Electronique embarquée
Conception et microprocesseurs

Richard Grisel – Professeur des Universités –
Université de Rouen

Conception et microprocesseurs

- Architectures et composants:
  - Logiciel;
  - Matériel.
- Test (Debugging).
- Test “in situ”.

Architecture matérielle

Contient plusieurs composants:

- CPU;
- Bus;
- Mémoire;
- Circuits I/O : mise en réseau, capteurs, actionneurs, etc.

Quel dimensionnement pour chaque partie (taille/vitesse)?

Architecture logicielle

La description fonctionnelle doit être séparé een plusieurs parties :

- Séparation entre les équipes;
- Oganisation conceptuelle;
- Performance;
- Testabilité;
- Maintenance.

Architectures matérielle et logicielle

"Hardware" et "software" sont liés:

- Le logiciel fonctionne sur un certain type de matériel;
- Le type de matériel est défini par les spécifications logicielles :
  - Vitesse ;
  - Mémoire
  - Entrées/Sorties

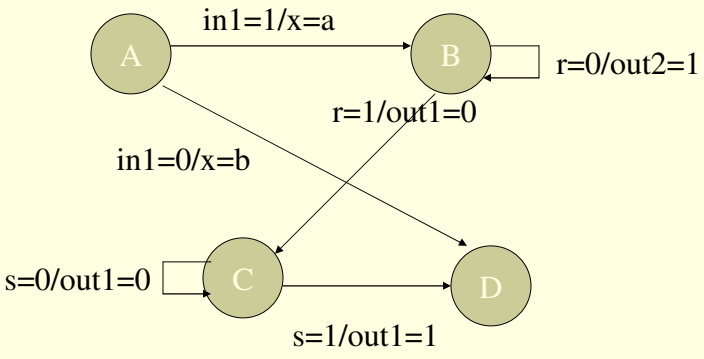
Composants logiciels

- Nécessité de séparer la conception en modules pour pouvoir écrire le code.
- Certains composants reviennent souvent.
- Une description générique d'un composant (si elle existe) peut être adaptée et utilisée dans plusieurs type de situations.

Machine d'états logicielle

- Une machine d'états a comme variable l'état courant et change d'état uniquement en fonction des entrées sensibles à cet état.
- Est utilisée pour :
  - Code pour le contrôle;
  - Systèmes réactifs.

Spécification d'une machine d'états



Structure en code C

- L'état actuel (courant) est dans une variable.
- La table des états est implémenté comme un "switch".
  - "Case" définit les états.
  - Dans un état, on peut tester les entrées.
- Le "Switch" est évalué de manière répétitive dans une boucle "while".

Structure en code C

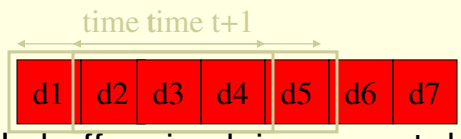
```
while (TRUE) {
   switch (state) {
       case state1: …
   }
}
```

Table d'états en C

```
switch (state) {
case A: if (in1==1) { x = a; state = B; }
  else { x = b; state = D; }
  break;
case B: if (r==0) { out2 = 1; state = B;
  }
  else { out1 = 0; state = C; }
  break;
case C: if (s==0) { out1 = 0; state = C;
  }
  else { out1 = 1; state = D; }
  break;
```

Flot de données

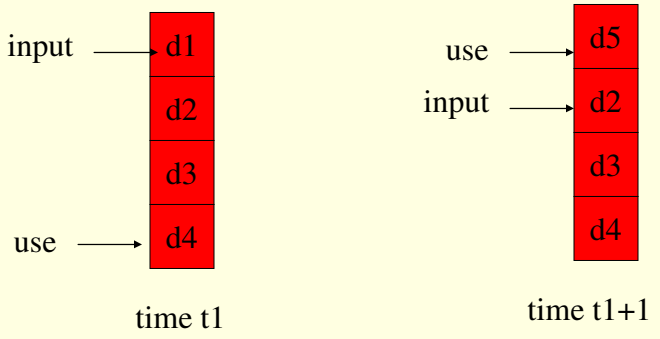
- Est communément utilisé en traitement du signal :
  - De nouvelles données arrivent systématiquement;
  - Chaque donnée a une "vie" limitée.



- Un buffer circulaire permet de gérer le flot de données.

Buffer circulaire

- Un pointeur indique la donnée en cours d'utilisation, un autre celle qui vient d'arriver:



Buffer circulaire en C

Calcul de la valeur f du filtre FIR:

```
f = 0;
  for (ic=0; ic < N; ic++){
    for (ibuff =0; ibuff =(ibuff == N-1 ?
      0: ibuff++)  {
      f = f + c[ic] * circ_buff[ibuff]
                              }  }
```

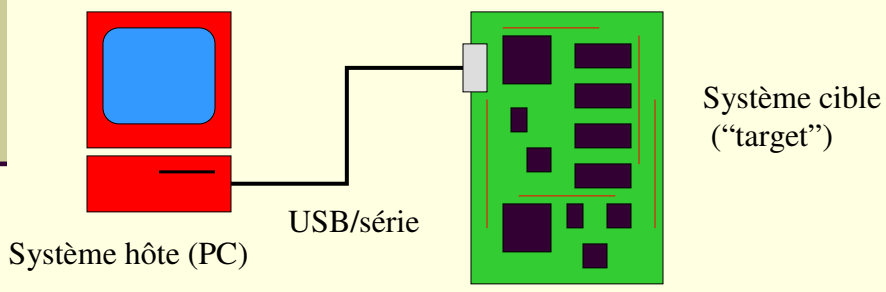
Les indices buffer et coefficients sont différents

Techniques de conception logicielles

- Nécessité de développer le plus de code possible sur une plateforme standard :
  - Environnement de programmation souple;
  - Facilités de test (“debug”).
- Peut impliquer une division des composants logiciels pour pouvoir tester sans que la plateforme soit complète (“émulation”, etc..).

Conception “Hôte/Cible”

- Utilisation d’un système hôte pour la préparation du logiciel pour la cible matérielle :



Outils sur l'hôte

- Cross compilateur:
  - Le processeur cible n'est généralement pas le même que sur l'hôte.
- Cross "debugger":
  - Visualise l'état de la cible, permet le contrôle de la cible (points d'arrêt, etc..).

Cartes d'évaluation

- Conçues par les fabricants de CPU ou autres.
- Contiennent CPU, mémoire(s), quelques circuits d'I/O.
- Une zone de "prototypage".
- Les schémas ("netlist") sont généralement fournis ce qui permet d'avoir un point de départ en terme de "design".

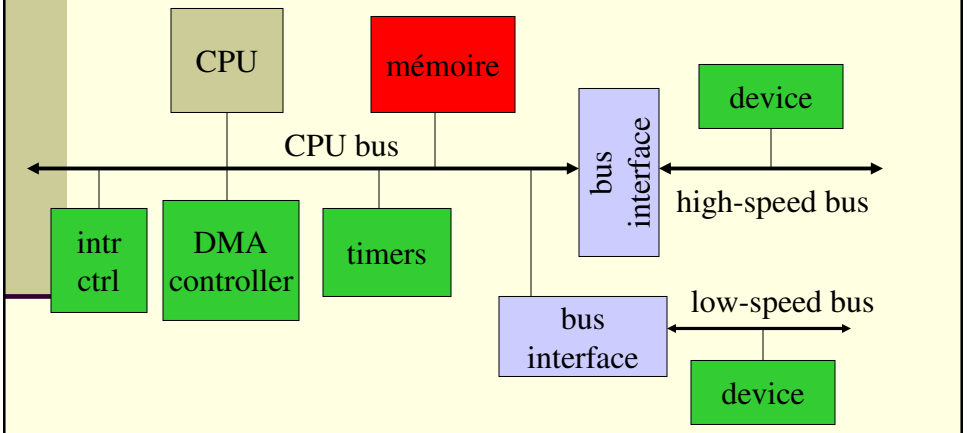
Ajout de “logique” à une carte

- Programmable logic devices (PLDs) densité faible à moyen en terme de composants.
- Field-programmable gate arrays (FPGAs) densité élevée et logique plus complexe.
- Application-specific integrated circuits (ASICs) dans des cas plus particuliers.

Le PC en tant que plateforme

- Avantages:
  - Faible coût et disponibilité;
  - Environnement logiciel riche et familier.
- Désavantages:
  - Nécessite beaucoup de ressources matérielles;
  - Peu adapté au temps réel.

Plateforme PC typique



Les bus de communication ?

- ISA (Industry Standard Architecture): PC originel de l'IBM PC, basse vitesse comparé aux standards actuels.
- PCI: Standard "haute" vitesse
  - 33 or 66 MHz.
- USB (Universal Serial Bus), Firewire: Interfaces séries haute vitesse à coût raisonnable.

Les éléments logiciels

- L'IBM PC utilise le BIOS (Basic I/O System) pour implémenter les fonctions de bas niveau:
  - démarrage;
  - "device drivers" de base.
- Le terme BIOS est un terme générique pour désigner le logiciel de plus bas niveau.

Exemple: StrongARM

- Un système StrongARM comprend:
  - Un CPU (Horloge de 3.686 MHz)
  - Un module de contrôle (Horlode de 32.768 kHz).
    - Horloge "temps réel";
    - Temporisateur pour l'OS;
    - Des I/O d'usage général;
    - Un contrôleur d'interruption;
    - Un contrôleur de gestion d'alimentation;
    - Un contrôleur d'initialisation ("reset").

Le test des systèmes embarqués

- Challenges:
  - La cible peut être difficile à observer;
  - La cible peut être difficile à contrôler;
  - Il peut être délicat de générer des entrées réalistes (pour le test);
  - La séquence d'initialisation peut être complexe.

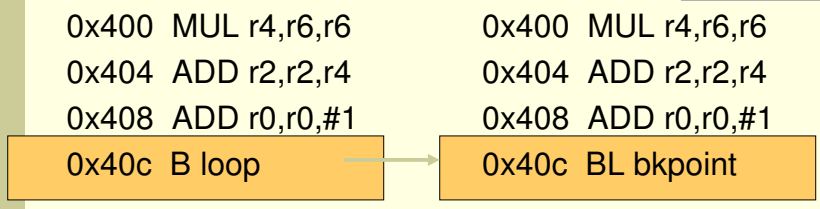
Les "debuggers" logiciels

- Un programme (moniteur) est résident sur la cible et fournit des fonctions de base pour le test.
- Ce moniteur doit utiliser peu d'espace mémoire.
- Le programme utilisateur ne doit pas détruire le moniteur (ou ses données) mais doit être capable de récupérer en cas d'échec (du à un dysfonctionnement du code utilisateur).

Les points d'arrêt

- Un point d'arrêt stoppe l'exécution, permet l'examen de l'état du système, voire le changement de cet état.
- En fait le point d'arrêt "remplace" l'instruction par un appel de sous-programme du moniteur (RAM, ROM ???).

Point d'arrêt ARM



Code classique

code avec point d'arrêt

Gestion des points d'arrêt (par le moniteur)

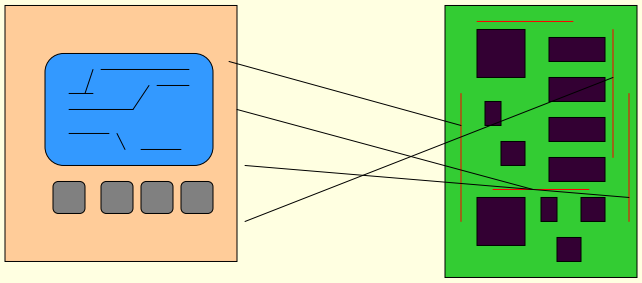
- Sauvegarde des registres.
- Examen de l'état du processeur.
- Avant le retour, restaure l'état du système.
  - Le moyen le plus sûr est de remplacer l'instruction par une autre et de l'exécuter à sa place.
  - On peut avoir un autre point d'arrêt après pour restaurer le point d'arrêt initial.

Les émulateurs "In-circuit"

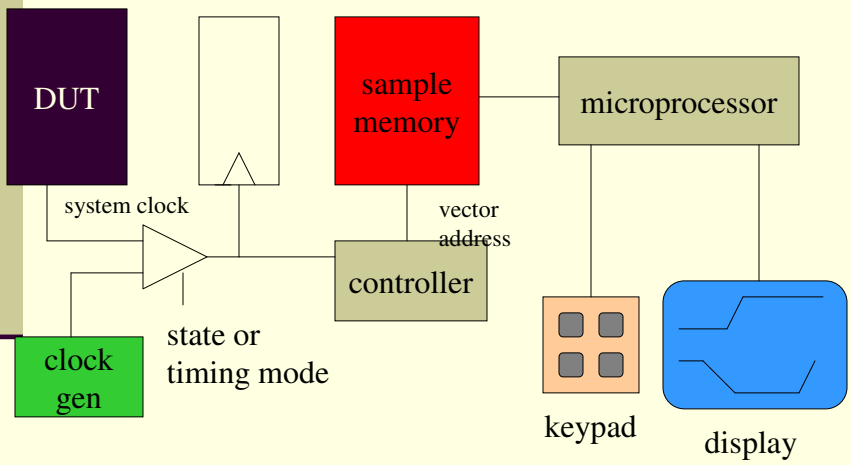
- C'est un microprocesseur spécialement instrumenté.
- On peut stopper l'exécution, examiner l'état du CPU, modifier les registres, etc..

Les analyseurs logiques

- C'est un "oscilloscope" pour les signaux numériques (plusieurs voies) .A



Architecture d'un analyseur logique



Comment expérimenter le code ?

- Exécution sur le système hôte.
- Exécution sur le système cible.
- Exécution par un simulateur (niveau instructions).
- Exécution dans un environnement matériel/logiciel de co-simulation.

Test en production

- But : S'assurer que la fabrication délivre des produits sans défaut.
- Peut être vérifié par une comparaison avec le comportement attendu du système.
  - La durée des tests est "chère".
- Il faut améliorer la confiance (dans le test) en minimisant le coût.

Nombre de tests: Exponentielle (n)?, avec n étant le nombre de bits des registres

Concepts pour le test

- Objectifs: proportion des systèmes fabriqués qui fonctionnent
  - Une fabrication “propre” maximise les objectifs
  - Des tests “propres” permettent une estimation précise des objectifs.
- Les problèmes: si des systèmes sont encore défectueux après les tets.

Les défauts

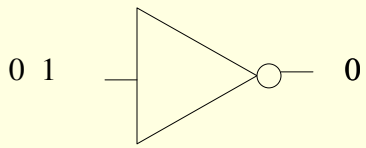
- Les problèmes de fabrication ont des causes nombreuses.
- Modèle de défaut : modèle qui prédit les effets d’un type particulier de défaut.
- Couverture des défauts: proportion de défauts possibles trouvés par un ensemble de vecteurs (de test).
  - UN modèle de défaut permet de déterminer la couverture des défauts.

Test logiciel contre matériel

- Quand on teste du code, il n'y a pas de modèle de défaut.
  - On vérifie l'implantation, pas la fabrication.
  - Des tests simples (e.g., ECC, CRCC) permettent de vérifier la cohérence du logiciel (après fabrication/transmission).
- Le matériel nécessite des tests en fabrication en plus des vérifications d'implantation .

Modèles de défaut matériels

- Modèle de collage à 0/1 (Stuck-at) :
  - On vérifie si une sortie reste collée à 0/1.



Test combinatoires

- Toutes les fonctions peuvent passer le test de collage à 0/1.
- Généralement on essaye de tester des défauts simples.
  - Un défaut à la fois.
  - Des défauts multiples peuvent se “masquer” les uns et les autres.
- On peut générer un test pour un fonction (porte logique):
  - En contrôlant les entrées de la fonction;
  - En observant les sorties de la fonction vers les autres fonctions.

Tests séquentiels

- Une machine d’état = Logique combinatoire + registres.
- Le test des fonctions séquentielles est considérablement plus difficile.
  - Un test de collage sur défaut modifie la machine pour chaque cycle.
  - Le comportement des défauts sur un cycle peut être masqué par le même défaut sur les autres cycles.

Chaines de “balayage” (Scan)

- Un registre balayable fonctionne de deux manières :
  - normal;
  - “scan” --- élément d’un registre à décalage.
- L’utilisation des chaînes ramène le tests séquentiel à un test combinatoire.
  - Chargement/Déchargement de la chaîne est lent.
  - On peut utiliser un “scan” partiel.

Génération des tests

- Les programmes de type “Automatic test pattern generation (ATPG)” produisent un ensemble de tests en fonction de la structure logique à tester .
- Quelques défauts peuvent ne pas être testables.
  - Un dépassement de temps (“timeout”) sur un défaut peut signifier que le test est impossible ou très difficile à réaliser.

“Boundary scan”

- Simplification du test multiple pour une carte.
  - Des registres pour les broches peuvent être configurés comme une chaîne de test.