

# PROCESSVERSLAG MODULEVORMING POTENTIOMETER + ROTARY ENCODER VOOR DE PIC VERSTERKER

Wessel Tip <contact@wessel.gg> (Student nummer 696770, <https://wessel.gg/>)

Technische Informatica op het InHolland Alkmaar (Jaar 2, Semester 1, Sept. 2023 - Feb. 2024)

## Samenvatting

In dit verslag zal het proces beschreven worden dat genomen is tijdens het maken van de specialisatie voor de versterker van het project voor Embedded Systems.

Als specialisatie is er voor gekozen om een aparte module voor de rotary encoder (inputselectie) en potentiometer (volumeselectie) te maken.

Ook zal er besproken worden welke problemen er zijn opgetreden en hoe deze zijn opgelost.

## INHOUDSOPGAVE

<b>1</b>	<b>Introductie</b>	<b>2</b>
<b>2</b>	<b>Correcties na 1<sup>e</sup> Posing</b>	<b>2</b>
2.1	PIC Versie . . . . .	2
2.2	Header Files . . . . .	2
2.3	Uitlezen Van De Rotary Op De PIC16F1829 . . . . .	2
2.4	PORTB moet naar PORTB gezet worden, anders bevriest de 887 . . . . .	3
2.5	Interrupts op de 1829 . . . . .	3
2.6	Memory Problemen . . . . .	3
<b>3</b>	<b>Softwarestappen</b>	<b>4</b>

# 1 INTRODUCTIE

In dit verslag zal mijn proces beschreven worden tijdens het maken van de specialisatie voor de versterker van het project voor Embedded Systems.

Als specialisatie is er voor gekozen om een aparte module voor de rotary encoder (inputselectie) en potentiometer (volumeselectie) te maken. De modules moeten ieder hun eigen initialisatiefuncties afhandelen, en moeten samen kunnen werken.

De rotary encoder module moet na initialisatie een functie meegeven die aangeroepen kan worden wanneer de rotary encoder een stap naar links of rechts maakt. Deze moet vervolgens een door de gebruiker gegeven waarde verhogen of verlagen. Ook moet de module werken met een interrupt, zodat de microcontroller niet constant hoeft te luisteren naar de encoder.

## 2 PROJECT

Voor dit project hebben wij de onderdelen in drieën verdeeld:

<b>Burhan</b>	Een library maken voor het bedienen van de LED matrices
<b>Robin</b>	Een library maken voor het bedienen van de infrarood
<b>Wessel</b>	Een library maken voor het bedienen van de rotary encoder en potentiometer

## 3 CORRECTIES NA 1<sup>e</sup> POGING

Dit verslag is gemaakt nadat de 1<sup>e</sup> toetsing niet goed verliep. Op dat moment bleken de manieren die ik had gebruikt om aan te duiden welke PIC versie gebruikt werd niet goed te werken. Ook verliep het draaien van de rotary encoder en potentiometer niet zoals verwacht.

### 3.1 PIC VERSIE

Om de pic versie aan te duiden maakte ik eerst gebruik van een `IFDEF` statement. Dit bleek later echter incorrect te zijn, omdat de compiler bij het compilen van de library dan een van de `ifdefs` uitkiest om mee te nemen.

Om dit op te lossen heb ik uiteindelijk geen gebruik gemaakt van de XC8 library registers, maar de adressen vanuit de datasheet (pagina 25-28 voor de PIC16F887A, en pagina 28-31 voor de PIC16F1829) om gegevens in de registers te schrijven/uit te lezen.

### 3.2 HEADER FILES

In de 1<sup>e</sup> versie van deze library had ik steeds problemen met het includen van de correcte header files. Dit kwam omdat de header files van de libraries aparte bestanden waren dan diegenen van het project. Dit heb ik opgelost door in MPLAB een albestaand bestand te importeren uit de "global" folder, en deze vervolgens te includen door middel van `"../global/naam.h"`.

### 3.3 UITLEZEN VAN DE ROTARY OP DE PIC16F1829

De functie die eerst gebruikt werd voor de rotary bleek niet te werken op de 1829. Daarom heb ik deze functie herschreven, en geen gebruik gemaakt van optimaliserende operaties zoals `bitshiften`. De nieuwe functie kan gezien worden in [Source Code 1](#).

**Source Code 1:** De functie om de rotary encoder op beide PICs uit te lezen.

```
1 int read_rotary(void) {
2     // Only check for a change in the CLK pin to identify "full click"
3     int a = GetReg(&encoder.pinA);
4
5     // If state changed, a movement is registered
6     if (a != encoder.LastState) {
7         int b = GetReg(&encoder.pinB);
```

```

8
9     if (b != a) encoder.Steps++; // Clockwise
10    else encoder.Steps--; // Anti-clockwise
11
12    // Only change counter if correct amount of clicks have been registered
13    if (encoder.Steps > (encoder.stepReq * 2)) {
14        encoder.Input++;
15        encoder.Steps = 0;
16    } else if (encoder.Steps < -(encoder.stepReq * 2)) {
17        encoder.Input--;
18        encoder.Steps = 0;
19    }
20
21    // Adjust counter for boundaries
22    if (encoder.Input < encoder.min) encoder.Input = encoder.max;
23    else if (encoder.Input > encoder.max) encoder.Input = encoder.min;
24
25    encoder.LastState = a;
26 }
27
28 if (encoder.host == PIC16F887A) SetReg(&encoder.encoderRail, GetReg(&encoder.encoderRail));
29
30 return encoder.Input;
31 }

```

### 3.4 PORTB MOET NAAR PORTB GEZET WORDEN, ANDERS BEVRIEST DE 887

De interrupt van de rotary encoder stopte de program loop op de 887 zodra hij een keer werd aangeroepen. De exacte oorzaak hiervan heb ik nog niet kunnen vinden, echter werkte het wel om PORTB naar zichzelf te schrijven.

### 3.5 INTERRUPTS OP DE 1829

De 1<sup>e</sup> versie van de library maakte geen gebruik van interrupts op de 1829. Hierdoor moest beide de rotary encoder en potentiometer uitgelezen worden in de while loop, wat niet de bedoeling was.

Om dit op te lossen is er uiteindelijk gebruik gemaakt van de interrupt on change PORTA.

### 3.6 MEMORY PROBLEMEN

Op eerste instantie had ik Robin benadert hoe hij het had gedaan met zijn library, hij had een `Reg` struct gebruikt met de pin adressen om dit op te lossen. Een voorbeeld van deze structs kan gezien worden in [Source Code 2](#). Echter bleek veel meer memory in gebruik te nemen dan dat de PIC16F1829 heeft.

Als eerste oplossing had ik de configuratiebits die naast elkaar staan te combineren, dit scheelde echter niet genoeg. Uiteindelijk heb ik besloten om de configuratiebits in te stellen zonder `Reg` struct door middel van de functie in [Source Code 3](#). Voor de velden die vaker gebruikt moeten worden, zoals de GOnDONE bit en PORTB register heb ik echter wel gebruik gemaakt van de `Reg` structs om eventuele typfouten te voorkomen.

De uiteindelijke code die gebruikt is om deze registers in te stellen/uit te lezen kunnen gezien worden in [Source Code 4](#).

**Source Code 2:** Initialisatiestructs van de rotary encoder op de PIC16F887A.

```

1 ROT = (ROTPins) {
2     .RBIEINTE = { (unsigned char*) 0x00B, 2, 3 },
3     .RBIE     = { (unsigned char*) 0x00B, 1, 3 },
4     .INTE     = { (unsigned char*) 0x00B, 1, 4 },
5     .GIEPEIE  = { (unsigned char*) 0x00B, 2, 6 },
6     .PEIE     = { (unsigned char*) 0x00B, 1, 6 },
7     .GIE      = { (unsigned char*) 0x00B, 1, 7 },
8     .IOCBA    = { (unsigned char*) 0x096, 1, pinA }, // 0x0D

```

```

9   .IOCBB = { (unsigned char*) 0x096, 1, pinB },
10  .INTEDG = { (unsigned char*) 0x081, 1, 6 },
11  .PORTB = { (unsigned char*) 0x106, 8, 0 },
12  .INTF = { (unsigned char*) 0x00B, 1, 1 },
13  .RBIF = { (unsigned char*) 0x00B, 1, 0 }
14 };
15
16 SetReg(&ROT.INTF, 0);
17 SetReg(&ROT.RBIF, 0);
18
19 SetReg(&ROT.GIEPEIE, 0b11);
20 SetReg(&ROT.RBIEINTE, 0b11);
21 SetReg(&ROT.GIE, 1);
22 SetReg(&ROT.INTE, 1);
23 SetReg(&ROT.PEIE, 1);
24 SetReg(&ROT.RBIE, 1);
25 SetReg(&ROT.IOCBA, 1);
26 SetReg(&ROT.IOCBB, 1);
27 SetReg(&ROT.INTEDG, 0);

```

Source Code 3: Initialisatiefunctie voor het schrijven naar registers zonder structs.

```

1  unsigned char SetRawReg(unsigned char* address, unsigned char size,
2      unsigned char offset, unsigned char V) {
3      unsigned char bitMask;
4      V = (unsigned char) (V << offset);
5      bitMask = (unsigned char) (((1 << size) - 1) << offset);
6      *(address) &= ~bitMask;
7      *(address) |= V & bitMask;
8  }

```

Source Code 4: Initialisatie van de register structs.

```

1  typedef struct _Reg {
2      unsigned char* address;
3      unsigned char size;
4      unsigned char offset;
5  } Reg;
6
7  void SetReg(Reg* R, unsigned char V) {
8      unsigned char bitMask;
9      V = (unsigned char) (V << R->offset);
10     bitMask = (unsigned char) (((1 << R->size) - 1) << R->offset);
11     *(R->address) &= ~bitMask;
12     *(R->address) |= V & bitMask;
13 }
14
15 unsigned char GetReg(Reg* R) {
16     unsigned char V, bitMask;
17     bitMask = (unsigned char) (((1 << R->size) - 1) << R->offset);
18     V = *(R->address) & bitMask;
19     V = V >> R->offset;
20     return V;
21 }

```

## 4 SOFTWARESTAPPEN

### 1 ONTWIKKELING VOOR MODULEVORMING

Als eerste stap is er code geschreven voor de potentiometer met ADC en de rotary encoder met interrupts. Vervolgens is deze code toegevoegd om een main file te vormen, hierdoor kon er getest worden hoe deze twee modules samen werken en welke specifieke configuratiebits nodig zijn.

#### 1.1 PROBLEMEN

Tijdens het testen van deze code bleek de interrupt in eerste instantie niet te werken, dit kwam doordat de "Change on PORTB" interrupt gebruikt was, en deze per pin ingesteld moet worden door middel van de **REGISTER 3-8: IOCB: INTERRUPT-ON-CHANGE PORTB REGISTER[1]** configuratiebits. Nadat dit ingesteld was, werkte de interrupt zoals geplant.

In eerste instantie was het idee dat de **REA** en **REB** pins van de rotary encoder in de initialisatiefunctie meegegeven zouden worden. Dit bleek echter niet te werken, omdat wanneer deze waarden meegegeven worden, alleen hun huidige staat meegegeven wordt, en dus niet verandert wanneer de rotary encoder draait. Daarom worden deze nu meegegeven met de `parse_rotary` functie.

### 2 OPSPLITSING

Nadat bevestigd was dat de twee modules samen werkten, is de code opgesplitst in twee aparte bestanden. Hierdoor is het mogelijk om de code voor de potentiometer en de rotary encoder apart te compileren. De configuratiebits zijn eerst wel nog centraal gehouden door middel van aparte initialisatie functies. Deze functies geven hun waarden aan met bepaalde structs. [Source Code 5](#)

**Source Code 5:** Initialisatie van de interrupt structs

```
1 | InterruptConfig int_config = { GIE_ENABLED, EINT_ENABLED, PEIE_ENABLED,  
2 |                               TOINT_ENABLED, RBINT_ENABLED, FALLING_EDGE };  
3 | init_int(int_config);
```

### 3 MODULEVORMING

Nadat alles getest en werkend gemaakt is wordt de code in aparte modules in MPLAB gestopt, vervolgens wordt hier ook de code voor elke corresponderende configuratiebit toegevoegd in een initialisatiefunctie.

### 4 COMPATIBILITEIT

Alle code is nu geschreven voor de PIC16F887A. Dit wordt nu opnieuw gedaan voor de PIC16F1829. Een van de eisen is dat de module werkt op beide PIC controllers met een binary. Daarom is er gebruik gemaakt van de register adressen om de code te compileren die nodig is voor de PIC16F887A of de PIC16F1829.

### 5 OPTIMALISATIE

De code is nu werkend op beide controllers, maar er is nog ruimte voor optimalisatie. Zo wordt er een plus en keerberekening gebruikt om de waarden van de potentiometer samen te voegen, echter kan dit ook al met een bitshift en bitwise OR operatie. De verandering kan gezien worden in [Source Code 6](#).

**Source Code 6:** Optimalisatie van de potentiometer code

```
1 | // Oude code  
2 | (ADRESH * 256) + ADRESL  
3 | // Nieuwe code  
4 | (ADRESH << 8 | ADRESL);
```

## REFERENCES

- [1] M. T. Inc., *PIC16F882/883/884/886/887 Data Sheet*, 2009, accessed: 06-03-2024. [Online]. Available: <http://ww1.microchip.com/downloads/en/devicedoc/41291f.pdf>