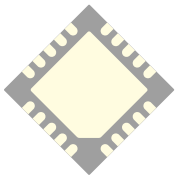
RvL techniek



# headers

## hardware-aansturing

documentnr: 499_.201

Voor hardware-aansturing raadpleegt men het gegevensblad, de uitgebreide "hardware manual". Gebruikmaking van enkel de IDE (integrated development environment) en meegeleverde hulpmiddelen krijgt men veel bronbestanden, waarin men al snel het overzicht verliest.
Dit document geeft aan hoe men vanuit het gegevensblad een header kan maken ten behoeve van compacte eigen broncode.

24 maart 2020
Robert van Lierop
rvl minus techniek at hetnet dot nl

# 1 RX65x

Voor deze beschrijving is gebruikgemaakt van een Renesas Target board for RX65N, los verkrijgbaar of in een combinatie geleverd bij de Renesas RX65N Cloud Kit.

Deze kit bestaat uit 3 printplaten:

1. het "target board"(TB), handleiding
2. het "option board"(OB), handleiding
3. de Silex PMOD Wifi-module

Het target board bevat de R5F565NEDDFP PLQP-100KB, met kenmerken:

**core** RXv2 120 MHz

**memory** 2048 kB Flash with dual-bank, 32kB data flash, 640 kB RAM

**Endian-setting** Big-endian or Little-endian

**package** LFQFP-100

**last buy** Dec 2033

Ik heb een git-repo gemaakt, Roberts-sw/RX-target-board-GCC met daarin een aantal projecten om met de chip aan de slag te gaan. Naast bovenstaande internet-koppelingen is het ook van belang om het uitgebreide gegevensblad, het User's Manual: Hardware (HW) binnen bereik te hebben.

Projecten om zelf mee aan de slag te gaan en iets te leren zijn het meest overzichtelijk indien ze klein zijn, en daarvoor is het handig als de eventuele download's ook klein zijn.

Het allereerste project is gemaakt met behulp van de IDE en de meegeleverde "smart configurator", waardoor met aanklikken van enkele zaken er iets gemaakt kan worden wat compileert en de microcontroller daadwerkelijk iets laat doen, maar dat leidt niet tot veel inzicht in benodigdheden.

Reden is dat het project wordt volgepropt met allerlei "hardware abstraction"-bestanden en de ene routine de andere aanroept om maar zoveel mogelijk inzicht in de werking van de controller weg te halen.

Door rustig met de debugger de stappen af te lopen tot aan de 'main'-routine en alleen de code die uitgevoerd wordt mee te nemen, kan men het geheel van een simpel programma in (nagenoeg) één bestand zetten om dat vervolgens nogemaals te laden en met de debugger te doorlopen.

Na bovenstaande stap heb ik wat documentatie gelezen, met name HW, en zoveel mogelijk zaken verwijderd of met een andere opzet getest om erachter te komen wat minimaal nodig is. Daaruit zijn enkele demo-projecten met andere instellingen van de controller-klok ontstaan.

Een .tgz-bestand van een dergelijk project is nog ruim boven de 60 kB, terwijl er amper broncode geschreven is. Nadere analyse leert dat het door Renesas geleverde bestand iodefine.h ruim 32.000 regels en 683 kB omvat voor de hardware-registers zowel Big-endian als Little-endian.

Teneinde verder inzicht te verschaffen in de werking wil ik de afhankelijkheid van het grote bestand elimineren. Omdat daarmee zeer veel tijd gemoeid is, neem ik telkens delen uit het gegevensblad HW die nodig zijn voor een project-onderdeel, en zet die dan in een header.

Deze header, rx65x.h, zal dan mogelijk in een project het bestand iodefine.h vervangen.

## 1.1 chip-header

Het headerbestand poogt met een kleine omvang de tegengekomen registers te benoemen in structuren die Endian-onafhankelijk zijn

Om Endian-onfhankelijk te zijn, schrap ik de registerbits in de header, en voeg voor elke struct

in commentaar verwijzing naar hoofdstuk(ken) in het “hardware manual” HW toe.
De opbouw wordt toegelicht aan de hand van een van de eerste definities:

```
    /* ------------------------------------------------------------
    system
        HW 3. Operating Modes
        HW 6. Resets
        HW 8. Voltage Detection Circuit (LVDA)
        HW 9. Clock Generation Circuit
        HW 11. Low Power Consumption
        HW 13. Register Write Protection Function
    ------------------------------------------------------------ */
#define SYSTEM_ (*(struct {\
/*0000*/u16 MDMONR,_0002,_0004,SYSCR0,SYSCR1,_000a,SBYCR,_000e;\
/*0010*/u32 MSTPCRA,MSTPCRB,MSPCRC,MSTPCRD;\
/*0020*/u32 SCKCR; u16 SCKCR2,SCKCR3,PLLCR; u08 PLLCR2; _(002b,5)\
/*0030*/u08 BCKCR,_0031,MOSCCR,SOSCR,LOCOCR,ILOCOCR,HOCOCR,HOCOCR2;\
    _(0038,4) u08 OSCOVFSR,_003d,_003e,_003f;\
/*0040*/u08 OSTDCR,OSTDSR; _(0042,5e)\
/*00a0*/u08 OPCCR,RSTCKCR,MOSCWTCR,SOSCWTCR; _(00a4,1c)\
/*00c0*/u08 RSTSR2,_00c1; u16 SWRR; _(00c4,1c)\
/*00e0*/u08 LVD1CR1,LVD1SR,LVD2CR1,LVD2SR; _(00e4,31a) u16 PRCR; _(0400,c1c)\
/*101c*/u08 ROMWT; _(101d,b263)\
/*c280*/u08 DPSBYCR,_c281,DPSIER0,DPSIER1,DPSIER2,DPSIER3,DPSIFR0,DPSIFR1,\
    DPSIFR2,DPSIFR3,DPSIEGR0,DPSIEGR1,DPSIEGR2,DPSIEGR3,_c28e,_c28f;\
/*c290*/u08 RSTSR0,RSTSR1,_c292,MOFCR,HOCOPCR,_c295,_c296,LVCMPCR,\
    LVDLVLR,_c299,LVD1CR0,LVD2CR0; _(c29c,4)\
/*c2a0*/u08 DPSBKR[32];\
} volatile *const)0x00080000)
```

De onderdelen zijn:

- /* HW 6. Resets */ HW-verwijzing naar hoofdstuk 6 voor meer informatie
- #define SYSTEM_ om los te staan van de iodefine-definitie `SYSTEM`.
- /* 0000 */ intern commentaar om het laatste adresdeel aan te duiden
- u16 echte “fixed-width” data types, gekoppeld aan die van <stdint.h>
- MDMONR registernaam als in HW en iodefine.h, zonder `.BIT`, `.WORD` etcetera.
- opvulling op 2 manieren:
  1. ,_0002 individueel aangeduid met het hex-einde van het beginadres
  2. ; _(00a4,1c) gebied als met macro(hex-einde van het beginadres,hex-bytegrootte). Gebiedsaanduiding altijd na afsluiting van het vorige element, deze vormt zelf ook afsluiting, zodat ervoor en erna data types nodig zijn!
- regeleinde voorafgegaan door \ om de struct voort te zetten met een extra regel.
- volatile om aan te geven dat de registerinhoud hardwaregestuurd is
- *const om aan te geven dat het adres vastligt
- 0x00080000 beginadres van de structuur