

PROG3 – Practicum Week3

Datum: 12-09-2013
Versie: 2.0
Auteur: Sander van Heumen

Intro Computer Architecture
Embedded C – Introduction 8051

Inleiding

- Lever de opdrachten van vorige week in, en stel vragen bij onduidelijkheden.
- Installeer de “**Keil C51 Development tools for classic and extended 8051 microcontrollers**”, zie [bijlage 1](#). Je hebt deze tools nodig om de ADuC847 te kunnen programmeren. Hoe moet je de ADuC847 programmeren en debuggen? Zie [bijlage 2](#). **Voer de instructies in bijlage 2 uit!**

1) Embedded C 8051 - Looplichtje

Om vertrouwt te raken met de hardware, ga je eerst een voorbeeld project bekijken. Dit voorbeeld project kun je vinden op Blackboard: [demo_8051.zip](#).

Dit voorbeeld project beschikt over handige timer functies. Het voorbeeld project **timer0** bevat een 4-tal bestanden:

- 1) `ctimer.h` // Hierin vind je een aantal functiedefinities om timer0 makkelijk in te stellen.
- 2) `ctimer.c` // De functieprototypes van de functiedefinities in `ctimer.c`
- 3) `ADuC847.h` // Hierin vind je verwijzingen naar o.a. on-chip peripherals
- 4) `main.c` // De `main()` –en initialisatieroutines.

Je hoeft je voorlopig alleen te concentreren op de `main.c`. In de `main()` (van het demo project) staan een aantal functie aanroepen die een timer, in dit geval timer0, op een eenvoudiger manier configureert. Hieronder is de `main()` weergegeven:

```
void main()
{
    init();
    initTimer0( 40000 );
    enableTimer0();

    for( ;; );
}
```

Hopelijk zie je veel overeenkomsten met de console applicaties die je tot nu toe hebt gemaakt. Deze `main()` draait echter in de zwarte-chip op de achterzijde van het microcontrollerbordje. Een belangrijk verschil!! Deze `main()` zal daarom nooit eindigen, vandaar de oneindige `for()` lus op het einde. Uiteraard had je hier ook `while(1);` of iets soortgelijks mogen schrijven. Programma's op de PC dienen wel te eindigen. Genoeg, we kijken nu naar de eerste functieaanroep:

- `init();`

Deze initialiseert de microcontroller. De kloksnelheid wordt hier ingesteld, de IO-poorten worden juist geconfigureerd etc. Al deze instellingen komen indirect uit de datasheet van de microcontroller, de ADuC847. Je hoeft echt niet allemaal te snappen wat hier staat, alleen dat in deze functie de controller *startklaar* wordt gemaakt. Op de volgende regel zien we de volgende functieaanroep:

- `initTimer0(40000);`

Deze functieaanroep, initialiseert de timer met een waarde 40000. Deze waarde 40000 komt overeen met een bepaalde tijdsduur, en wel $40000 * 50\mu s = 2$ seconden. Wanneer nu timer 0 wordt aangezet, zal telkens om de 2 seconden de functie “`void T0_overflow(void)`” automatisch worden uitgevoerd / aangeroepen. Je kan dit dus zien als een timer-event/interrupt welke om de 2 seconden optreedt.

Uiteraard dien je de timer wel nog aan te zetten, anders gebeurt er helemaal niets:

- `enableTimer0();`

De timer staat nu aan, en de routine “`void T0_overflow(void)`” wordt in dit geval om de 2 seconden uitgevoerd. *Wat doet de `main()` nu nog?*

De timer zet je uit door:

- `disableTimer0();`

aan te roepen.

Om een beetje gevoel te krijgen van wat er allemaal gebeurt in de microcontroller, dien je in de lege functie “`void T0_overflow(void)`”, welke onder de `main()` staat, een looplicht te programmeren. Het looplicht maak je op de LEDS zichtbaar door te schrijven naar P0.

```
void T0_overflow(void)
{
    P0 = 0xFF; // zet ALLE 8 LEDjes aan
}
```

Experimenteer met de waarde die je doorgeeft aan `initTimer0()`, en zie daar, de snelheid van het looplicht veranderd.

2) Embedded C 8051 – Disco NightFever

Ontwerp (outlining of activity diagram) en schrijf nu een programma welke de LEDS (aangesloten op P0) op de volgende wijzen laat knipperen:

LED 0	Om de 100ms
LED 1	Om de 200ms
LED 2	Om de 400ms
LED 3	Om de 800ms
LED 4	Om de 1.0 seconden
LED 5	Om de 2 seconden
LED 6	Om de 5 seconden
LED 7	Om de 10 seconden

Je kan hiervoor het demo-project van de vorige opgaven gebruiken als basis.

3) Co-op scheduler - Theorie

Tijdens het hoorcollege is het je hopelijk duidelijk geworden, dat wanneer we betrouwbare software voor Embedded systemen willen ontwikkelen, we gebruik *kunnen* maken van software gebaseerd op **Time-Triggered Architectures**. De grootste tegenhanger van dergelijke architecturen zijn de zogenaamde **Event-Triggered Architecturen**. Ook met deze architecturen kun je fatsoenlijke en betrouwbare software maken, alleen is hiervoor veelal meer kennis nodig: “ze zijn complexer”. Wij gaan deze week een **co-operative scheduler gebruiken**, wat een voorbeeld is van een Time-Triggered Architecture. Het draait tijdens dit vak om het ontwerpen van **software** (voor Embedded systemen) op een hoog niveau. De details, de instellingen van de gebruikte microcontroller (op het laagste niveau: denk hierbij aan het instellen van SFR etc.) vallen buiten de strekking van dit vak. Maak je vooral niet druk als je niet weet wat er precies allemaal gebeurt, dat hoeft ook niet (mag wel).

Hieronder vind je een aantal theorievragen met betrekking tot de besproken software architecturen, probeer deze met behulp van het hoorcollege en bijbehorende PowerPoint te beantwoorden.

- Wat bedoelen we met een Embedded systeem? En geef hiervan een aantal voorbeelden.
- Wat zijn de belangrijkste verschillen tussen de hardware van een moderne desktop pc en een ‘simpel’ Embedded systeem? *Door deze verschillen moet je namelijk ook anders software gaan ontwikkelen.*
- Wat geven we met de term MIPS aan? En geef hiervan een voorbeeld. Denk hierbij aan klokfrequentie, en het aantal instructies per clock-cycle.
- Een hoge programmeertaal staat verder af van de hardware; waarom spreken we dan toch van een hogere programmeertaal?
- Jantje zegt: “Multitasking op simpele Embedded systemen bestaat niet, althans niet wanneer multitasking betekend dat er instructies parallel worden uitgevoerd”. Leg uit waarom deze stelling wel / niet klopt.
- Wat is het verschil tussen een Time-Triggered Architecture en een Event-driven architecture?
- Wat bedoelen we met *deterministic processing*?
- De grote kracht van een co-operative scheduler is dat altijd slechts één taak actief is, en dat deze niet kan worden onderbroken. *Hoe noemen we dit verschijnsel?*
- Wanneer we een co-operative scheduler gebruiken moeten we er zorg voor dragen dat de worst-case executie tijd van een taak niet groter is dan de *tick-interval* van de scheduler. Leg uit waarom?
- Wat bedoelen we met ‘een taak’(task) in een co-operative scheduler?

4) Functie-pointers

In de volgende opdracht gaan we een **co-operative scheduler** bouwen. Deze kun je gebruiken in je software projecten (denk aan OIP, minoren etc.) om software gestructureerd op te bouwen. De basis van deze co-operative scheduler is weergegeven op de laatste pagina van het hoorcollege. Hierin staat dat er gebruikt wordt gemaakt van zogenaamde **functie-pointers**: *pointers naar functies*. Ook hiervan hoeft je op dit moment niet veel kennis van te hebben, je moet ze enkel kunnen gebruiken.

- Zoek op internet op wat functie-pointers zijn en geef hiervan een duidelijk voorbeeld in C.

Bij de co-operative scheduler op de volgende pagina maken we enkel gebruik van een functie-pointer die geen argumenten accepteert, en geen argument retourneert. In de volksmond noemen wij dat een: “void-void-functiepointer”. Deze heeft de volgende vorm:

```
void (*fp) (void); // Declareert een functiePointer met de naam fp die
                  // wijst naar een functie met geen argumenten, en
                  // geen return argument
```

5) Co-operative scheduler

De co-operative scheduler die wij gaan gebruiken heeft de volgende vorm:

```
/* ..... */
/* ..... */

void main(void)
{
    // Setup microcontroller
    PLLCON = 0x00;
    P0 = 0x00;

    // Set up the scheduler
    SCH_Init_T2();

    // Prepare for the 'Flash_LED' task
    LED5_Flash_Init();

    // Add the 'Flash LED' task (on for ~1000 ms, off for ~1000 ms)
    // - timings are in ticks (~1 ms tick interval)
    // (Max interval / delay is 65535 ticks)
    SCH_Add_Task(LED5_Flash_Update, 0, 1000);

    // Start the scheduler
    SCH_Start();

    while(1) {
        SCH_Dispatch_Tasks(); // executes the tasks!
    }
}
```

We kunnen hierin een aantal stappen onderscheiden:

- **Het initialiseren van de microcontroller** (o.a. het correct instellen van de klokfrequentie van de CPU).
- **Het initialiseren van de scheduler** (o.a. het configureren van de hardware timer zodat deze timer ticks genereert van precies 1 milliseconde)
- **Het initialiseren van een 'user-defined-task'**. Dit kan een **periodieke taak** zijn, maar ook een taak die maar 1 keer uitgevoerd hoeft te worden (**one-shot-task**). De taak hierboven noemen wij `LED5_Flash_update()` welke simpelweg een bepaalde led laat knipperen. Initialisatie van deze "`LED5_Flash_update()`" is niet zo spectaculair, maar je kunt je voorstellen dat wanneer je een taak aanmaakt die een PID regeling implementeert, de initialisatie van de taak een stuk complexer zal zijn.
- **Het toevoegen van taken aan de scheduler**. De scheduler zal dan deze taken netjes op het juiste moment uitvoeren. In het voorbeeld wordt er maar één taak toegevoegd aan de scheduler, namelijk de functie "`LED5_Flash_update()`".

Wij als eindgebruiker van deze scheduler hoeven niet te weten hoe de scheduler intern werkt, als die maar werkt. Uiteraard willen we wel zelf taken maken, en deze door de scheduler op de juiste momenten laten uitvoeren. Hierbij moet je een aantal zaken goed in de gaten houden:

- Een taak is een functie die de volgende vorm heeft: `void mijnTaak(void);`
Een dergelijke taak/functie kun je aan de co-op-scheduler toevoegen met behulp van de functie:

```
SCH_Add_Task(mijnTaak, 0, 1000);
```

Uiteraard dien je dus eerst een taak aan te maken, dat betekent dat je een functie-prototype + definitie moet maken van de 'taak' die je wilt laten uitvoeren. Het tweede argument, 0, hierboven geeft aan met welke vertraging de taak moet worden gestart vanaf het moment dat het systeem operationeel is.



Laat geen taak op hetzelfde moment beginnen, maar gebruik een offset van ~5msec. Een voorbeeld:

```
SCH_Add_Task(mijnTaak, 0, 1000);  
SCH_Add_Task(mijnTaak2, 5, 2000);
```

Het 3^{de} argument, 1000, hierboven geeft aan om de hoeveel timer-ticks de taak moet worden uitgevoerd. De timer-ticks in het voorbeeld project worden om de milliseconde gegenereerd, dus dat is lekker makkelijk! Een waarde van 1000 betekent: "voer deze taak elke seconde uit."

Maak nu de opdracht van vorige week, "Embedded C Disco Night Fever", met behulp van deze **co-operative scheduler**. Het project dat je hiervoor als basis moet gebruiken vind je op Blackboard:

[co_op_scheduler_uv4.zip](#)

Bijlage 1 – Downloaden & installeren Keil IDE t.b.v programmeren ADuC847 Microcontroller

Ten behoeve van het programmeren van de ADuC847 dienen jullie een evaluatie versie te downloaden genaamd “Keil C51 Development tools for classic and extended 8051 microcontrollers”. **Let op** wanneer je deze IDE download zal je je moeten registreren...

- <https://www.keil.com/demo/eval/c51.htm>



Installatie-instructies

- Dubbelklik op C51vXXX.exe (de XXX stellen hier 3 cijfers voor)
- Klik op Next
- Vink “*I agree to all the terms of the preceding License Agreement*” aan. Klik op Next.
- Kies een installatie-directory.
Kies altijd een pad ZONDER spaties om problemen te voorkomen.
Aanbevolen pad: **C:\Keil**
Backup-up eventuele vorige geïnstalleerde versies.
- *Customer Information*
Vul je gegevens in, klik op Next.
- De installatie zal nu starten. Wacht tot de installatie is voltooid. Klik dan op Finish.
- Er staat nu een icoontje op het bureaublad genaamd “Keil uVision4”, klik hier dubbel om de ontwikkelomgeving van Keil te starten.

Bijlage 2 – Programmeren en Debuggen van de ADuC847 Microcontroller gebruikmakend van de KEIL uVision4 ontwikkelomgeving.

Deze bijlage bestaat uit de volgende onderdelen:

- Aansluiten ADuC847 + installeren FTDI D2xx driver t.b.v “virtuele COM-poort”
- Project aanmaken in Keil uVision4
- Programmeren van project in ADuC847
- Debuggen van project op ADuC847

Aansluiten ADuC847 + installeren FTDI D2xx driver t.b.v “virtuele COM-poort”

Indien je het ADuC847-kitje in het verleden al vaker hebt gebruikt op je laptop en deze nog steeds dezelfde Windows-installatie bevat, dan kun je deze stap overslaan en mag je verder gaan met lezen op “Project aanmaken in Keil uVision4”

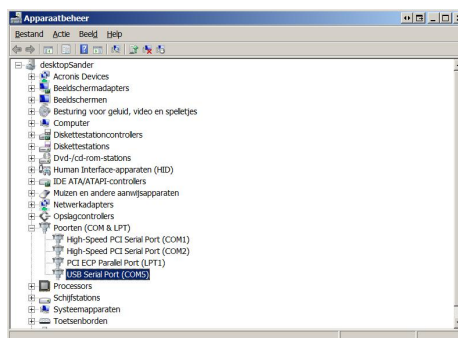
Sluit nu het ADuC847 kitje aan op een willekeurige USB –poort van je laptop. Indien je het ADuC847-kitje voor de eerste keer aansluit, zul je drivers moeten installeren t.b.v. de communicatie tussen PC en ADuC847 microcontroller. Dit zijn de **D2xx virtuele COMpoort drivers van FTDI**. Deze kun je van de volgende website downloaden:

- <http://www.ftdichip.com/Drivers/D2XX.htm>

Ik ga ervan uit dat je weet hoe je een driver moet installeren. Lukt het je niet? Raadpleeg dan de desbetreffende docent tijdens het practicum.

Indien de driver correct is geïnstalleerd en het ADuC847-kitje is aangesloten op een USB-poort, zal het systeem zijn uitgebreid met een virtuele seriële poort. Dit kun je controleren in **Configuratiescherm/Systeem/Apparaatbeheer/Poorten(COM & LPT)**. Een voorbeeld hiervan is weergegeven in Figuur 1. Onthoudt het COM-poort nummer, deze moet je namelijk in Keil uVision instellen. In het plaatje hieronder is het COM-poort nummer “vijf” -> “COM5”.

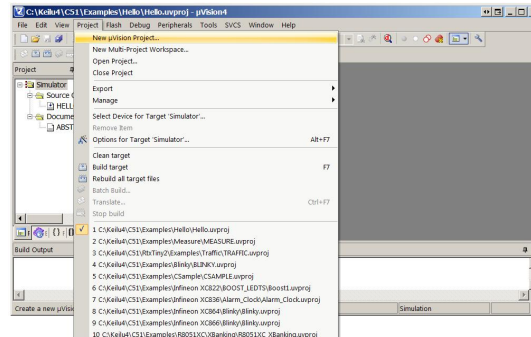
TIP: Door het ADuC847-kitje telkens op dezelfde USB-poort aan te sluiten zorg je ervoor dat deze ook telkens hetzelfde COM-poort nummer krijgt toegewezen waardoor je dit in uVision niet telkens opnieuw hoeft in te stellen. Handig!



Figuur 1 - Apparaatbeheer COMpoort

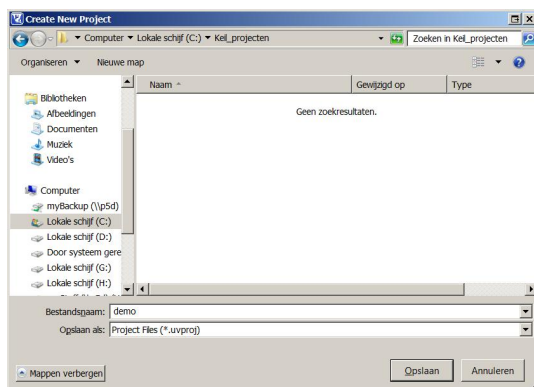
Project aanmaken in Keil uVision4

1. Start Keil uVision4, zorg ervoor dat de ADuC847 is aangesloten op je laptop en de driver hiervoor correct is geïnstalleerd (zie vorige pagina).
2. Klik boven in de menubalk op **Project/New uVision Project...** Zie Figuur 2.

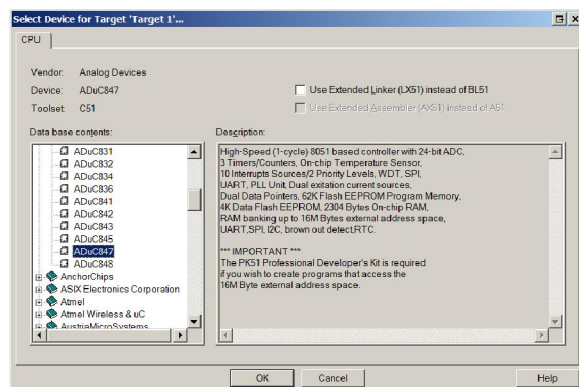


Figuur 2 – New uVision Project

3. Kies een geschikte locatie om je project op te slaan. In het voorbeeld is gekozen voor de map “C:\Keil_projecten” en projectnaam “demo”. Klik op **Opslaan**. Zie Figuur 3.
4. “Select Device for Target ‘Target 1’...”. Zie Figuur 4.
Selecteer **Analog Devices**, en dan **ADuC847**. Klik op OK. Beantwoord de vraag: “Copy Analog Devices Startup Code to Project Folder and Add File to Project ?” met Ja.

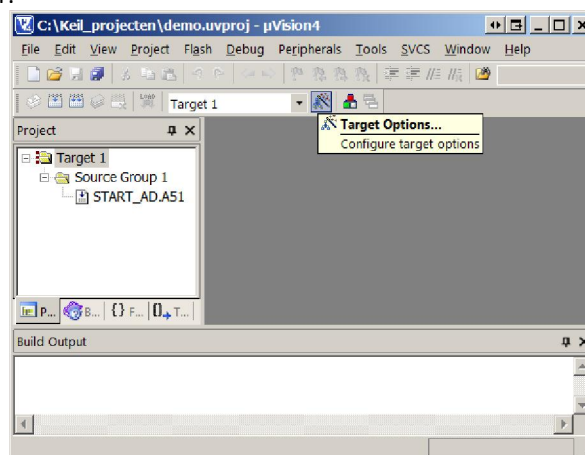


Figuur 3 – Project map

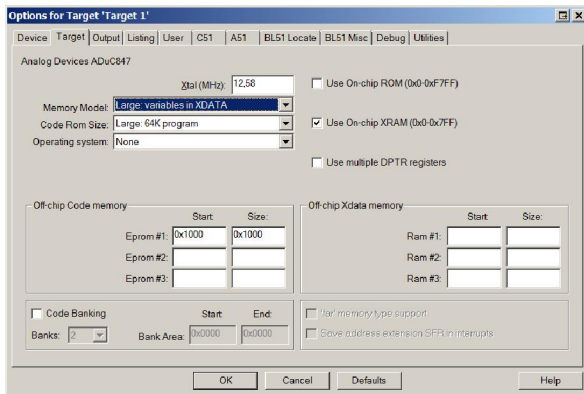


Figuur 4 - Select Device target

5. Klik nu op het **toverstafje** om de **Target Options** te openen. Zie Figuur 5 . Neem nu de instellingen op de volgende pagina's over:

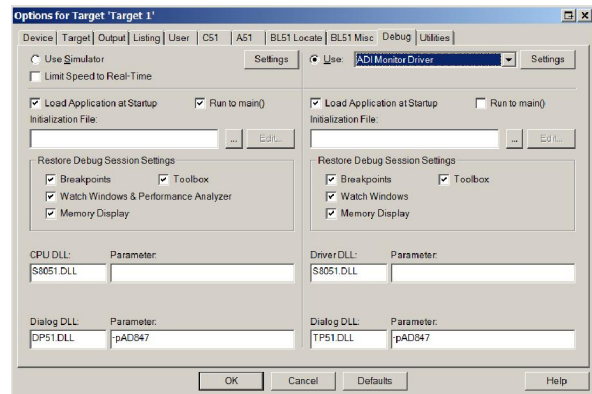


Figuur 5 – Configure Target Options

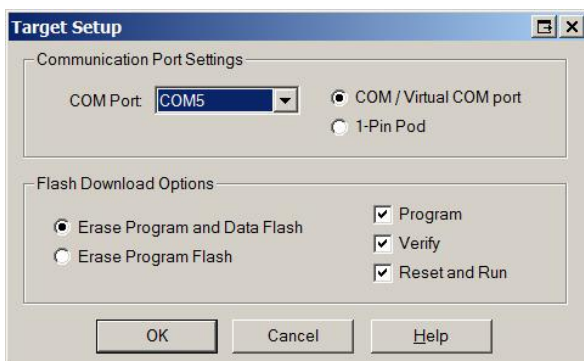


Figuur 6 - Tabblad TARGET

Let op: typ bij Xtal (MHz) 12 PUNT 58 MHz.

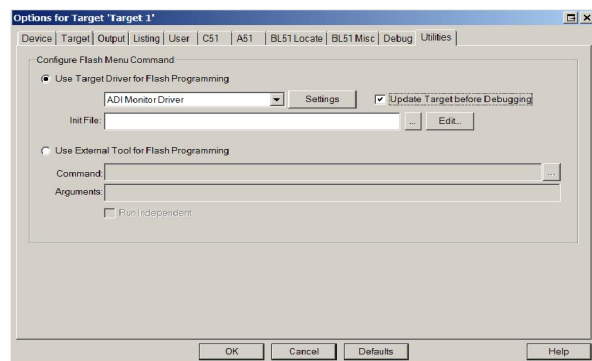


Figuur 7 - Tabblad DEBUG



Figuur 8 - Tabblad DEBUG/ADI Monitor Settings

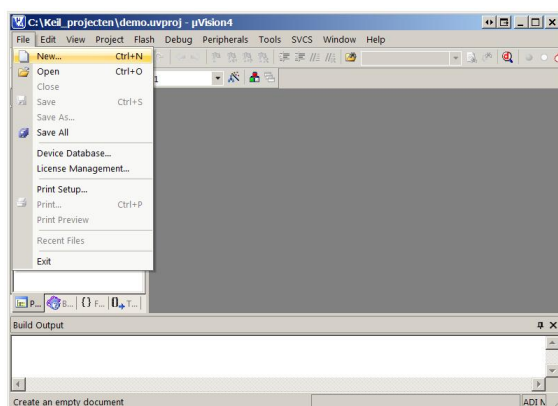
Selecteer hier de **COM-poort** van de ADuC847.



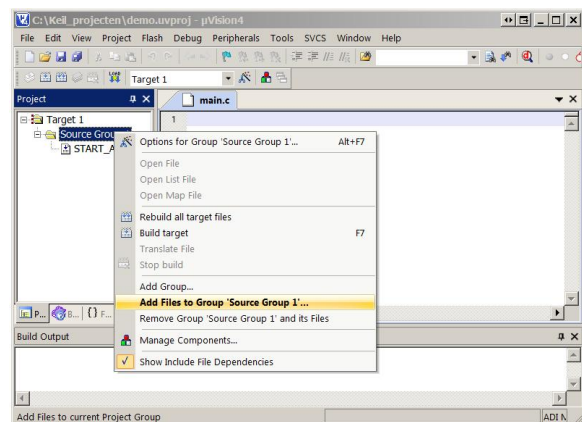
Figuur 9 - Tabblad Utilities

Klik nu op **OK**. We gaan nu een source(*.c) file aanmaken en voegen deze aan het project “Demo” toe. Het aanmaken en toevoegen van headerfiles(*.h) gebeurt op dezelfde manier.

6. Selecteer in de menubalk **File/New**. Zie Figuur 10. Er wordt nu een leeg bestand geopend. Selecteer vervolgens het bestand en selecteer in de menubalk **File/Save As...** Kies nu een geschikte naam, ik kies voor **main.c** (hierin plaatsen we namelijk de **main()**-routine).. Nu we het bestand **main.c** hebben aangemaakt dienen we het enkel nog toe te voegen aan ons project. Dit doen we door rechts te klikken op **Source Group** en dan te klikken op **Add Files to Group “Source Group 1...”**, zie Figuur 11.



Figuur 10 - Create New file (empty)



Figuur 11 – Add files to project

7. Neem nu de volgende voorbeeld-code over in `main.c`

```
#include <ADuC847.h>

// MotorControl_v0.1
sbit FORWARD = P2^5;
sbit REVERSE = P2^6;

// Leds
sbit LED1 = P0^0;
sbit LED2 = P0^1;

// switches
sbit SW1 = P1^6;           // active low
sbit SW2 = P1^7;           // active low
#define SW1_PRESSED() !SW1 // maak hier handig gebruik van!
#define SW2_PRESSED() !SW2 // maak hier handig gebruik van!

// functie-prototypes
void init (void);

void main()
{
    init();

    while (1)
    {
    }
}

void init (void)
{
    PLLCON = 0x00; // Core Clock Frequency = 12.582912
    WDE = 0;       // Turn watchdog off

    /* Leds */
    P0 = 0x00;

    /* DAC */
    DACCON = 0x0F; // DAC enabled, DAC PIN = P14, 8bit DAC, output [0 - Vcc]
    DACL = 0x00;   // DAC output: 0V

    /* SWITCHES */
    P1 = 0;
    SW1 = 0; // Digital input (switch1)
    SW2 = 0; // Digital input (switch2)

    /* MOTORCONTROL */
    FORWARD = 0; // set to move motor forward
    REVERSE = 0;


    /* TIMERS */
    TMOD = 0x00;

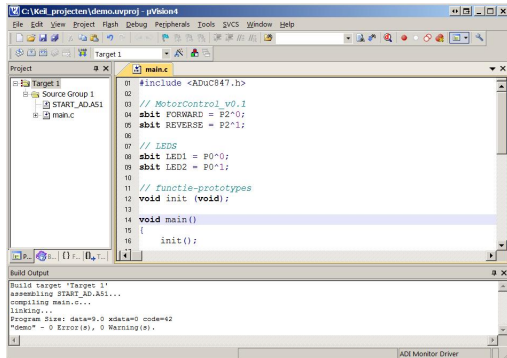
    /* INTERRUPTS */
    EA = 1; // GIE
}
```

Een aantal belangrijke opmerkingen:

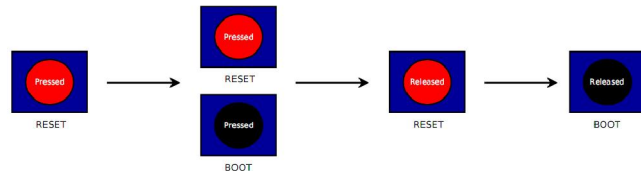
- `#include <ADuC847.h>`, hierin staan de declaratie van de **SpecialFunctionRegisters (SFR)**. Deze heb je nodig om de juiste hardware onderdelen te kunnen aanspreken vanuit C. Denk hierbij aan IO-poorten en geïntegreerde bouwstenen zoals een DAC of seriële poort (UART).
- `sbit FORWARD = P2^0;` Om het aanspreken van één individuele pin van een IO-poort te vergemakkelijken kun je zogenaamde **sbit** declaraties definiëren. Aan IO-PIN 2.1 is nu de naam FORWARD gekoppeld. Zie hiervoor ook [bijlage 3](#).
- `init()`; Hierin staan een aantal initialisatie waarden. Belangrijk is dat we de **Core Clock** op de juiste frequentie instellen en dat we de **WatchDog** uitzetten.
- DAC wordt geactiveerd en op 0V gezet.

8. Compileren en programmeerstand.

Door nu op **Rebuild**  te klikken wordt het project “gebuid” (volledig gecompileerd). Indien je geen typfouten hebt gemaakt zal er in de build output de melding “**0 Error(s), 0 Warning(s)**” verschijnen. Zie Figuur 12. Alvorens we de microcontroller kunnen programmeren moeten we deze wel nog eerst in programmeerstand zetten. Dit doen we door de drukknoppen **RESET** en **BOOT** te bedienen in de volgorde zoals afgebeeld in Figuur 13.

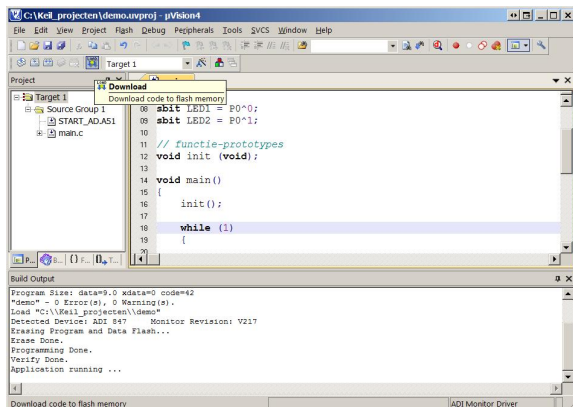


Figuur 12 – AduC847 succesvol geprogrammeerd

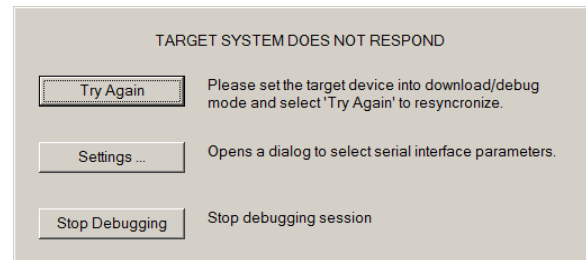


Figuur 13 – AduC847 in programmeerstand

9. Klik nu op om de microcontroller te programmeren. Zie Figuur 14. Indien er een foutmelding wordt weergegeven zoals in Figuur 15, dan betekent dit meestal dat de AduC847 niet in programmeerstand staat, of omdat de juiste COM-poort niet is ingesteld onder **Target Options**.




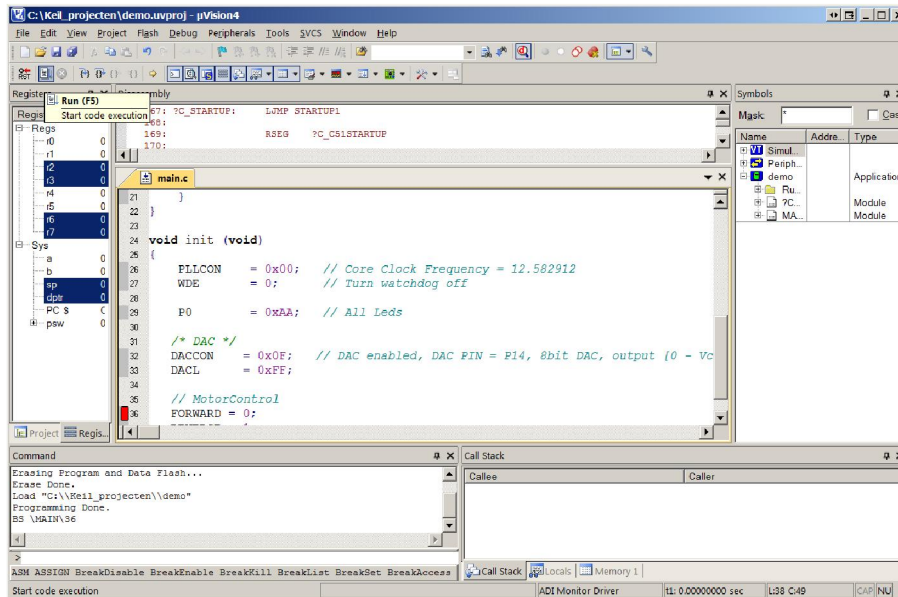
Figuur 14 – Succesvol programmeren AduC847





Figuur 15 – AduC847 reageert niet en kan niet worden geprogrammeerd

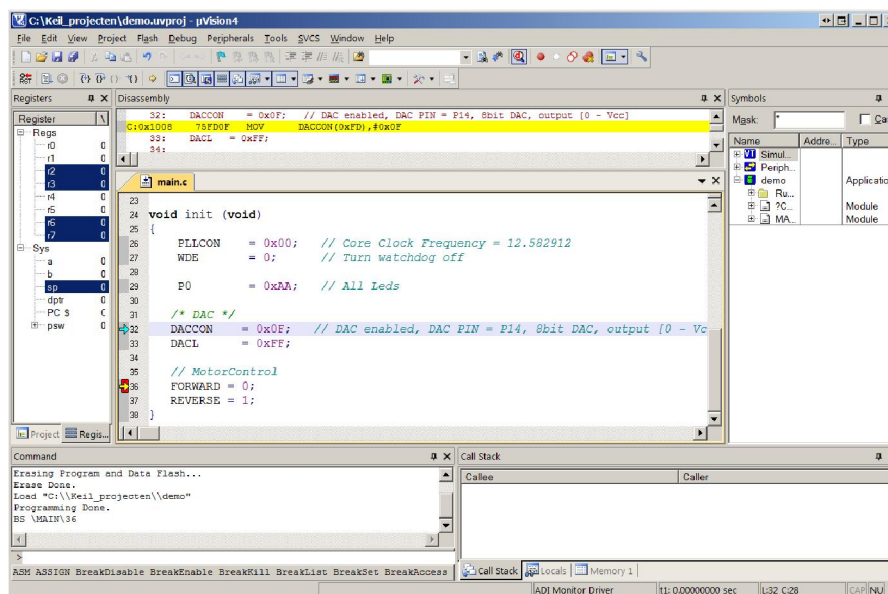
10. Debuggen m.b.v. breakpoints

We gaan eerst een breakpoint selecteren. Dit doen we door voor een code-regel dubbel te klikken (net zoals bij Visual Studio). Zie Figuur 16. Al we nog eens dubbelklikken op het rode rechthoekje voor de coderegel verdwijnt het breakpoint weer. We laten echter één breakpoint staan. **Zorg ervoor dat de ADuC847 in programmeermodus staat** (zie Figuur 13). Klik nu op de . De debugger wordt nu geopend. Zie Figuur 16.



Figuur 16 – breakpoint + debugmode

11. Wanneer je nu op Run  klikt zal het programma worden uitgevoerd totdat het een breakpoint tegenkomt. Zie Figuur 17. In de menubalk optie **Debug** vind je vergelijkbare debug functionaliteiten zoals onder Microsoft Visual Studio: *Step Over*, *Step Out* etc...
12. Klaar bent met debuggen? Klik weer op de  om de debugger **NETJES** af te sluiten.



Figuur 17 – programma uitgevoerd tot Breakpoint (zie gele pijl in rood rechthoekje)