koppeling tussen de RWS C-regelaar en FLASH te kunnen maken, moet de regelaar bij FLASH bekend zijn. Bij het aanmaken van een nieuw project (pull-down-menu 'Bestand', keuze 'Nieuw project') kunnen nieuwe regelaars worden aangemaakt. Dit kan ook binnen het pull-down-menu 'Instellingen', keuze 'Regelaars'. In de door de gebruiker aangegeven directory kan FLASH de volgende (lege) bestanden plaatsen: CRAPDEF.H, CRAPCOD.C, CRAPTAB.C, MAKEFILE en RWSCREG.DEF. Indien de files CRAPDEF.H, CRAPCOD.C en CRAPTAB.C al bestaan, wordt gevraagd of ze overschreven moeten worden.

5.3.2 Kruispuntplaatje en FLEXSYT-II netwerk

In de netwerk editor kan het kruispuntplaatje worden getekend en kan het FLEXSYT-netwerk worden ontworpen. FLASH haalt de detectoren en signaalgroepen uit een CONFIL file. Om de getekende detectoren en

signaalgroepen op eenvoudige wijze aan het netwerk te kunnen koppelen, moet eerst per regelaar een CONFIL worden aangemaakt via de button 'FLXRWS' in het hoofdscherm van het project.

5.3.3 Compileren RWS C-regeling

Alvorens te compileren, dient per regelaar een CONFIL te worden aangemaakt via de button 'FLXRWS' in het hoofdscherm van het project.

Om de regeling binnen FLASH te kunnen compileren, is de MAKEFILE nodig. Controleer daarom of het MAKEFILE-bestand in de werkdirectory voor de betreffende C-regelaar staat en of de juiste versie van de makefile actueel is. Is dit niet het geval, dan is waarschijnlijk de instelling van 'Pad nieuwe RWS-C' niet correct, of is de directory C:\CR\REG\DEFAULT leeg. In deze gevallen moet het juiste pad worden opgegeven of moeten de lege basisfiles en de default-directory worden geplaatst.

Na het compileren zijn de volgende bestanden aangemaakt in de werkdirectory van de betreffende C-regelaar:
CONFIL is nodig voor simulatie (koppeling signaalgroepen en detectoren): niet verwijderen

RWSCREG.DLL	DLL van de regelaar: niet verwijderen
CRAPTAB.ERR	bevat eventuele foutmeldingen m.b.t. CRAPTAB.C
CRAPCOD.ERR	bevat eventuele foutmeldingen m.b.t. CRAPCOD.C
LINK.ERR	bevat eventuele foutmeldingen m.b.t. het linken
CONV.C	
CONV.OBJ	
CRAPCOD.OBJ	
CRAPTAB.OBJ	
CRAPDIM.OBJ	
CCIE_INT.OBJ	

De CONFIL en de RWSCREG.DLL dienen bewaard te worden. Zij zijn nodig voor het uitvoeren van een simulatie. De overige bestanden in de lijst mogen verwijderd worden. Uiteraard moeten de CRAPDEF.H, CRAPCOD.C, CRAPTAB.C, de MAKEFILE ook bewaard worden.

5.3.4 Workbench

Binnen de workbench van FLASH kunnen handmatige simulaties en simulaties met FLEXSYT-II worden uitgevoerd. Ook is het mogelijk de detectoren tijdens een FLEXSYT-simulatie handmatig te beïnvloeden. Het functioneren van de verkeersregeling kan bekeken worden in de monitor. Het is bijvoorbeeld mogelijk de simulatie te versnellen, stap voor stap te laten uitvoeren of te laten pauzeren. Hiernaast is het bijvoorbeeld mogelijk om per regelaar de volgende acties uit te voeren:

- weergave van de interne signaalgroeptoestand weer te geven;
- activeren van fixatie;
- aanmaken van een dump;
- activeren van de debugger (zie hoofdstuk 6);
- herstarten van de regelaar;
- beïnvloeden van de status van de regelaar (bijvoorbeeld geel knipperen, alles rood).

5.3.5 Overige functionaliteiten

Naast het uitvoeren van de verschillende FLEXSYT-II deelprogramma's kunnen twee hulpprogramma's voor de RWS C-regelaar worden aangestuurd. Het betreft het programma MKDEBFLS voor het aanmaken van debugbestanden en het programma POSTDUMP voor het omzetten van een (relatief) dumpbestand naar een functioneel dumpbestand. Beide programma's worden in hoofdstuk 7 besproken.

6 De debugger

6.1 Inleiding

De debugger is een programma dat de mogelijkheid biedt om vrijwel alle variabelen en hulpvariabelen van de RWS C-regelaar te bekijken en parameterinstellingen te wijzigen. De maximale te displayen waarde in de debugger is 32.767.

Het bekijken van de waarden van de variabelen en de hulpvariabelen gebeurt aan de hand van, door de gebruiker, vooraf te definiëren debugschermen. Debugfiles kunnen met het hulpprogramma MKDEBFLS.EXE (zie hoofdstuk 7) op een gebruikersvriendelijke manier worden aangemaakt.

In de debugschermen kunnen items worden gedefinieerd; elk item geeft de waarde of de toestand van een hulpvariabele of van een variabele weer. Het opvragen en wijzigen van de variabelen en hulpvariabelewaarden in de testomgeving gebeurt binnen FLASH (FLASH-versie debugger). Het debuggen met een verkeersregelautomaat voor de afname (straat-versie debugger) gebeurt via een seriële verbinding. De opmaak van de debugschermen is gelijk voor de FLASH- en de straat-versie van de debugger.

6.2 Gebruik debugger onder FLASH

De debugger onder FLASH bestaat uit een debugscherm (boven) en een meldingenvenster (onder). In het debugscherm worden de betreffende waarden uit de regelaar weergegeven. In het meldingenvenster wordt gemeld dat de debugschermen zijn geladen en worden eventuele inconsistenties in de opmaak van de debugschermen weergegeven.

Met de toetsen PgUp en PgDn kunnen de schermen worden gewisseld. De bediening van de debugger met functietoetsen is als volgt (het venster van de debugger moet geactiveerd zijn):
- F3: Stop weergeven van de variabelen. De variabelen in het scherm worden niet meer ververst.
- F5: Continue weergeven van de variabelen. De variabelen in het scherm worden continu ververst.
- <enter>: Wijzig parameter. De instelling van de geselecteerde parameter zal in de regelaar worden gewijzigd met de waarde zichtbaar op het scherm. Indien de regelaar deze wijziging niet toestaat, dan wordt de werkelijke waarde op het scherm teruggeplaatst.

Wanneer de regelaar gestopt is en daarna opnieuw wordt gestart, moet de debugger met F5 expliciet worden verzocht om de variabelen te verversen. Na het opnieuw starten van de regelaar weet deze niet dat er nog een debugger actief is waar gegevens naartoe moeten worden gestuurd. Door eenmalig de optie 'continue weergeven van de variabelen' te selecteren, worden de variabelen weer coninue ververst.
Naast de functietoetsen zijn er toetscombinaties waarmee de debugger gemanipuleerd kan worden:
- Ctrl+O: Open een nieuw debugscherm
- Ctrl-S: Sluit één van de aanwezige debugschermen
- Ctrl-K: Kies/activeer één van de aanwezige debugschermen
- PgUp: Toon (indien aanwezig) het vorige debugscherm
- PgDn: Toon (indien aanwezig) het volgende debugscherm
- Ctrl-L: Selecteer een ander lettertype.

De besproken opties van de debugger zijn ook bereikbaar via pull-down-menu's.

Het selecteren van een variabele in het debugscherm kan zowel met de muis als met één of meerdere toetsencombinaties. Door met de muis een variabele aan te klikken, wordt deze variabele geselecteerd. Als

het een boolean variabele betreft, wordt geprobeerd deze variabele te inverteren (het actief zijn van een boolean variabele wordt met een 'vinkje' weergegeven). Door het gelijktijdig indrukken van de Ctrl-toets samen met één van de pijltjestoetsen kan één van de buren van de actieve variabele worden geselecteerd. Als het een boolean variabele betreft, kan met de spatiebalk de waarde van de boolean variabele worden gewijzigd.

Waarden van parameters in de RWS C-regelaar worden gewijzigd door in het debugscherm bij het betreffende item een andere waarde in te vullen en vervolgens op <enter> te drukken om de nieuwe waarde te activeren.

6.3 Debuggen voor een afname (straat-versie)

De debuggerversie voor op straat functioneert vrijwel hetzelfde als de debugger onder FLASH (zie 6.2).

De verschillen met de FLASH -versie zijn:
- de FLASH -versie is een zogenaamde DLL, de straat-versie een executable (EXE);
- communicatie met de regeling gaat via het computergeheugen (FLASH -versie) dan wel via de seriële kabel (straat-versie);
- in de straat-versie is de functionaliteit van DPCOM opgenomen (zie hieronder).
- in de straat-versie is functionaliteit voor automatische afname “AutoAfname” opgenomen (zie hieronder).
- in de straat-versie is functionaliteit voor automatische parameteraanpassing “AutoAanpas” opgenomen (zie hieronder).

De executable kan als elk Windows programma normaal worden opgestart (CRDEBUG.EXE).

Standaard worden de volgende instellingen gebruikt:
- seriële poort: COM1;
- snelheid seriële poort (baudrate): 9600;
- standaard komt het eerste debugscherm in beeld.

Met zogenaamde opstartparameters kunnen deze instellingen anders worden gekozen. Het instellen van de seriële poort kan op de volgende twee manieren:

1. met een '-' code aangeven welke instelling wordt bedoeld (volgorde is niet van belang):
   crdebug.exe -c <poort nummer> -b <baudrate>

   voorbeelden:
   - crdebug.exe -c 4
   - crdebug.exe -c 1 -b 9600
   - crdebug.exe -b 9600 -c 1

2. op eenvoudige wijze de argumenten invullen (volgorde ligt nu wel vast):
   crdebug <poort nummer of naam> <baudrate>

   voorbeelden:
   - crdebug.exe COM4
   - crdebug.exe 4
   - crdebug.exe COM1 9600
   - crdebug.exe 1 9600

Wanneer de debugger wordt gestart met DPCOM of -D als argument, dan komt DPCOM direct zichtbaar op het scherm.

voorbeelden:
- crdebug.exe -d
- crdebug.exe DPCOM

Wanneer de debugger wordt gestart met AFNAME of -A als argument, dan komt AutoAfname direct zichtbaar op het scherm.

voorbeelden:
- crdebug.exe -a
- crdebug.exe AFNAME

Wanneer de debugger wordt gestart met PARAMETERS of -P als argument, dan komt AutoAanpas direct zichtbaar op het scherm.

voorbeelden:
- crdebug.exe -p
- crdebug.exe PARAMETERS

DPCOM
De functionaliteit van DPCOM is opgenomen in de straatversie van de debugger. In het menu "Venster" van de debugger is de keuze "Start DPCOM" aanwezig. Na selectie van deze keuze wordt DPCOM opgestart.

DPCOM wordt gebruikt om een dump uit een regeling op te halen.
Na het opstarten verschijnen knoppen met de volgende mogelijkheden:

- Verificatie verbinding
- Maken dump -> LET OP: zie *)
- Ophalen dump
- Afbreken ophalen dump

- Schoonmaken statusregels
- DPCOM Afsluiten

*) een reeds bestaande dump wordt overschreven! Haal eventueel eerst de bestaande dump op.

Onder dit menu staan twee statusregels. In de eerste statusregel wordt weergegeven wat het programma verzonden heeft naar de regelaar. In de tweede statusregel wordt weergegeven wat het programma voor antwoord van de regelaar heeft ontvangen.
Onder de statusregels wordt weergegeven hoeveel bytes het programma van de regelaar heeft ontvangen.

De knoppen hebben de volgende betekenis:
- “Verificatie verbinding“ wordt gebruikt om te testen of de verbinding met de regelaar in orde is. Is dit het geval dan antwoordt de regelaar met: ER IS COMMUNICATIE MET DE BASISSPECIFICATIE VERSIE: x!
  Voor x wordt het versienummer van het basisspecificatiedeel van de regelaar ingevuld.
- “Maken dump“ wordt gebruikt om een dump in de regelaar aan te maken. De regelaar antwoordt dan met DUMP IS AANGEMAAKT!
  LET OP: Bij gebruik wordt een bestaande dump in de regelaar overschreven!
- “Ophalen dump“ wordt gebruikt om een dump op te halen. Bij een correcte verbinding antwoordt de regelaar met DUMP RWS C-REGELAAR waarna het verzenden van de dump plaatsvindt.

Tijdens het verzenden wordt het aantal bytes getoond dat door de pc is ontvangen. Bij een gemiddelde regeling bestaat een complete dumpfile al gauw uit 15.000 bytes. De totale transporttijd bedraagt enkele minuten.
Bij het einde van de dumpfile geeft de regelaar EINDE DUMP. Indien geen dump aanwezig is zal de regelaar antwoorden met DUMP NIET BESCHIKBAAR!

- “Afbreken ophalen dump“ - wordt gebruikt om het ophalen van een dump af te breken. De regelaar antwoordt dan met VERZENDEN DUMP IS AFGEBROKEN!
  Indien “Afbreken ophalen dump” wordt gebruikt terwijl er geen dump wordt opgehaald antwoordt regelaar met GEEN AKTIE UITGEVOERD!
- “Schoonmaken statusregels” wordt gebruikt om de informatie in de statusregels te wissen.

Indien na een commando geen communicatie met de regelaar tot stand is gekomen, geeft het programma na enkele seconden aan dat er geen reactie is ontvangen.

Een opgehaalde dump wordt weggeschreven naar de file CRF.DMP op de harde schijf. Dit is een ASCII file die m.b.v. een editor of een tekstverwerkingsprogramma kan worden uitgelezen.
Meerdere opgehaalde dumps worden altijd achteraan een eventueel bestaande file CRF.DMP toegevoegd, zodat geen oude informatie verloren gaat.

De dumpinformatie blijft in het regelprogramma beschikbaar en kan dus eventueel meerdere keren worden opgevraagd.

Aangezien na compilatie de symbolische namen van variabelen en parameters zijn verdwenen bevat de dump slechts de absolute nummers van signaalgroepen, detectoren, hulpvariabelen en dergelijke. Deze nummers staan in de dumpfile tussen (). Met behulp van de file CRAPDEF.H kunnen deze nummers terugvertaald worden naar de symbolisch nummering (met het programma POSTDUMP.EXE, zie paragraaf 8.2).

Bijvoorbeel als in de CRAPDEF.H het volgende staat

```
#define SG08_4     21
```

dan betekent de volgende regel uit de dumpfile

```
D(21)     *    .   .   .
```

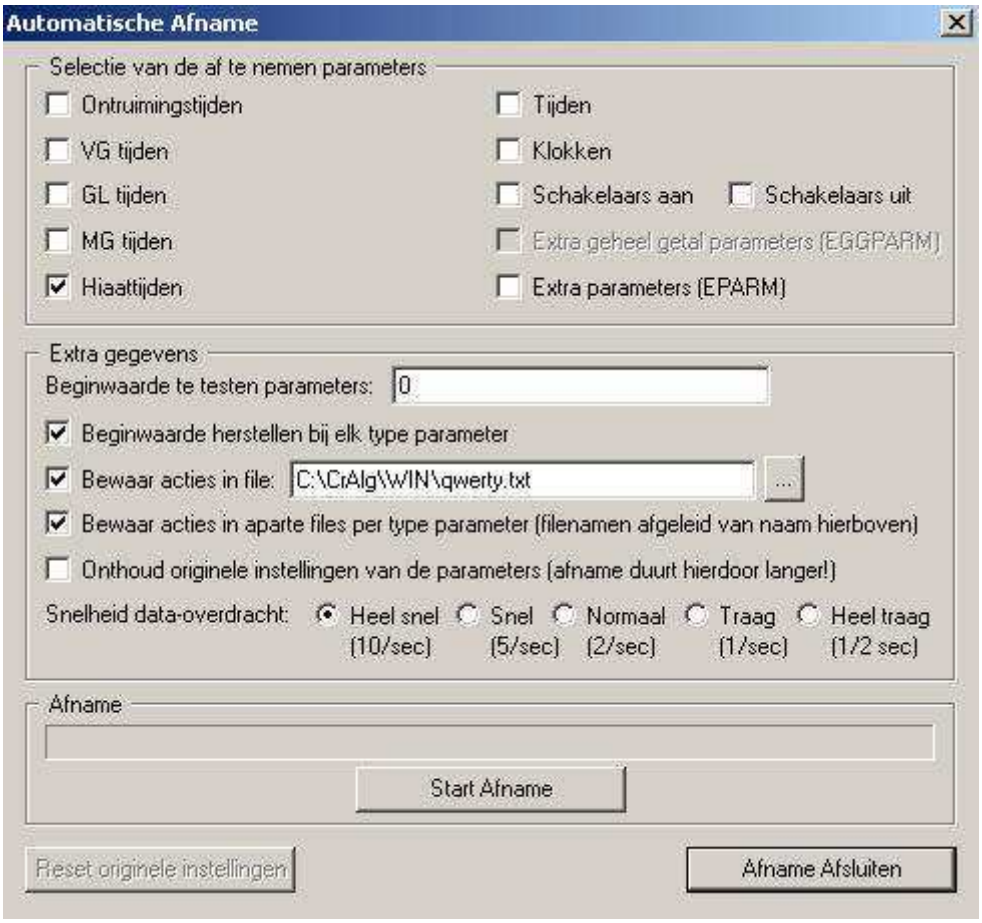
dat detector D(SG08_4) op staat en dat voor die detector geen fouten aanwezig zijn.

Overigens is het zo dat alleen die detectoren in een dump worden opgenomen die staan in de structure

```
struct SG_DETECTIE SG_DETECTIE_tabel[] ={}
```

Het is dus raadzaam alle detectoren in deze structure op te nemen.

Automatische afname
In het menu "Venster" van de debugger is de keuze "Start Automatische Afname" aanwezig. Na selectie van deze keuze wordt AutoAfname opgestart. Na het opstarten verschijnt het volgende scherm:



AutoAfname wordt gebruikt om veel terugkerende handelingen van een afname in het toestel te automatiseren. Middels AutoAfname kan een groot aantal parameters van een regeling een bepaalde waarde worden gegeven. Welke parameters dit zijn, welke waarde toegekend wordt etc. kan geregeld worden in het invoerscherm.

Auto Afname verzorgt een invoerscherm met daarop de volgende mogelijkheden:

Selectie van de af te nemen parameters: de diverse typen parameters (zoals onder andere ontruimingstijden, VG en GL tijden, tijden, klokken, EGGPARM parameters en EPARM parameters) kunnen onafhankelijk van elkaar worden geselecteerd middels een checkbox. Niet in de regeling aanwezige parametertypen worden grijs weergegeven, deze zijn dus niet selecteerbaar. De gebruiker dient hier te kiezen welke parameter(s) getest zullen gaan worden. Niet aanwezige parameters, bv. SG04, in een parameterlijst SG01, SG02, SG03, SG05, SG06 etc. worden genegeerd (“gat” in de definitielijst).

Invoerveld om beginwaarde in te stellen. De beginwaarde mag maximaal 4095 zijn, dit is een grens die afkomstig is uit het communicatieprotocol van de debugger. Wordt de waarde groter dan 4095, dan wordt deze weer op 0 gezet. Op het moment dat de automatische afname gestart wordt krijgen de waarden van de diverse parameters een waarde afhankelijk van deze beginwaarde.

Optieveld om de beginwaarde te herstellen bij elk type parameter. Wordt deze optie niet geselecteerd, dan wordt de beginwaarde eenmalig ingesteld en blijft deze doortellen over alle typen parameters. Wordt de optie wel geselecteerd, dan begint elke nieuwe groep van te testen parameters met de beginwaarde.

Optieveld om aan te geven of uitgevoerde acties gelogt moeten worden in een uitvoerbestand, gecombineerd met een invoerveld voor de naam van dat uitvoerbestand. In dit uitvoerbestand zal per parametertype een tabel worden geplaatst waarin de gebruiker kan zien welke parameter/volgnummer welke waarde heeft gekregen. Hierin zal de logische naam van een parameter (bv. SG03) worden geplaatst in plaats van het onderliggende nummer (bv. 2).

Optieveld om aan te geven of de gelogte gegevens in één uitvoerbestand dienen te komen (met als naam de hierboven ingestelde naam) of een uitvoerbestand per geselecteerde parameter. De naam van de diverse uitvoerbestanden wordt dan samengesteld uit de hierboven ingevoerde naam gevolgd door een liggend streepje en een korte codering van het type parameter (de extensie blijft hetzelfde). Voorbeeld: de ingestelde naam is LOG.TXT. Dienen de parameters in aparte bestanden te worden bewaard, dan krijgen deze de namen LOG_ONT.TXT, LOG_VG.TXT, LOG_HIAAT.TXT, LOG_KLOK.TXT etc.

Optieveld om de originele instellingen van de parameters te onthouden. Is deze optie geselecteerd dan wordt bij het voor de eerste keer versturen van een parameter naar de regeling, de (originele) waarde opgevraagd die deze parameter in de regeling heeft. Deze waarde wordt dan bewaard om hem later bij “Reset originele instellingen” (zie hieronder) te kunnen terugsturen naar de regeling.

Snelheid data-overdracht: de parameters worden één voor één over de seriële verbinding naar het toestel gestuurd. Normaal gesproken kan dit met de hoogst mogelijke in te stellen snelheid, “heel snel”, waarbij er tien parameters per seconde worden verstuurd. Indien deze snelheid (communicatie) problemen oplevert, is het mogelijk om deze snelheid in een aantal stappen te vertragen tot een laagste snelheid van één parameter per twee seconden.

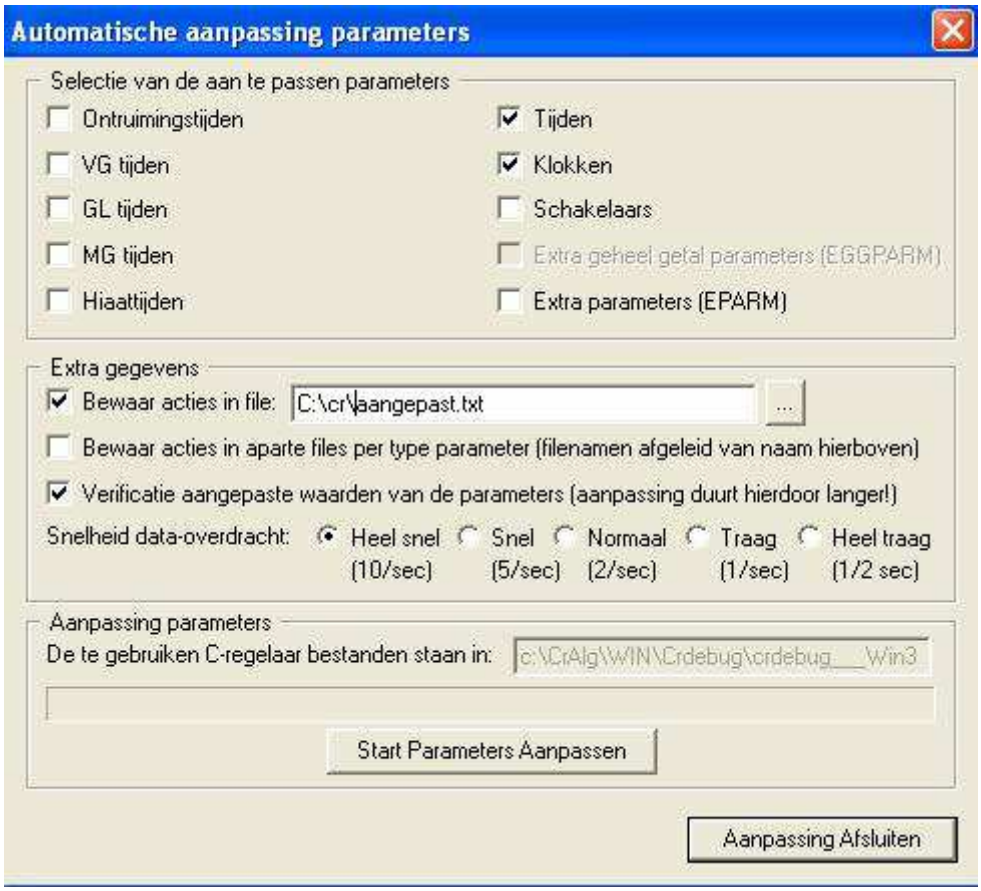
Weergaveveld waarmee de voortgang van de automatische afname kan worden gevolgd.

Start Afname: button waarmee de automatische afname wordt gestart. De diverse parameters krijgen een waarde waarbij de eerste parameter de waarde “beginwaarde” krijgt, de volgende “beginwaarde”+1, etc. De parameters worden één voor één naar de regelaar gestuurd, met de ingestelde pauze tussen elke parameter.

Reset originele instellingen: deze button kan alleen geselecteerd worden indien tenminste één originele instelling van een individuele parameter is opgevraagd en bewaard. Deze button maakt het mogelijk om na het testen met AutoAfname in één keer alle geteste parameters terug te zetten op hun originele instellingen.

Afname Afsluiten: button waarmee de automatische afname wordt afgesloten. De ingestelde waarden van dit invoerscherm worden bewaard, zodat bij opnieuw opstarten de “oude” waarden en selecties standaard worden ingevuld.

Automatische aanpassing parameters
In het menu "Venster" van de debugger is de keuze "Start Automatische Parameteraanpassing" aanwezig. Na selectie van deze keuze wordt AutoAanpas opgestart. Na het opstarten verschijnt het volgende scherm:



AutoAanpas wordt gebruikt om de handelingen van het aanpassen van parameters te automatiseren. Middels AutoAanpas kan een groot aantal parameters van een regeling een bepaalde waarde worden gegeven. Welke parameters dit zijn kan geregeld worden in het invoerscherm. De waarden die aan de diverse parameters worden gegeven zijn de waarden zoals die zijn opgenomen in de broncodebestanden CRAPTAB.C en CRAPDEF.H van de regeling.

Bij regelingen (met name bij CRSV-regelingen) komt het voor dat waarden van parameters wijzigen na de inbedrijfsstelling. Bijvoorbeeld bij CRSV indien de met behulp van modelcijfers berekende pulsen niet overeenkomen met de werkelijkheid. In dat geval tel je vaak een periode, om vervolgens de TRANSYT-berekeningen opnieuw uit te voeren. De dan berekende pulsen (in combinatie met wat op straat is gezien) moeten dan opnieuw worden ingesteld. Nu dient vaak op kantoor een lijst te worden gemaakt met parameters die moeten worden aangepast, die daarna stuk voor stuk via de debugger of het instelpaneel moeten worden aangepast. Foutgevoeliger en qua omstandigheden niet altijd optimaal. Temperatuur buiten en traagheid van de instelpanelen kan hierbij nadelig werken.

AutoAanpas verzorgt een invoerscherm met daarop de volgende mogelijkheden:

Selectie van de aan te passen parameters: de diverse typen parameters (zoals onder andere ontruimingstijden, VG en GL tijden, tijden, klokken, EGGPARM parameters en EPARM parameters) kunnen onafhankelijk van elkaar worden geselecteerd middels een checkbox. Niet in de regeling aanwezige parametertypen worden grijs weergegeven, deze zijn dus niet selecteerbaar. De gebruiker dient hier te kiezen welke parameter(s) aangepast zullen gaan worden.

Optieveld om aan te geven of uitgevoerde acties gelogt moeten worden in een uitvoerbestand, gecombineerd met een invoerveld voor de naam van dat uitvoerbestand. In dit uitvoerbestand zal per parametertype een tabel worden geplaatst waarin de gebruiker kan zien welke parameter/volgnummer welke waarde heeft gekregen. Hierin zal de logische naam van een parameter (bv. SG03) worden geplaatst in plaats van het onderliggende nummer (bv. 2).

Optieveld om aan te geven of de gelogte gegevens in één uitvoerbestand dienen te komen (met als naam de hierboven ingestelde naam) of een uitvoerbestand per geselecteerde parameter. De naam van de diverse uitvoerbestanden wordt dan samengesteld uit de hierboven ingevoerde naam gevolgd door een liggend streepje en een korte codering van het type parameter (de extensie blijft hetzelfde). Voorbeeld: de ingestelde naam is LOG.TXT. Dienen de parameters in aparte bestanden te worden bewaard, dan krijgen deze de namen LOG_ONT.TXT, LOG_VG.TXT, LOG_HIAAT.TXT, LOG_KLOK.TXT etc.

Optieveld om een verificatie te doen van de ingestelde waarden. Na het “setten” van een nieuwe waarde wordt deze waarde opgevraagd bij de regeling en bewaard. Op het eind van de complete aanpasactie worden alle ingestelde waarden vergeleken met de waarden die ze in de regeling gekregen hebben. Deze dienen uiteraard gelijk te zijn, anders volgt een foutmelding.

Snelheid data-overdracht: de parameters worden één voor één over de seriële verbinding naar het toestel gestuurd. Normaal gesproken kan dit met de hoogst mogelijke in te stellen snelheid, “heel snel”, waarbij er tien parameters per seconde worden verstuurd. Indien deze snelheid (communicatie) problemen oplevert, is het mogelijk om deze snelheid in een aantal stappen te vertragen tot een laagste snelheid van één parameter per twee seconden.

Weergaveveld ter controle van de directory van waaruit de broncode bestanden van de regeling worden gelezen. Er is geen beveiliging mogelijk tegen het sturen van parameters van een andere regeling, zorg er dus voor dat dit weergaveveld de juiste inhoud heeft.

Weergaveveld waarmee de voortgang van de automatische parameter aanpassing kan worden gevolgd.

Start Parameters Aanpassen: button waarmee de automatische parameter aanpassing wordt gestart. De diverse parameters krijgen de waarde zoals ingesteld in het CRAPTAB.C bestand. De parameters worden één voor één naar de regelaar gestuurd, met de ingestelde pauze tussen elke parameter.

Aanpassing Afsluiten: button waarmee de automatische parameter aanpassing wordt afgesloten. De ingestelde waarden van dit invoerscherm worden bewaard, zodat bij opnieuw opstarten de “oude” waarden en selecties standaard worden ingevuld.

7 Hulpprogramma's

Naast de in hoofdstuk 6 beschreven Debugger kent de RWS C-regelaar nog deze andere hulpprogramma's:
- MKDEBFLS: genereren en aanmaken debugschermen
- POSTDUMP: omzetten van een relatief dumpbestand naar een functioneel dumpbestand

Naast de twee genoemde programma's is er de internet-site van de RWS C-regelaar met voorbeelden, standaarden en updates (zie paragraaf 1.5).

7.1 MKDEBFLS

Het programma MKDEBFLS genereert automatisch debugschermen voor de RWS-C-regelaar en biedt de mogelijkheid om zelf debugschermen samen te stellen. Voor het genereren van debugschermen wordt uit de files CRAPTABB.C, CRAPCOD.C en CRAPDEF.H de benodigde informatie ingelezen.

MKDEBFLS kan uitsluitend vanuit een 32-bits Windows-omgeving worden opgestart. Voor het gebruik is dus minimaal Windows98 nodig.

7.1.1 Selecteren regeling

Voordat debugschermen gegenereerd of samengesteld kunnen worden, moet eerst een regeling geselecteerd worden. Dit gaat via het openen van de directory waar de regeling staat. In het hoofdmenu kan dit middels 'File'-'Open' of de button ‘openen folder’.

Nu worden de files CRAPTAB.C, CRAPDEF.H en CRAPCOD.C ingelezen. Indien het preprocessorstatement **#ifdef pc_omgeving** gebruikt is, wordt de gebruiker gevraagd of pc_omgeving als gedefinieerd moet worden beschouwd.
Tevens wordt de gebruiker attent gemaakt op lege structures en/of syntaxfouten in de diverse tabellen.

7.1.2 Automatisch genereren debugschermen

Na het selecteren van een directory kan in het hoofdmenu via het submenu 'Debugschermen' een venster worden geopend voor het automatisch genereren van debugschermen. De gebruiker ziet in dit venster de eerder geselecteerde directory. Deze is niet te wijzigen in dit venster.

Voor de naamgeving van de debugschermen zijn drie mogelijkheden. Default wordt de naam 'debug_??.avv'. Indien een andere naamgeving gewenst wordt, kan dit worden opgegeven.
De volgnummers van de debugschermen beginnen default met 01. Dit kan worden gewijzigd. Let er wel op dat na het genereren van debugscherm 99 geen nieuwe schermen meer worden aangemaakt. Wordt dus gestart met volgnummer 91, dan worden maximaal 9 schermen gegenereerd.

De volgnummers met de bijbehorende inhoud van de debugschermen worden in een tekstbestand MKDEBFLS.TXT geschreven.

Na de gewenste opties te hebben aangegeven, kan de gebruiker de debugschermen genereren door het aanklikken van de 'OK'-button.
Indien er al debugschermen bestaan met de geselecteerde naam, krijgt de gebruiker de mogelijkheid een nieuwe naam of volgnummer op te geven of de bestaande schermen te overschrijven.

Debugschermen kunnen op twee manieren worden gegenereerd:
- debugschermen voor testen
- debugschermen voor afname

Voor “testen“ worden de volgende schermen aangemaakt:
- signaalgroeptoestanden
- realisatie signaalgroepen
- opties
- instelling en lopen signaalgroeptijden
- instelling en lopen (garantie) ontruimingstijden
- instelling en lopen overige tijden
- schakelaars
- klokken
- tellers
- hulpvariabelen
- extra parameters
- detectie(fouten)
- koppelsignalen
- variabelen behorende bij een optie
- standaard hulpfuncties
- overige opties
- overige signaalgroepopties

Voor “afname“ wordt een selectie uit bovenstaande lijst aangemaakt:
- signaalgroeptoestanden
- instellingen tijden signaalgroepprocedure
- instelling en lopen (garantie) ontruimingstijden
- maximumgroentijden
- (extra) hiaattijden
- instelling en lopen overige tijden
- schakelaars
- klokken
- extra parameters
- detectie(fouten)
- koppelsignalen

7.1.3 Zelf debugschermen samenstellen

In het submenu 'Debugschermen' kan ook voor het 'definiëren van debugschermen' worden gekozen. Na het inlezen van de benodigde informatie dient, via 'Toevoegen' een of meer namen van extra debugschermen te worden opgegeven.
Selecteer een van de namen van de extra debugschermen en klik op de toets 'Bewerken'. Nu wordt een nieuw venster geopend met de mogelijkheid tot het toevoegen, verwijderen en wijzigen van een debug-item.

Het 'item Toevoegen' opent een venster met alle mogelijk debug-items. Door muisklikken is een specifiek item te selecteren. Met 'Toevoegen' wordt dit item in het debugscherm geplaatst. Een volgend item kan op dezelfde wijze worden toegevoegd. Dit zal in het debugscherm onder het eerste item worden geplaatst. Ook is het mogelijk een verklarende tekst in plaats van een debug-item toe te voegen aan het debugscherm.
Na 'Sluiten' zijn de geselecteerde debug-items en de verklarende tekst zichtbaar in het debugscherm.

Door middel van drag & drop kunnen de items door de gebruiker binnen het venster verplaatst worden. Indien items (deels) over elkaar geplaatst worden, wordt de gebruiker bij het opslaan attent gemaakt op deze fout.

Geselecteerde items kunnen ook weer worden verwijderd via een klik op 'item Verwijderen'.

De zelfgemaakte debugschermen worden opgeslagen in een database (in ASCII-formaat). De database heeft altijd de naam MKDEBFLS.DB en staat in de directory met de regelaar-bestanden. Een volgende keer dat het programma MKDEBFLS wordt gestart voor die betreffende regelaar, wordt de database ingelezen en kunnen de gemaakte debugschermen worden gewijzigd.

Na aanmaken of wijzigen van extra debugschermen dienen (opnieuw), via 'Genereren debugschermen', alle schermen te worden gegenereerd. De zelf gedefinieerde schermen zullen worden toegevoegd aan de standaardschermen.

7.2 POSTDUMP

Het programma ‘postdump.exe’ is onderdeel van de RWS C-regelaar en wordt gebruikt om een dump van een RWS C-regeling te voorzien van functionele namen.

De plaats waar het programma ‘postdump.exe’ staat, moet onder het pull-down-menu 'Instellingen', Locaties Bestanden' worden opgegeven.

Het programma wordt opgestart in de workbench met de knop "postdump". Ook is het mogelijk om postdump buiten FLASH te starten (er moet een pad openstaan naar het programma of het programma moet in de actieve directory worden opgenomen). Dit gebeurt met het volgende commando:
‘postdump <filenaam dumpfile> <filenaam output>’.
Indien geen filenamen worden opgegeven, gebruikt het programma voor de dumpfile CRF.DMP en voor de output DUMP.OUT.

Zowel de dumpfile als de file CRAPDEF.H van het betreffende kruispunt moeten in de huidige werkdirectory aanwezig zijn.

Indien er reeds een file bestaat met de output filenaam dan vraagt het programma of deze file overschreven mag worden. Het programma wordt gestopt als de gebruiker ontkennend antwoordt.

Het programma vraagt de gebruiker vervolgens om de naam van het kruispunt op te geven. Deze naam wordt in de uitvoer vermeldt teneinde identificatie mogelijk te maken.
Indien geen naam wordt opgegeven gebruikt het programma als naam "ONBEKEND".

De eerste 60000 karakters van een dumpfile worden verwerkt.

De volgende controles worden uitgevoerd:
- aanwezigheid dumpfile
- aanwezigheid file CRAPDEF.H
- controle inhoud file CRAPDEF.H op
    - aanwezigheid defines voor NUMBK, NUMSG en NUMOV
    - waarde NUMBK kleiner of gelijk aan 10
    - waarde NUMSG kleiner of gelijk aan 64
    - waarde overige NUM?? kleiner of gelijk aan NUMOV
    - dubbele definitie symbool met verschillende waarden

  - . meermaals voorkomen gereserveerde woorden binnen commentaar
  - . aanwezigheid gereserveerd woord horende bij NUM?? indien NUM?? voorkomt
  - . waarden symbolen per categorie kleiner dan NUM?? van die categorie
  - . expressie bevat -, * of / teken
- definitie van symbolen in CRAPDEF.H die voorkomen in de dumpfile
- regel in dumpfile langer dan 80 karakters
- voorkomen in de dumpfile van de volgende kopjes
  - . ALGEMEEN
  - . EINDE DUMP
  - . EXTRA PARAMETERS,
  - . HULPVARIABELEN,
  - . KLOKKEN,
  - . SCHAKELAARS,
  - . TIJDEN;
- controle op informatie van meerdere dumps in de dumpfile.

Indien één of meer van bovenstaande controles niet met goed resultaat worden afgesloten, stopt het programma met een foutmelding. De foutmelding geeft in dit geval aan welke controle niet goed is verlopen.

Indien NUM?? niet gedefinieerd is, wordt NUM?? gelijk gemaakt aan NUMOV.

Indeling CRAPDEF.H
Teneinde een zo duidelijk mogelijke uitvoer te krijgen verdient het aanbeveling om in de CRAPDEF.H alle categorieën variabelen te scheiden.

De start van elke categorie wordt aangegeven door een gereserveerd woord in een commentaarregel. De gereserveerde woorden zijn:
- blokken
- signaalgroepen
- detectoren
- tijden
- hulpvariabelen
- klokken
- schakelaars
- tellers
- extra geheel getal parameters
- extra parameters.

Iedere categorie wordt afgesloten door symbolen die de grootte van de categorie aangeven. Deze zijn:
- NUMBK
- NUMSG
- NUMD
- NUMT
- NUMHF
- NUMKL
- NUMSCH
- NUMTL
- NUMEGGP
- NUMEP.

Wanneer een categorie niet gebruikt wordt, hoeft deze niet te worden opgenomen.

Commentaar mag zowel volgens ANSI C met /* en */ als volgens C++ met // worden aangegeven.

Preprocessor commando's
De volgende preprocessor-commando's mogen worden gebruikt:
- #ifdef
- #ifndef
- #else
- #endif
- #define.

In #define-statements mogen symbolen gebruikt worden die door andere #define-statements zijn gedefinieerd. De enige toegestane expressie met symbolen is daarbij het optellen.

Bijlage 1: Foutmeldingen

1. Foutmeldingen door de PB
   De volgende meldingen kunnen op het scherm verschijnen:

   - ‘CIE_C_INTERFACE : iber overflow’ inkomende/uitgaande berichten buffer overflow.

   - ‘CIE_C_INTERFACE : uber overflow’ het contact met de debugger is verbroken, controleer de seriële
     verbinding.

   - ‘functie %x %x is onbekend’ onbekende functie-aanvraag van debugger: controleer de versie van de debugger en de RWS C-regelaar.

2. Foutmeldingen door de RWS C-regelaar
   Hier is sprake van interne fouten in de RWS C-regelaar. Neem contact op met de beheerder.

   De volgende fouten kunnen gemeld worden:
   - "FATALE FOUT: SG: .. in onbekende toestand: .. gekomen!"
   - "FATALE FOUT: Volgnummer RGA: .. is te groot!"
   - "FATALE FOUT: De waarde voor TIJD: .. is onbekend!"
   - "FATALE FOUT: Overflow inkomende berichtenbuffer!"
   - "FATALE FOUT: Wegschrijven kruispuntafbeelding mislukt!"
   - "FATALE FOUT: Inlezen kruispuntafbeelding mislukt!"
   - "FATALE FOUT: Te weinig geheugen om TIJD: .. in te stellen!"
   - "FATALE FOUT: Te weinig geheugen om TIJD: .. van SG: .. in te stellen!"
   - "FATALE FOUT: Te weinig geheugen om TIJD: .. van SG: .. en SG: .. in te stellen!"
   - "FATALE FOUT: Volgnummer voor gekoppelde regeling te hoog!"
   - "FATALE FOUT: Aantal klokken groter dan 20"

3. Foutmeldingen bij het invoeren van de applicatie

   Tijdens het compileren of linken kunnen fouten geconstateerd worden, die door de compiler resp. de linker gemeld worden met een code. Deze code kan opgezocht worden in de handleiding van de compiler.
   Het komt vaak voor dat de fout pas enkele regels later gemeld wordt, dan de plaats waar de fout staat. De reden is dan dat er op de gemelde plaats pas echt een syntaxfout geconstateerd wordt.

   Veel voorkomende fouten zijn:
   - vergeten van de haakjes ‘{ }’ bij meerdere coderegels onder een if- of while-statement
   - vergeten van de haakjes ‘( )’ bij aanroep van een functie zonder parameters
   - gebruik van ‘=’ als ‘==’ bedoeld wordt
   - vergeten van ‘,’ of ‘;’ bij afsluiting van een tabel of een statement
   - vergeten van de decimale punt.

   Vaak genereert de C-compiler een hele reeks fouten, die het gevolg zijn van de eerste fout. Let dus over het algemeen alleen op de eerste fout.

Bijlage 2: Referenties en instructies

Hieronder volgt een overzicht van alle mogelijke referenties en alle toegestane instructies in het applicatiedeel van de RWS C-regelaar met een korte toelichting.

Er zijn verschillende soorten referenties:
- toestandsreferenties (bijvoorbeeld A(SGn) staat voor signaalgroep n heeft een aanvraag, halterende_H_TIJD(SGn_i) staat voor een halterende hiaattijd van detector SGn_i)
- begin meldingen bij het waar geworden zijn van een toestand (begin_A(SGn) staat voor signaalgroep n heeft een aanvraag gekregen)
- einde meldingen bij het beëindigen van een toestand (einde_A(SGn) staat voor het wegvallen van de aanvraag voor signaalgroep n)
- instellingsreferenties (bijvoorbeeld inst_EPARM(naam) staat voor de instelling van de extra parameter naam)
- waardereferenties (bijvoorbeeld waarde_TIJD(naam) staat voor de actuele waarde van de (lopende) tijd naam)
- functionele referenties (bijvoorbeeld datum (b,c,d) voor de actuele kalenderdatum, eerste_plaats_positie+i voor een interfacepositie).

Met instructies is het mogelijk de toestand van variabelen te beïnvloeden. Het eerste deel van de instructienaam geeft aan op welke manier de variabele wordt beïnvloed, voorbeelden zijn:
- set_XX(...)- en reset_XX(...)-instructies
  hiermee wordt de toestand van een variabele functie waar of niet waar gemaakt of wordt de betreffende functie geactiveerd of ingetrokken
- start_XX(...)-, herstart_XX(...)-, halteer_XX(...)-, dehalteer_XX(...)-instructies:
  hiermee wordt het verloop van het aflopen van een tijdelement beïnvloed (bijvoorbeeld start_TIJD(naam) laat het tijdelement TIJD(naam) lopen)
- kies_XX(...)-instructies: het actueel maken van keuzesets tijden (geldt voor eerste hiaattijden, tweede hiaattijden en zes sets maximumgroentijden)
- wijzig_EPARM(naam), maak_TL(naam): het wijzigen van de instelling van een extra parameter of een teller
- incr_TL(naam), decr_TL(naam): het verhogen respectievelijk verlagen van de instelling van een teller
- plaats_in_INT(g, h): het plaatsen van informatie in de interface (bijvoorbeeld voor verklikking op het politiepaneel).

Meer gedetailleerde informatie over referenties en instructies zie BS.

referenties	instructies	toelichting
A(SGn) begin_A(SGn) einde_A(SGn)		aanvraag
AFK(SGn) begin_AFK(SGn) einde_AFK(SGn)		afkappen
AFK_OP(SGn) begin_AFK_OP(SGn) einde_AFK_OP(SGn)	set_AFK_OP(SGn) reset_AFK_OP(SGn)	afkapoptie
ALLES_ROOD_APPLICATIE		alle richtingen zijn rood
	set_ALT_toewijzing(SGn, BKl) reset_ALT_toewijzing(SGn, NKl)	toewijzing alternatieve realisatie
AR(SGn, BKl) begin_AR(SGn, BKl) einde_AR(SGn, BKl)		alternatieve realisatie
BA(SGn) begin_BA(SGn) einde_BA(SGn)		bezettijdaanvraag
BLOK(BKl) begin_BLOK(BKl) einde_BLOK(BKl)		blok
	reset_BLOKKENSCHEMA	reset alle primaire en alternatieve realisaties
BL(SGn) begin_BL(SGn) einde_BL(SGn)		blokkeren
BL_OP(SGn) begin_BL_OP(SGn) einde_BL_OP(SGn)	set_BL_OP(SGn) reset_BL_OP(SGn)	blokkeeroptie
BO(BKl) begin_BO(BKl) einde_BO(BK;)		blokophouden
BO_OP(BKl) begin_BO_OP(BKl) einde_BO_OP(BKl)	set_BO_OP(BKl) reset_BO_OP(BKl)	blokophoudoptie
BR(SGn) begin_BR(SGn) einde_BR(SGn)		bijzondere realisatie
D(SGn_i) begin_D(SGn_i) einde_D(SGn_i)	set_AVR_toewijzing_D(SGn_i) reset_AVR_toewijzing_D(SGn_i) set_BS_H5p_toewijzing_D(SGn_i) reset_BS_H5p_toewijzing_D(SGn_i) set_H1e_toewijzing_D(SGn_i) reset_H1e_toewijzing_D(SGn_i) set_H1e_spec_toewijzing_D(SGn_i) reset_H1e_spec_toewijzing_D(SGn_i) set_H2e_toewijzing_D(SGn_i) reset_H2e_toewijzing_D(SGn_i) set_H2e_spec_toewijzing_D(SGn_i) reset_H2e_spec_toewijzing_D(SGn_i)	detectie/detectiefuncties
DA(SGn) begin_DA(SGn) einde_DA(SGn)		detectieaanvraag

referenties	instructies	toelichting
dag(dg)		weekdag
dagnummer()		nummer van de kalenderdag
datum (4',2',2')		kalenderdatum
DFOUT(SGn_i)	set_DFA_toewijzing_D(SGn_i)	detectiebewaking
DFOUT_BG(SGn_i)	reset_DFA_toewijzing_D(SGn_i)	
DFOUT_OG(SGn_i)	set_DFU_toewijzing_D(SGn_i)	
	reset_DFU_toewijzing_D(SGn_i)	
DSI_LUS		sel. detectie: lusnummer
DSI_VTG		sel. detectie: voertuigcategorie
DSI_LYN		sel. detectie: lijnnummer
DSI_WDNST		sel. detectie: wagendienstnummer
DSI_BEDR		sel. detectie: bedrijfsnummer
DSI_NUM		sel. detectie: voertuignummer
DSI_DIR		sel. detectie: richtingsnummer
DSI_STAT		sel. detectie: voertuigstatus
DSI_PRI		sel. detectie: prioriteitsklasse
DSI_STP		sel. detectie: stiptheidsklasse
DSI_TSTP		sel. detectie: stiptheid [s]
DSI_LEN		sel. detectie: voertuiglengte [m]
DSI_SPD		sel. detectie: voertuigsnelheid [m/s]
DSI_LSS		sel. detectie: afstand tot stopstreep [m]
DSI_TSS		sel. detectie: passeertijd stopstreep [s]
DSI_RIT		sel. detectie: ritnummer
DSI_RITC		sel. detectie: ritcategorie
DSI_ROUT		sel. detectie: routenummer openbaar vervoer
DSI_TYPE		sel. detectie: type melding
DSI_OVC		sel. detectie: codering meldpunt openbaar vervoer
DSI_XGRAD		sel. detectie: breedtegraad graden
DSI_XMIN		sel. detectie: breedtegraad minuten
DSI_XSEC		sel. detectie: breedtegraad seconden
DSI_XHSEC		sel. detectie: breedtegraad honderdste seconden
DSI_YGRAD		sel. detectie: lengtegraad graden
DSI_YMIN		sel. detectie: lengtegraad minuten
DSI_YSEC		sel. detectie: lengtegraad seconden
DSI_YHSEC		sel. detectie: lengtegraad honderdste seconden
DSI_JAAR		sel. detectie: jaartal
DSI_MND		sel. detectie: maand
DSI_DAG		sel. detectie: dag
DSI_UUR		sel. detectie: uur
DSI_MIN		sel. detectie: minuten
DSI_SEC		sel. detectie: seconden
eerste_ophaal_positie+g		voor haal_uit_INT(g)
eerste_plaats_positie+g		voor plaats_in_INT(g,h)
inst_EGGPARM(naam)	wijzig_EGGPARM(naam, w)	extra geheel getal parameter
inst_EPARM(naam)	wijzig_EPARM(naam, y)	extra parameter
	set_FB()	fasebewaking
FIX		fixatie
G(SGn)		groen
begin_G(SGn)		
einde_G(SGn)		
GEEN_blk_g(SGn)	set_GEEN_blk_g(SGn)	t.b.v. uitbreiden PPA
begin_GEEN_blk_g(SGn)	reset_GEEN_blk_g(SGn)	
einde_GEEN_blk_g(SGn)		

referenties	instructies	toelichting
GG_TIJD(SGn) begin_GG_TIJD(SGn) einde_GG_TIJD(SGn) inst_GG_TIJD(SGn) waarde_GG_TIJD(SGn)		garantiegroentijd
GGL_TIJD(SGn) begin_GGL_TIJD(SGn) einde_GGL_TIJD(SGn) inst_GGL_TIJD(SGn) waarde_GGL_TIJD(SGn)		garantiegeeltijd
GL(SGn) begin_GL(SGn) einde_GL(SGn)		geel
GL_TIJD(SGn) begin_GL_TIJD(SGn) einde_GL_TIJD(SGn) halterende_GL_TIJD(SGn) inst_GL_TIJD(SGn) waarde_GL_TIJD(SGn)	start_GL_TIJD(SGn) herstart_GL_TIJD(SGn) reset_GL_TIJD(SGn) halteer_GL_TIJD(SGn) dehalteer_GL_TIJD(SGn)	geeltijd
GO_TIJD(SGn, SGj) begin_GO_TIJD(SGn, SGj) einde_GO_TIJD(SGn, SGj) inst_GO_TIJD(SGn, SGj) waarde_GO_TIJD(SGn, SGj)		garantie ontruimingstijd
GR_TIJD(SGn) begin_GR_TIJD(SGn) einde_GR_TIJD(SGn) inst_GR_TIJD(SGn) waarde_GR_TIJD(SGn)		garantie roodtijd
HF(naam) begin_HF(naam) einde_HF(naam)	set_HF(naam) reset_HF(naam)	hulpvariabele
H_TIJD(SGn_i) begin_H_TIJD(SGn_i) einde_H_TIJD(SGn_i) halterende_H_TIJD(SGn_i) inst_H_TIJD(SGn_i) inst_H_TIJD1(SGn_i) inst_H_TIJD2(SGn_i) waarde_H_TIJD(SGn_i)	start_H_TIJD(SGn_i) herstart_H_TIJD(SGn_i) reset_H_TIJD(SGn_i) halteer_H_TIJD(SGn_i) dehalteer_H_TIJD(SGn_i) wijzig_H_TIJD(SGn_i, y) kies_H_TIJD(SGn_i) kies_E_H_TIJD(SGn_i)	hiaatijd
haal_uit_INT(g)		ophalen uit interface
nw_info_in_INT		er is nieuwe informatie in interface
plaats_in_INT (g, h)		plaatsen waarde ‘h’ in interface
	set_INITIEEL_BLOKKENSCHEMA	wijzigt blokkenschema in initiele instelling
jaarnummer()		geeft jaarnummer terug

referenties	instructies	toelichting
KL(naam) begin_KL(naam) einde_KL(naam)		klok
	set_KSH_toewijzing(HFi) reset_KSH_toewijzing(HFi) set_KSU_toewijzing(HFi) reset_KSU_toewijzing(HFi)	toewijzingen voor uitgaande koppelsignalen
	lees_MON_IBER	lezen uit Cie C-interface
	maak_ALLES_ROOD()	maakt alle richtingen rood (na GG_tijden)
maandnummer()		geeft maandnummer terug
MA(SGn) begin_MA(SGn) einde_MA(SGn)	set_MA(SGn) reset_MA(SGn)	meeaanvraag
MAH4(SGn, i) begin_MAH4(SGn, i) einde_MAH4(SGn, i)		meeaanvraag-hulpvariabele 4
MAH5(SGn, i) begin_MAH5(SGn, i) einde_MAH5(SGn, i)		meeaanvraag-hulpvariabele 5
MAH6(SGn, i) begin_MAH6(SGn, i) einde_MAH6(SGn, i)		meeaanvraag-hulpvariabele 6
MAH7(SGn, i) begin_MAH7(SGn, i) einde_MAH7(SGn, i)		meeaanvraag-hulpvariabele 7
MA_OP(SGn) begin_MA_OP(SGn) einde_MA_OP(SGn)	set_MA_OP(SGn) reset_MA_OP(SGn)	meeaanvraag optie
MG_TIJD(SGn) begin_MG_TIJD(SGn) einde_MG_TIJD(SGn) halterende_MG_TIJD(SGn) inst_MG_TIJD(SGn) waarde_MG_TIJD(SGn)	start_MG_TIJD(SGn) herstart_MG_TIJD(SGn) reset_MG_TIJD(SGn) halteer_MG_TIJD(SGn) dehalteer_MG_TIJD(SGn) wijzig_MG_TIJD(SGn, y)	maximumgroentijd
inst_MG_TIJD1(SGn) inst_MG_TIJD2(SGn) inst_MG_TIJD3(SGn) inst_MG_TIJD4(SGn) inst_MG_TIJD5(SGn) inst_MG_TIJD6(SGn) inst_MG_TIJD7(SGn) inst_MG_TIJD8(SGn) inst_MG_TIJD9(SGn) inst_MG_TIJD10(SGn) inst_MG_TIJD11(SGn) inst_MG_TIJD12(SGn)	kies_MG_TIJD1() kies_MG_TIJD2() kies_MG_TIJD3() kies_MG_TIJD4() kies_MG_TIJD5() kies_MG_TIJD6() kies_MG_TIJD7() kies_MG_TIJD8() kies_MG_TIJD9() kies_MG_TIJD10() kies_MG_TIJD11() kies_MG_TIJD12()	

referenties	instructies	toelichting
MVG(SGn) begin_MVG(SGn) einde_MVG(SGn)		meeverlenggroen
O_TIJD(SGn, SGj) begin_O_TIJD(SGn, SGj) einde_O_TIJD(SGn, SGj) inst_O_TIJD(SGn, SGj) waarde_O_TIJD(SGn, SGj)	start_O_TIJD(SGn, SGj) reset_O_TIJD(SGn, SGj)	ontruimingstijd
PG(SGn, BKl) begin_PG(SGn, BKl) einde_PG(SGn, BKl)		primair gerealiseerd
PPA(SGn, BKl) begin_PPA(SGn, BKl) einde_PPA(SGn, BKl)		periode privilege alternatief
PPAH(SGn) begin_PPAH(SGn) einde_PPAH(SGn)		periode privelege alternatief hulpvariabele
PPA_OP(SGn, BKl) begin_PPA_OP(SGn, BKl) einde_PPA_OP(SGn, BKl)	set_PPA_OP(SGn, BKl) reset_PPA_OP(SGn, BKl)	periode privelege alternatief optie
PPB(SGn)		periode privelege bijzonder
PPB_OP(SGn) begin_PPB_OP(SGn) einde_PPB_OP(SGn)	set_PPB_OP(SGn) reset_PPB_OP(SGn)	periode privelege bijzonder optie
PPP(SGn, BKl) begin_PPP(SGn, BKl) einde_PPP(SGn, BKl)		periode privelege primair
PPP_OP(SGn, BKl) begin_PPP_OP(SGn, BKl) einde_PPP_OP(SGn, BKl)	set_PPP_OP(SGn, BKl) reset_PPP_OP(SGn, BKl)	periode privelege primair optie
	set_PRIM_toewijzing(SGn, BKl) reset_PRIM_toewijzing(SGn, NKl)	toewijzing primaire realisatie
PPS(SGn) begin_PPS(SGn) einde_PPS(SGn)		periode privelege speciaal
PPS_OP(SGn) begin_PPS_OP(SGn) einde_PPS_OP(SGn)	set_PPS_OP(SGn) reset_PPS_OP(SGn)	periode privelege speciaal optie
PPV(SGn) begin_PPV(SGn) einde_PPV(SGn)		periode privelege versneld
PPV_OP(SGn) begin_PPV_OP(SGn) einde_PPV_OP(SGn)	set_PPV_OP(SGn) reset_PPV_OP(SGn)	periode privelege versneld
PR(SGn, BKl) begin_PR(SGn, BKl) einde_PR(SGn, BKl)		primaire realisatie
R(SGn) begin_R(SGn) einde_R(SGn)		rood

referenties	instructies	toelichting
RGAH(SGn, i) begin_RGAH(SGn, i) einde_RGAH(SGn, i)		richting gevoelige aanvraag hulpvariabele
ROG(SGn) begin_ROG(SGn) einde_ROG(SGn)		recht op groen
RVAG2e(SGn)		retour voertuigafhankelijk groen
RVAG2e_OP(SGn) begin_ RVAG2e_OP(SGn) einde_ RVAG2e_OP(SGn)	set_RVAG2e_OP(SGn) reset_RVAG2e_OP(SGn)	retour voertuigafhankelijk groen optie
RVG(SGn) begin_RVG(SGn) einde_RVG(SGn)		rood voor groen
RWG(SGn)		retour wachtgroen
RWG_OP(SGn) begin_ RWG_OP(SGn) einde_ RWG_OP(SGn)	set_RWG_OP(SGn) reset_RWG_OP(SGn)	retour wachtgroen optie
RWR(SGn)		retour wachtrood
RWR_OP(SGn) begin_ RWR_OP(SGn) einde_ RWR_OP(SGn)	set_RWR_OP(SGn) reset_RWR_OP(SGn)	retour wachtrood optie
SCH(naam) begin_SCH(naam) einde_SCH(naam)		schakelaar
	schrijf_MON_UBER	schrijven in Cie C-interface
status_regelen(x)		status regelen installatie x
TIJD(naam) begin_TIJD(naam) einde_TIJD(naam) halterende_TIJD(naam) inst_TIJD(naam) waarde_TIJD(naam)	start_TIJD(naam) herstart_TIJD(naam) reset_TIJD(naam) halteer_TIJD(naam) dehalteer_TIJD(naam) wijzig_TIJD(naam, y)	tijd
tijd(2',2')	werkelijke tijd in uren en minuten	
TL(naam)	incr_TL(naam) decr_TL(naam) maak_TL(naam, z)	teller
TO_PRIM(SGn)		tegengaan overname naar primair
TMVG(SGn)		toestaan meeverlenggroen
TMVG_OP(SGn) begin_TMVG_OP(SGn) einde_TMVG_OP(SGn)	set_TMVG_OP(SGn) reset_TMVG_OP(SGn)	toestaan meeverlenggroen optie
TR(SGn) begin_TR(SGn) einde_TR(SGn)		tegenhouden realisatie
TR_OP(SGn) begin_TR_OP(SGn) einde_TR_OP(SGn)	set_TR_OP(SGn) reset_TR_OP(SGn)	tegenhouden realisatie optie
VAA(SGn)		versneld afsluiten

referenties	instructies	toelichting
begin_VAA(SGn) einde_VAA(SGn)		aanvraaggebied
VAA_OP(SGn) begin_VAA_OP(SGn) einde_VAA_OP(SGn)	set_VAA_OP(SGn) reset_VAA_OP(SGn)	versneld afsluiten aanvraaggebied optie
VAG1e(SGn) begin_VAG1e(SGn) einde_VAG1e(SGn)		eerste verlenggroen
VAG2e(SGn) begin_VAG2e(SGn) einde_VAG2e(SGn)		tweede verlenggroen
VG(SGn) begin_VG(SGn) einde_VG(SGn)		vastgroen
VG_TIJD(SGn) begin_VG_TIJD(SGn) einde_VG_TIJD(SGn) halterende_VG_TIJD(SGn) inst_VG_TIJD(SGn) waarde_VG_TIJD(SGn)	start_VG_TIJD(SGn) herstart_VG_TIJD(SGn) reset_VG_TIJD(SGn) halteer_VG_TIJD(SGn) dehalteer_VG_TIJD(SGn)	vastgroentijd
	setVlogVRIid("naam") setVlogMode(vlogmode)	opgeven naam VRI voor VLOG opgeven wijze van genereren VLOG-data
VMVG(SGn)		vasthouden meeverlenggroen
VMVG_OP(SGn) begin_VMVG_OP(SGn) einde_VMVG_OP(SGn)	set_VMVG_OP(SGn) reset_VMVG_OP(SGn)	vasthouden meeverlenggroen optie
VNM(SGn) begin_VNM(SGn) einde_VNM(SGn)		versneld naar meeverlenggroen
VNMH(SGn) begin_VNMH(SGn) einde_VNMH(SGn)		versneld naar meeverlenggroen hulpvariabele
VNM_OP(SGn, BKl) begin_VNM_OP(SGn, BKl) einde_VNM_OP(SGn, BKl)	set_VNM_OP(SGn, BKl) reset_VNM_OP(SGn, BKl)	versneld naar meeverlenggroen
voer_dump_uit()		dump aanmaken
VRVG(SGn)		vasthouden rood voor groen
VRVG_OP(SGn) begin_VRVG_OP(SGn) einde_VRVG_OP(SGn)	set_VRVG_OP(SGn) reset_VRVG_OP(SGn)	vasthouden rood voor groen optie
VVAG2e(SGn)		vasthouden (tweede) verlenggroen
VVAG2e_OP(SGn) begin_VVAG2e_OP(SGn) einde_VVAG2e_OP(SGn)	set_VVAG2e_OP(SGn) reset_VVAG2e_OP(SGn)	vasthouden (tweede) verlenggroen optie
	set_WB_toewijzing() reset_WB_toewijzing()	wachtblok

referenties	instructies	toelichting
WG(SGn) begin_WG(SGn) einde_WG(SGn)	set_WSGR_toewijzing(SGn) reset_WSGR_toewijzing(SGn)	wachtgroen
WR(SGn) begin_WR(SGn) einde_WR(SGn)		wachtrood
W_REG_VRI		verkeersregeltoestel regelt daadwerkelijk
WSRH begin_WSRH einde_WSRH		wachtstandrood hulpvariabele

Legenda:

Bij de indices staat:

i	voor een volgnummer
j	voor een algemene index
l	voor een bloknummer
n	voor een signaalgroepnummer

Verder betekent:

g.	positie in interface (0<=g<64)
h.	is de te plaatsen waarde waarvoor geldt –32.768<=h<=32.767
y.	dient een niet negatief geheel getal te zijn (Uint32) dat het tienvoud van de bedoelde te wijzigen absolute waarde representeert
z.	is een niet negatief geheel getal (Uint16)
dg	een dag van de week (MAANDAG, DINSDAG, WOENSDAG, DONDERDAG, VRIJDAG, ZATERDAG of ZONDAG)
4’	een 4 cijferig getal
2’	een 1 of 2 cijferig getal
x	een getal tussen 1 en 32767
w	dient een geheel getal te zijn (Int32) dat de bedoelde te wijzigen waarde representeert
SGn	signaalgroep n
BKl	blok l
SGn_i	detector
SGi	signaalgroep i
naam	vrij te kiezen afkorting of aanduiding onder de restrictie dat ‘naam’

- tezamen met de eventueel daaraan toe te voegen index of indeces uniek moet zijn
- niet gelijk mag zijn aan een andere binnen de RWS C-regelaar gebruikte afkorting
- niet gelijk mag zijn aan een keyword van C en
- maximaal 20 karakters mag bedragen

BIJLAGE 3: Legende van verplichte elementen die in de CRAPDEF.H moeten worden opgenomen

/* .... blokken */	nodig voor de programma's FLXRWS en POSTDUMP
/* .... detectoren */	nodig voor de programma's FLXRWS en POSTDUMP
/* .... extra parameters */	nodig voor de programma's FLXRWS en POSTDUMP
/* .... hulpfuncties */	nodig voor de programma's FLXRWS en POSTDUMP
/* .... klokken */	nodig voor de programma's FLXRWS en POSTDUMP
/* .... schakelaars */	nodig voor de programma's FLXRWS en POSTDUMP
/* .... signaalgroepen */	nodig voor de programma's FLXRWS en POSTDUMP
/* .... tellers */	nodig voor de programma's FLXRWS en POSTDUMP
/* .... tijden */	nodig voor de programma's FLXRWS en POSTDUMP
NUMBK	aantal blokken
NUMD	aantal detectoren
NUMEGGP	aantal extra geheel getal parameters
NUMEP	aantal extra parameters
NUMHF	aantal hulpfuncties
NUMKL	aantal klokken
NUMOV	aantal overige symbolische namen
NUMSCH	aantal schakelaars
NUMSG	aantal signaalgroepen
NUMT	aantal tijden
NUMTL	aantal tellers
NUM_OVI	aantal overige ingangssignalen
NUM_OVU	aantal overige uitgangssignalen

Bijlage 4: Legende van elementen die als toewijzing moeten c.q. kunnen worden gebruikt in de structures in de CRAP-TAB.C

symbool:		staat voor:
AAN		schakelaar staat in de stand aan
ALT		SG staat alternatief in het genoemde blok
AVR		detector heeft een aanvraagfunctie
BS_H5p		detector wordt gebruikt volgens de BS Hoofdstuk 5 formule p
DFA		bij fout geeft de detector een continue melding
DFU		bij fout geeft de detector geen enkele melding
DK		drukknop
DP		aanwezigheidsdetector
DS		selectieve detector
GEEN		aan de desbetreffende detector is geen van de functies AVR, BS_H5p, H1e of H2e toegewezen
	ook:	de desbetreffende HF is geen in- of uitgaand koppelsignaal van de regeling
	ook:	aan het desbetreffende koppelsignaal is geen van de functies KSU of KSH toegewezen
H1e		detector wordt gebruikt voor de 1e hiaattijd
H2e		detector wordt gebruikt voor de 2e hiaattijd
KSH		als de regelaar niet regelt dient dit uitgaande koppelsignaal door de procesbesturing gehandhaafd te worden
KSU		als de regelaar niet regelt dient dit uitgaande koppelsignaal door de procesbesturing gereset te worden
NG		niet gebruikt
PRIM		SG staat primair in het genoemde blok
STOP		geeft het einde van een struct aan
TOEPASSEN		de betreffende hulpfunctie wordt toegepast
UIT		schakelaar staat in de stand uit
WAAR		de bedoelde functie is altijd waar
WSGR		de SG is een wachtstandgroen richting

Literatuurlijst

[BS] Basisspecificatie voor verkeersregelinstallaties
Versie C2014
ing. C. Cappendijk
Rijkswaterstaat, Water Verkeer en Leefomgeving, maart 2014

[TOEL_BS] Toelichting op de basisspecificatie
Versie C2014
Rijkswaterstaat, Water Verkeer en Leefomgeving, maart 2014

[Cie_C] Beschrijving van de software interface tussen het applicatieprogramma en de procesbesturing voor verkeersregeltoestellen.
Commissie C van de CVN, otober 2011

[FLEX] Handleiding FLEXSYT-II-, deel 1 en deel 2
ir. F. Middelham
ing. T.C. Wang
Rijkswaterstaat, Dienst Verkeerskunde, januari 1992

[AANV_CIE_C] RWS Aanvulling op de Cie C interface
Tekst voor deel 3 van het RWS standaardbestek voor verkeersregelinstallaties
Maart 2014

[FLASH] Handleiding FLASH
drs. E. Vermeer
ing. T.C. Wang
Rijkswaterstaat, Adviesdienst verkeer en vervoer, november 1998