1 Project Embedded systems

Inhoud

1.1 De vereisten

Bespreek uw project samen met de docent. Begin enkel aan uw project als de docent het heeft goedgekeurd.

Het project moet minimaal LwIP of FreeRTOS (nuttig) gebruiken. Indien je kiest voor FreeRTOS moeten er meerdere taken (nuttig) gebruikt worden. Andere evenwaardige Third party-libraries kunnen eventueel ook maar enkel na overleg met de docent.

Enkel de toegevoegde code moet voldoen aan de style guide, Naming Conventions en coding standaard voorschriften zoals hieronder beschreven.

1.2 Style Guide

Als Style Guide gelden volgende voorschriften:

Gebruik geen variabelen van het type unsigned char, int, long of long long (zowel signed, unsigned als *) maar gebruik hun typedef equivalent zoals bijvoorbeeld: int8_t, uint8_t, … (zie "inttypes.h" en/of "stdint.h"). Uitzondering hierop zijn de char en char* als ze enkel en alleen voor ASCII waardes gebruikt worden.

Alle code moet juist geïndenteerd zijn d.w.z. inspringen op de juiste plaats. (zie ook verder)

1.2.1 Layout binnen een bestand:

```
/* Library includes komen eerst... */
#include <stdlib.h>

/* ...dan de includes van stellarisware gegroepeerd (inc,
driverlib, utils) ... */
#include " driverlib/systick.h"

/* ...gevolgd door uw eigen includes. */
#include "eigen.h"

/* Dan komen de defines, met haakjes waar nodig. */
#define A_DEFINITION    ( 1 )

/*
 * Waarna de Static (file private) function prototypes komen
 */
static void prvAFunction( uint32_t ulParameter );

/* De File scope variables komen als laatste, vlak voor de
functie definities */
static myType_t xMyFileGlobalVariable

/* Om dit deel af te sluiten gebruik je een horizontale lijn
zoals hieronder. Dit doe je ook achter elke functie definitie

/*-----------------------------------------------------------*/

void vAFunction( void )
{

}
/*-----------------------------------------------------------*/

static UBaseType_t prvNextFunction( void )
{

}
/*-----------------------------------------------------------*/
```

1.2.2 Layout van C constructs

```
/*
- Functie namen staan op 1 lijn inclusief het return type en de
argumenten.
- Er is geen spatie voor het open haakje '(' maar wel 1 spatie
erna.
- Er is ook een spatie voor het sluiten haakje ')'
- Er is een spatie na elke komma.
- functie parameters krijgen beschrijvende namen (_niet_ zoals
onderstaand voorbeeld)
- De accolades staan alleen op een lijn uitgelijnd onder elkaar.
*/
void vAnExampleFunction( int32_t lPar1 )
{
/* Declaraties van variablen worden niet geidenteerd en staan
enkel vanboven in een functie voor het eerste statement.
*/
uint8_t ucByte;
```

```
    /* De Code is geindenteerd.  Accolades staan op zich en
uitgelijnd onder elkaar.
     */
    for( ucByte = 0U; ucByte < fileBUFFER_LENGTH; ucByte++ )
    {
        /* Terug indenteeren*/
    }
}

/* For, while, do en if constructies volgen een vergelijkbaar
patroon.
- Er is geen spatie voor het open haakje '(' maar wel 1 spatie
erna.
- Er is ook een spatie voor het sluiten haakje ')'
- Er is een spatie na elke punt komma ';'.
- Er is een spatie voor en na elke operator (vb: '+')
- Je mag niet/minimaal vertrouwen op "operator precedence".
Gebruik daarom haakjes om de volgorde expliciet kenbaar te
maken.
- Magic numbers (constantes) die niet 0 zijn worden vervangen
door een constante variabele of een #defined constant.
for( ucByte = 0U; ucByte < fileBUFFER_LENGTH; ucByte++ )
{
}

while( ucByte < fileBUFFER_LENGTH )
{
}

/* operator precedence: gebruik van de haakjes in een 'if' */
if( ( ucByte < fileBUFFER_LENGTH ) && ( ucByte != 0U ) )
{
    /* operator precedence: gebruik van de haakjes in een
bewerking */

    ulResult = ( ( ulVal1 + ulVal2 ) - ulVal3 ) * ulVal4;
}

/* Voorwaardelijke compilatie krijgt dezelfde layout als andere
code.*/
#if( configUSE_TRACE_FACILITY == 1 )
{
    /* Voorbeeld: Onderstaande code alleen _toevoegen_ als
bovenstaande config op 1 staat. */
    pxNewTCB->uxTCBNumber = uxTaskNumber;
}
#endif
```

1.3 Naming Conventions

1.3.1 Namen van variabelen

Namen van variabelen van het type **uint32_t** worden voorafgegaan van **ul**, waar 'u' staat voor unsigned en 'l' voor long.

Namen van variabelen van het type **uint16_t** worden voorafgegaan van **us**, waar 'u' staat voor unsigned en 's' voor short.

Namen van variabelen van het type **uint8_t** worden voorafgegaan van **uc**, waar 'u' staat voor unsigned en 'c' voor char.

In lijn met bovenstaande regels worden namen van variabelen van het type **int32_t**, **int16_t**, **int8_t** vooraf gegaan door l, s, of c

Namen van variabelen van **niet stdint types** (zie stdint.h) worden voorafgegaan van **x** voor signed types of **ux** voor unsigned types.

Namen van **Enumerated** variabelen worden voorafgegaan van een **e**.

Namen van **pointer** variabelen worden bijkomstig voorafgegaan door het karakter **p**. Bijvoorbeeld: een pointer naar een uint16_t krijgt dan de prefix pus. **Void** pointers krijgen de prefix **pv**

Na de prefix is de eerste letter van de naam een hoofdletter. Indien de naam uit meerdere woorden bestaat is elke begin letter van elk nieuw woord terug een hoofdletter. (geen underscore) bijvoorbeeld:

uint32_t ulMijnNaam; //met hoofdletter M en N omdat dit het begin is van een woord.

1.3.2 Namen van functies

Functies met een beperkte scope (file scope) met behulp van het keyword static krijgen de prefix prv van private. Hun declaratie staat niet in een header-bestand file maar bovenaan het C bestand (zie Layout binnen een bestand) De prefix van het return type zoals hieronder beschreven hoeft dan niet.

Functies krijgen de prefix van hun return type net zoals hierboven beschreven is bij Namen van variabelen. Waarbij v gebruikt word voor het return type void.

Voor de eigenlijke functienaam komt de naam van het bestand waar de functie definitie staat (kan eventueel een afgekorte naam van het bestand zijn).

Elk nieuw deel/woord begint met een hoofdletter.

Voorbeeld1 ulFileGetCounter:

- Return type is uint32_t (unsigned long)
- Functie definitie staat in file.c
- Naam van de functie is GetCounter

Voorbeeld2 vBestandSetPWM:

- Return type is uint32_t (unsigned long)
- Functie definitie staat in bestand.c
- Naam van de functie is SetPWM

Voorbeeld3 prvSpecialAllocateMem:

- Private functie kan enkel gebruikt worden in special.c
- Return type is niet af te leiden uit de functienaam.
- Functie definitie staat in special.c
- Naam van de functie is AllocateMem

1.3.3 Namen van Macro's

Macro's worden geprefixed met de naam van het bestand waarin het gedefineerd wordt. De prefix is bestaat uit allemaal kleine letters (lower case). De naam van de Macro zelf bestaat enkel uit hoofdletters. Elk woord is gescheiden door een hoofdletter.

Voorbeeld: configIP4_SET_ADDR

- Is een macro omdat het 2de deel uit hoofletters bestaat
- Staat in het bestand config.h gedefinieerd.
- Heeft als naam IP4_SET_ADDR

1.4 Coding standard

Al de zelfgeschreven code moet conform zijn met de "MISRA C coding standard guidelines". Omdat het naleven van heel de standaard teveel werk is binnen dit vak. Moet uw project enkel voldoen aan een subset van de misra c standaard.

- 5.2 (req): Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier.
- 6.1 (req-): The plain char type shall be used only for storage and use of character values.
- 6.3 (adv): 'typedefs' that indicate size and signedness should be used in place of the basic types.
- 7.1 (req): Octal constants (other than zero) and octal escape sequences shall not be used.
- 8.5 (req-): There shall be no definitions of objects or functions in a header file.

- 8.7 (req): Objects shall be defined at block scope if they are only accessed from within a single function.
- 8.11 (req): The static storage class specifier shall be used in definitions and declarations of objects and functions that have internal linkage.
- 9.1 (req-): All automatic variables shall have been assigned a value before being used.
- 13.3 (req): Floating-point expressions shall not be tested for equality or inequality.
- 13.4 (req): The controlling expression of a 'for' statement shall not contain any objects of floating type.
- 13.6 (req-): Numeric variables being used within a 'for' loop for iteration counting shall not be modified in the body of the loop.
- 14.1 (req-): There shall be no unreachable code.
- 14.4 (req): The 'goto' statement shall not be used.
- 14.8 (req): The statement forming the body of a 'switch', 'while', 'do ... while' or 'for' statement be a compound statement.
- 14.9 (req): An 'if-expression' construct shall be followed by a compound statement. The 'else' keyword shall be followed by either a compound statement, or another 'if' statement.
- 14.10 (req): All 'if ... else if' constructs shall be terminated with an 'else' clause.
- 15.2 (req): An unconditional 'break' statement shall terminate every non-empty 'switch' clause.
- 15.3 (req): The final clause of a 'switch' statement shall be the 'default' clause.
- 16.2 (req): Functions shall not call themselves, either directly or indirectly.
- 16.5 (req): Functions with no parameters shall be declared with parameter type void.
- 16.7 (adv-): A pointer parameter in a function prototype should be declared as 'pointer to const' if the pointer is not used to modify the addressed object.
- 16.8 (req): All exit paths from a function with non-void return type shall have an explicit 'return' statement with an expression.
- 17.5 (adv): The declaration of objects should contain no more than 2 levels of pointer indirection.
- 21.1 (req-): Minimization of run-time failures shall be ensured by the use of at least one of:
  - static analysis tools/techniques;
  - dynamic analysis tools/techniques;
  - explicit coding of checks to handle run-time faults.

Voor de duidelijkheid volgens 14.10 mag het volgende wel:

```
if( ( ucByte < fileBUFFER_LENGTH ) && ( ucByte != 0U ) )
{
    ulResult = ( ( ulVal1 + ulVal2 ) - ulVal3 ) * ulVal4;
}
```

Maar niet:

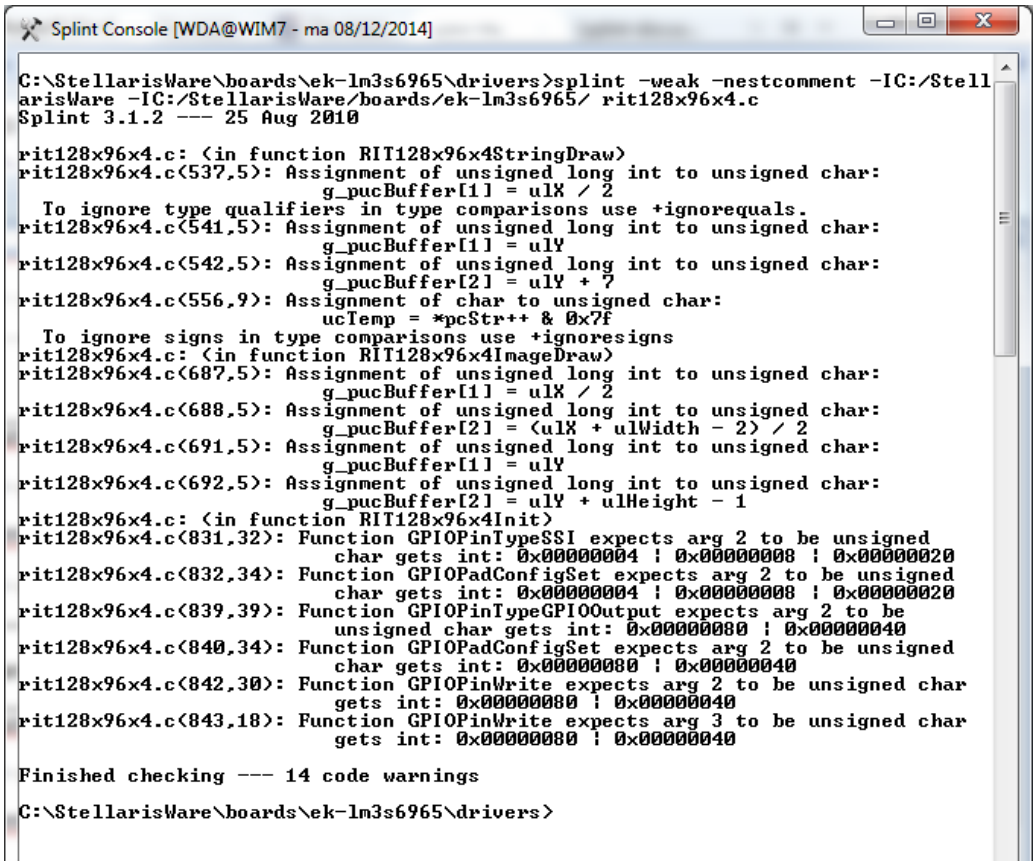
```
if( ( ucByte < fileBUFFER_LENGTH ) && ( ucByte != 0U ) )
{
    ulResult = ( ( ulVal1 + ulVal2 ) - ulVal3 ) * ulVal4;
}
else if ( ucByte == fileBUFFER_LENGTH )
{
    ulResult = ( ( ulVal1 + ulVal2 ) - ulVal3 ) * ulVal4;
}
```

Dit zou dan moeten worden veranderd naar:

```
if( ( ucByte < fileBUFFER_LENGTH ) && ( ucByte != 0U ) )
{
    ulResult = ( ( ulVal1 + ulVal2 ) - ulVal3 ) * ulVal4;
}
else if ( ucByte == fileBUFFER_LENGTH )
{
    ulResult = ( ( ulVal1 + ulVal2 ) - ulVal3 ) * ulVal4;
}
else
{
    /* ucByte kan nooit groter worden dan fileBUFFER_LENGTH */
}
```

Voor 21.1 kan je o.a. cppcheck gebruiken. Zie
http://sourceforge.net/projects/cppcheck/

Maar ook splint zie http://www.splint.org Indien je deze installeert krijg je een splint console. Hieronder zie je een voorbeeld hoe je deze kan gebruiken op het bestand rit128x96x4.c in de map C:\StellarisWare\boards\ek-lm3s6965\drivers.



De commando's die hiervoor gebruikt zijn:

cd C:\StellarisWare\boards\ek-lm3s6965\drivers
splint -weak -nestcomment -IC:/StellarisWare -IC:/StellarisWare/boards/ek-lm3s6965/ rit128x96x4.c

- -weak zorgt ervoor dat er niet te streng gecheckt wordt.
  kan je weglaten als je veel warnings/errors wil.
- -nestcomment zorgt ervoor dat er geen warnings gegeven worden voor geneste comments. Dit komt in de source van stellarisware veel voor omdat ze //********** gebruiken om horizontale lijnen te maken in de tekst.
- -I om de include directory mee te geven met splint zodat deze weet waar hij de header files kan vinden. Zie de configuratie van uw project onder de tab compile voor de include paden van uw project. Merk op dat splint enkel met absolute paden omkan.

1.5 Hoe inleveren en wanneer

- Pagina aanmaken op TRAC (vlak na goedkeuring):
- Op de hoofdpagina van ES een link met
  naam, voornaam, titel van het project
- Op je eigen (nieuwe) pagina
  titel van het project, naam, voornaam, korte beschrijving, lijst van alle programma's en alle bibliotheken met hun versies die je gebruikt hebt. Een link naar je SVN repo.
- Indienen 48u voor aanvang van het examen via:
  http://svn.pbei.be/[uw_traclogin]/projectES/

(Een svn client kan je downloaden van https://tortoisesvn.net/)
Indien problemen met svn: via mail.

- Zelf een Readme schrijven met daarin:
  - Zelfde info als op uw trac pagina +
  - Welke bestanden je zelf hebt aangemaakt/aangepast
  - Eventuele extra's: zoals welke tools je gebruikt hebt en je bevindingen. vb: cppcheck, astyle, splint, doxygen,…
  - Plaats deze readme in de root van je project (dus ook op svn)

1.6 Over de mondelinge verdediging van je project

Je zal vragen krijgen over je project. Deze vragen lijken op vragen zoals je mag verwachten op een eindwerk verdediging. Ze zijn bedoeld om te zien of je je project zelf gemaakt hebt en om te zien hoe goed je alles begrijpt/onder de knie hebt.
Wat moet je thuis zeker bestuderen of wat moet je zeker kunnen (lijst is niet limitatief):

- Je startup-code kunnen uitleggen
- Elk woord/karakter kunnen uitleggen binnen zijn context van elke zelf geschreven/aangepast bestand
- Kunnen verdedigen waarom je iets doet zoals jij het doet
- Globale werking van de gebruikte libs/functie calls
- De gebruikte hardwarefunctionaliteiten kunnen uitleggen
- …

Een werkend project geeft geen garantie op goede cijfers! (maar het helpt wel). Bereid alles dus zorgvuldig voor zodat je EN een werkend project hebt EN dat je alles begrijpt.