



## L2 - Embedded C programming

### Le langage C pour les systèmes embarqués

#### Objectifs

- Découvrir le C embarqué au moyen d'exercices sur une carte cible avec un microcontrôleur STR912FA (coeur ARM)
- Comprendre le fonctionnement d'une chaîne de compilation
- Analyser les options d'optimisation de code du compilateur
- Développer des applications pour accéder aux périphériques d'E/S
- gérer des interruptions
- gérer la transmission/réception série de données
- Interfacer C et assembleur pour mettre en oeuvre des instructions spécifiques du processeur cible

#### Matériel

- Un PC, une carte STR912-SK et une sonde JTAG Lauterbach par binôme
- Environnement de développement croisé : Eclipse avec compilateur GCC et environnement de debug Trace32 (pour les sondes Lauterbach)
- Manipulations sur carte nue
- Un support de cours ainsi que les solutions des exercices

#### Pré-requis

- Connaissance élémentaire du langage C
- Connaissances de l'arithmétique binaire
- Connaissance d'un processeur recommandée

#### Plan

##### *Conception d'un logiciel structuré*

- Découpage d'une application en plusieurs fonctions
- Séparation en fichiers distincts
- Intérêt des fichiers header et des librairies

##### *Analyse des éléments constitutifs d'une chaîne de compilation*

- Explication des étapes du processus de génération de code en natif et en croisé
- Rôle du compilateur, de l'assembleur et du linker
- Paramétrage en fonction d'un mapping mémoire
- Structure d'un programme source C, distinction des parties essentielles : préprocesseur, déclaration des variables globales, fonctions

- Découpage d'une application en fichiers distincts
- Le préprocesseur
- Les instructions #define et #include
- Ecriture de macros
- Précautions à prendre dans les headers pour éviter les redéclarations de variables

*Exercice : mise en œuvre de la chaîne de compilation*

### Les types du langage C et les opérateurs

- Classes d'allocation des variables (static, automatic, register et extern) et analyse de leurs durées de vie
- Les types scalaires : char, int, float et double
- Codage des nombres entiers en code binaire naturel ou en code complément à 2
- Déclaration des variables
- Les conversions de type implicites et explicites (casting)
- Les directives de compilation pour forcer l'alignement du code et des données, intérêt sur les processeurs munis de caches
- L'attribut volatile pour éviter les optimisations de code par le compilateur
- Les constantes
- Les opérateurs du C : logiques, arithmétiques et relationnels
- L'opérateur " , "
- Priorité entre opérateurs

*Exercice : écriture de programmes mettant en oeuvre les concepts ci-dessus*

### Les traitements itératifs et conditionnels

- La structure if/else
- La structure switch/case
- Les boucles : while, do/while et for
- Les instructions break et continue
- L'instruction goto
- Les optimisations du compilateur au niveau des structures de contrôle du langage C : exemple le dépliement des boucles

*Exercice : écriture de programmes mettant en oeuvre les notions ci-dessus*

### Les pointeurs et les tableaux

- Mise en évidence de la distinction adresse / contenu
- Calculs d'adresses, opérations mêlant pointeurs et adresses absolues
- Les tableaux mono et multi-dimension
- Pointeurs constants et volatile
- Les pointeurs sur fonctions
- Les tableaux de pointeurs

*Exercice : allocation d'un pointeur sur un port d'IO*

*Exercice : opérations courantes sur les pointeurs*

*Exercice : copie de tableaux avec les notations pointeur et tableau*

### Les structures et les unions

- Déclaration de variables structure
- Déclaration de pointeurs sur des variables structure
- Accès aux champs d'une structure
- Mise en évidence du padding imposé par les règles d'alignement
- Directive de compilation : #pragma pack
- Les formats big et little endian

- Les structures à champ de bits et modélisation des périphériques
- Utilisation de typedef
- Les tableaux de structures
- Le type union pour envisager un même emplacement mémoire de différentes manières

*Exercice : Exercice : modélisation du timer du Str912FaW44*

### Les fonctions

- Prototypage des fonctions : notions d'argument et de valeur de retour
- Déclarations anticipées de fonctions
- Les espaces de validité des variables
- La récursivité : impact sur la pile
- Mise en évidence de l'allocation d'un stack frame lors du processus d'appel
- Visualisation des stacks frames empilés (call stack)
- la convention d'appel
- La fonction main et ses arguments
- Utilité des tableaux de pointeurs sur des fonctions
- Interface C/assembleur

*Exercice : Exercice : appel d'une fonction avec les deux modes de passage de paramètres*

*Exercice : Exercice : écriture d'une fonction récursive qui calcule la factorielle d'un nombre*

*Exercice : Exercice : interfacier du C avec du code assembleur*

### L'allocation dynamique

- Les fonctions d'allocation dynamique de mémoire : malloc, calloc, realloc et free
- Intérêt de l'opérateur sizeof
- Allocation dynamique versus allocation statique de mémoire
- Allocation dynamique de mémoire et temps réel
- Algorithmes d'allocation dynamique de mémoire
- Distinction tas / pile
- Tas par défaut et tas privé

*Exercice : Exercice : réalisation d'une liste chaînée de tâches prêtes*

### Interruptions, transmission/réception série de caractères

- Nécessité des interruptions dans un système embarqué
- Distinction entre déclenchement sur front et sur niveau
- Acquiescement logiciel
- Ecriture d'un gestionnaire d'interruption : distinction des 3 étapes prologue / corps / épilogue
- Table de vecteurs
- Ecriture des fonctions d'installation et de lecture de vecteur
- Initialisation d'un UART
- Ecriture de fonctions bas niveau pour lire et écrire des caractères du et vers un UART

*Exercice : Exercice : programmation d'un timer pour générer des interruptions*

*Exercice : Exercice : programmation d'un uart pour recevoir et transmettre des caractères*

### Le langage C dans le contexte embarqué

- Phases de startup et de terminaison
- Cstartup : initialisation du pointeur de pile et mise à 0 des variables non initialisées
- Nécessité de l'attribut volatile

- Portabilité:
  - big-endian et little endian
  - padding et alignement
- Optimisation du code
- Ecriture de code efficace

*Exercice : Mise en évidence et détection de fuites mémoire*

### Les entrées/sorties standards

- Les fonctions de la bibliothèque stdio
- Prototypage des fonctions de base : getchar et putchar
- Prototypage des fonctions qui effectuent un formatage : printf, scanf
- Notion de fichier : l'accès aux fichiers
- Traitement des erreurs : stderr et exit

*Exercice : utilisation des fonctions stdio pour la communication avec un terminal*

### Mise au point

- Sonde JTAG Lauterbach
- Communication avec la cible
- Les différents niveaux de mise au point : C, assembleur
- Les fenêtres du debugger : source, mémoire, pile, variables, registres
- Positionnement de points d'arrêt
- Analyse de la pile et extraction des stacks frames correspondant aux fonctions imbriquées

## Renseignements pratiques

**Duration : 4 days**

**Cost : 1900 € HT**

**Prochaines sessions :** du 21 au 24 May 2012  
du 19 au 22 June 2012  
du 18 au 21 September 2012  
du 16 au 19 October 2012



SARL au capital de 15400€ - SIRET 449 597 103 00026 - RCS Nanterre - NAF 722C - Centre de Formation : 19, rue Pierre Curie - 92400 Courbevoie  
Siège social et administration : 21, rue Pierre Curie - 92400 Courbevoie - Tél. 01 41 16 80 10 - Fax. 01 41 16 07 78

Last site update: Tue 22 May 2012 10:50:29 CEST

<http://www.ac6-formation.com/>