Architecture des objets connectés

SE302a

Samuel Tardieu
23 septembre 2020

Définitions, concepts et historique

Systèmes embarqués

Un *système embarqué*, ou *système enfoui*, est un système autonome dont le but premier n'est pas d'être un ordinateur avec ses périphériques classiques [1].

1. Il existe autant de définitions que de cours

Systèmes embarqués

Il y a plusieurs types de systèmes embarqués, par exemple :

- ceux très chers et peu nombreux (spatial, avionique, ferroviaire, bateaux, nucléaire)
- les utilitaires et cachés (carte à puce, injection électronique, bouilloire, grille-pain)
- les médicaux (stimulateur cardiaque, contrôle de neurones pour pallier certains effets de la maladie de Parkinson)
- les gadgets (guirlande à motif sélectionnable, modèles réduits)
- les objets du quotidien (télécommande, téléphone mobile)

Certains appareils entrent dans plusieurs catégories (FitBit).

Caractéristiques des systèmes embarqués

Certaines caractéristiques varient énormément d'un système à l'autre :

- capteurs et effecteurs
- coût (développement, fabrication, distribution, maintenance)
- énergie (sur pile, batterie, secteur, énergie renouvelable)
- puissance de calcul
- stockage (firmware, données)
- propriétés temps-réel ou nom
- possibilité de communication (USB, ZigBee, WiFi, LoRa, 5G, etc.)
- évolutivité
- certifications (développement, fonctionnement)

Objets connectés

Une évolution naturelle, volonté de tout connecter :

- *consumer electronics* (électronique grand-public)
- objets connectés à travers une infrastructure locale (WiFi, BlueTooth)
- IoT (*internet of things*), objets communiquants autonomes (SMS, LoRa, 5G)

Exemples d'utilisation

- maison intelligente (*smart home*, *home automation*)
- ville intelligente (*smart city*)
- réseau électrique intelligent (*smart grid*)
- assistance aux personnes âgées (*elder care*) et handicapées
- agriculture
- renseignement

SE302 a une coloration « objets connectés », mais un grand nombre de concepts étudiés ici s'appliquent aux autres types de systèmes embarqués.

Premiers systèmes embarqués modernes

- système de guidage d'Apollo (~1965), utilisant les premiers systèmes intégrés

FIGURE 1 – Clavier et écran du système de guidage d'Apollo

Premiers systèmes embarqués modernes (suite)

- système de guidage du missile Minuteman II (1966), première production en masse de systèmes intégrés
- systèmes conçus autour du microprocesseur 4004 d'Intel (1971)
- systèmes conçus autour de microcontrôleurs (début des années 1980)

Aujourd'hui, les microcontrôleurs embarquent un grand nombre de périphériques (timers, GPIO, convertisseurs, PWM).

Communication avec l'extérieur

- Capteur : transforme des quantités du monde physique en valeur numérique (microphone, capteur de luminosité, accéléromètre, gyroscope), souvent de manière approximative
- Effecteur : agit sur l'environnement physique (haut-parleur, led), souvent de manière approximative
- Bus de communication : échange d'informations, idéalement sans perte

Un système embarqué qui ne communique pas ne sert à rien.

Capteurs

Principe d'un capteur

- Capte une valeur physique et la transforme en valeur numérique
- Peut nécessiter un filtrage préalable (théorème de Nyquist-Shannon) pour éviter le repliement de spectre en fonction de la période d'échantillonnage
- Introduit du bruit de quantification dû à la précision limitée du résultat

GPIO (entrée numérique)

- GPIO : *general-purpose input/output*
- En général, permet d'opter pour une entrée ou une sortie (effecteur)
- Même en mode sortie, il peut être utile de pouvoir lire l'entrée
- Fonctionnement général : si l'entrée est au dessus d'un seuil disponible dans la documentation, un 1 est lu, un 0 sinon
- Parfois équipé d'un pull-up ou pull-down interne activable
- Parfois équipé d'un trigger de Schmitt
- Parfois tolérant à des tensions supérieures (FT pour *five volt-tolerant* pour des microcontrôleurs alimentés en 3,3V)
- Parfois relié à un contrôleur d'interruptions (front ou niveau)

Exemple de GPIO sur STM32

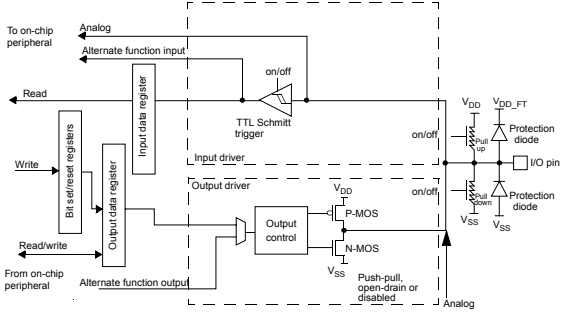

FIGURE 2 – Schéma d'un GPIO générique sur STM32 (source ST)

On notera que ce port peut également servir d'entrée à un convertisseur analogique-numérique.

Exercice : s'il vous plait... dessine moi un bouton !

- Comment faire pour connecter un bouton poussoir sur une entrée de microcontrôleur comme un GPIO de STM32 ?

Utilisation des timers

- Certains GPIO peuvent être couplés en entrée avec des timers
- Lorsqu'un événement (ou une combinaison d'événements) survient, la valeur du timer est capturée et stockée.
- Lors de la survenance, une interruption peut être déclenchée.

Il est donc possible d'obtenir les dates de survenance de un ou plusieurs événements avec la précision du timer (souvent jusqu'à un cycle d'horloge du microcontrôleur) même si le microcontrôleur n'est pas disponible pour un traitement immédiat.

Exemples de capteurs branchés sur une entrée numérique

- Interrupteur, bouton-poussoir
- Capteur Reed : interrupteur opéré par un champ magnétique qui fait se coller ou se séparer deux lames (ouverture de porte, détection de passage d'un point d'une roue), peu adapté aux fréquences rapides
- Photo-transistor en tout ou rien

Convertisseur analogique-numérique (ADC)

- Convertit linéairement une tension comprise entre la masse et une tension de référence V_{REF} possiblement différente de la tension d'alimentation et fournit un résultat sur N bits
- Le pas de conversion est donc $\frac{V_{REF}}{2^N}$
- Fonctionne en général en utilisation ponctuelle ou périodique
- Attention au repliement de spectre, peut nécessiter un filtre passe-bas préalable
- En anglais : *analog to digital converter* (ADC)

Défauts du convertisseur analogique-numérique

- Souffre d'un bruit intrinsèque de quantification lié à la résolution du résultat $\pm\frac{V_{REF}}{2^N}$
- Si la tension mesurée est grande devant l'erreur et que l'arrondi se fait au plus proche, la moyenne de l'erreur est 0 et sa moyenne quadratique (RMS) de $\frac{V_{REF}}{2^N\sqrt{12}}$
- Si l'arrondi se fait par tronquation, la moyenne de l'erreur est $\frac{V_{REF}}{2^{N+1}}$ et la moyenne quadratique $\frac{V_{REF}}{2^N\sqrt{3}}$
- Est sensible aux variations de V_{REF} et de température

Fonctionnement d'un ADC

Il existe plusieurs méthodes d'implémenter un ADC :

- Simple rampe (comparaison avec un DAC qui croît) : lent et peu stable
- Approximations successives (dichotomie sur des comparaisons avec un DAC) : plus rapide, mais peu stable
- Double rampe (charge d'un condensateur à partir de la tension à mesurer et mesure du temps de décharge) : lent mais stable

Fonctionnement d'un ADC (suite)

- Sigma-delta (suréchantillonage sur 1 bit avec rebouclage sur l'entrée) : précis, assez rapide et peu cher
- Flash (ensemble de comparaisons avec des diviseurs de tension correspondant à chaque valeur possible) : très rapide mais coûteux (point de $2^N - 1$ résistances) et imprécis dans la linéarité
- Semi-flash pipeline (découpage des N bits en plusieurs groupes) : rapide, assez coûteux, toujours des imprécisions dans la linéarité

Exemple de calibration en usine : STM32 (1/2)

- Une tension d'alimentation analogique V_{DDA} doit être fournie, proche de V_{DD} (différence de 300mV max pendant le démarrage, 140mV en régime stationnaire), et utilisée en général comme référence
- Une tension V_{REF_INT} est générée en interne et garantie stable à environ 1,22V
- Le convertisseur analogique-numérique est calibré en usine et la mesure de V_{REF_INT} avec V_{DDA} = 3,3V à 30℃ (avec une variation max de 5℃) est stockée dans une zone en lecture seule

Exemple de calibration en usine : STM32 (2/2)

- V_{REF_INT} est connectée en interne sur une entrée du convertisseur analogique-numérique
- Étant donné que V_{REF_INT} est stable sur toute la durée de vie du microcontrôleur, il est possible en la mesurant à nouveau de déduire la valeur courante de V_{DDA} pour peu qu'on soit à la bonne température

Précautions à prendre

Pour utiliser un ADC efficacement :

- Le signal d'entrée doit être filtré pour éviter les hautes-fréquences (repliement de spectre)
- Le signal d'entrée doit avoir une amplitude tenant dans les bornes de l'entrée analogique (typiquement 0-V_{REF})
- Le signal d'entrée doit occuper la plus grande plage de valeurs possible quitte à l'amplifier (ce qui peut rajouter du bruit)
- La résolution de l'ADC doit être adaptée aux données à traiter
- L'ADC doit être choisi tel que la marge de bruit soit tolérable (des ADC à amplificateur intégré à très faible bruit existent)

Exemples de capteurs branchés sur une entrée analogique

- Accéléromètre analogique : la tension qu'ils présentent est proportionnelle à l'accélération sur un axe
- Capteur à effet Hall : mesure du champ magnétique (sans partie mécanique), même si la plupart du temps une entrée en tout ou rien suffit
- Détecteur de luminosité (photo-diode dans sa partie linéaire)
- Potentiomètre

Cas d'étude : rotation autour d'un axe

- Détection d'une position absolue
  - Avec un ADC : potentiomètre sur l'axe
  - Avec des GPIO : codage de la position sur *N* bits avec *N* photo-diodes
  - Somme des positions relatives
  - Intégration de la vitesse angulaire
- Détection d'une variation d'angle
  - Roue codeuse, avec ou sans quadrature de phase
  - Intégration de la vitesse angulaire sur un intervalle de mesure
- Vitesse angulaire
  - Utilisation d'un gyroscope
  - Comptage de tours ou de portions de tours (capteur Reed, capteur à effet Hall, photo-diode)

Capteurs MEMS

- MEMS : microelectromechanical systems, dispositifs électro-mécaniques
- Aussi appelés micromachines au Japon ou technologie microsystème en Europe
- Construits à base de silicium, des métaux, des matériaux piézo-électriques, des polymères, etc.
- Exploitent des propriétés capacitives ou résistives des matériaux
- Utilisés pour construire des accéléromètres, gyroscopes, boussoles, capteurs de pression, microphones, etc.

Exemple : accéléromètre MEMS un axe

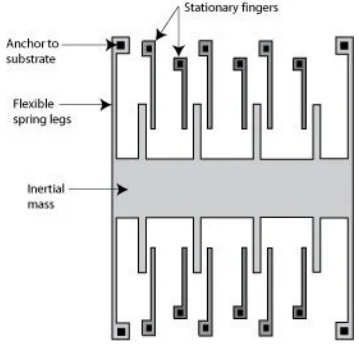

FIGURE 3 – Accéléromètre en peigne (source tomshardware.fr)

La structure symétrique permet d'éliminer les vibrations parasites.

Centrale inertielle

Une centrale inertielle (IMU, *inertial measurement unit*) combine généralement :

- 3 accéléromètres (mesure des forces non gravitationnelles par axe)
- 3 gyroscopes (mesure de la vitesse angulaire par axe)
- 3 boussoles (mesure du champ magnétique par axe)

L'intégration permet un repère parfaitement orthogonal et un alignement parfait des capteurs.

L'IMU étant branchée sur une carte posée sur une table :

- Que mesure l'accéléromètre 3 axes au repos ?
- Où pointe le champ magnétique désigné par la boussole 3 axes au repos ?

Limites des centrales inertielles

- La boussole est peu fiable en mouvement
- Le gyroscope ajoute un biais qui évolue lentement dans le temps (basse fréquence)
- L'accéléromètre est bruité et combine la réaction à la gravité avec les autres accélérations du mobile

Ces limites ne sont pas rédhibitoires mais doivent être prises en compte lors de la conception du système.

Fusion de capteurs : un exemple

La fusion de plusieurs capteurs permet parfois d'améliorer la connaissance du monde extérieur. Pour un véhicule circulant sur un terrain plat par exemple :

- Le GPS donne une position absolue à une fréquence basse (~1Hz)
- La dérivée de la position fournit la vitesse
- L'accélération sur deux axes, disponible à une fréquence plus élevée (100Hz) fournit la variation de vitesse
- La vitesse une fois intégrée fournit la variation de position

Il est donc possible d'améliorer la connaissance de la position du véhicule en combinant le GPS et l'accéléromètre.

Fusion de capteurs : un autre exemple

Au sein d'une IMU, on peut combiner les capteurs pour obtenir des informations plus fiables par exemple pour trouver la direction du sol :

- Au repos, l'accéléromètre permet de déterminer le vecteur de gravité et donc l'orientation du mobile par rapport au plan vertical.
- En mouvement, le gyroscope permet de déterminer la variation d'orientation du mobile par rapport au plan vertical en partant de l'orientation précédente.

En combinant accéléromètre et gyroscope, on déterminera l'orientation par rapport au plan vertical de manière plus fiable.

Fusion de capteurs : comment faire ? (1/2)

Il existe plusieurs techniques, plus ou moins simples à mettre en œuvre, pour fusionner les capteurs :

- Filtre de Kalman : à partir de l'estimation de l'état d'un système, des mesures effectuées et de la connaissance des actions entreprises par les effecteurs, estime l'état suivant en minimisant les erreurs. Fonctionne pour un système linéaire, avec un modèle de bruits de mesure et d'effecteur gaussien centré en 0
- Filtre de Kalman étendu : extension du filtre de Kalman à des modèles non-linéaires

Fusion de capteurs : comment faire ? (2/2)

- Filtre complémentaire : en mélangeant les données des différents capteurs et de l'état estimé précédent avec des coefficients à déterminer, filtre le bruit sur les capteurs et estime l'état suivant
- D'autres types de fusion dédiés aux capteurs concernés, principalement si les relations ne sont pas linéaires

Ces modèles s'étudient en cours d'automatique, de traitement du signal ou lors de projets.

Effecteurs

GPIO (sortie numérique) (1/2)

- GPIO : *general-purpose input/output*
- En général, permet d'opter pour une entrée ou une sortie (effecteur)
- Même en mode sortie, il peut être utile de pouvoir lire l'entrée
- Fonctionnement général : on peut piloter la sortie vers deux niveaux (transistors NMOS et PMOS, *push-pull*) ou uniquement la forcer à l'état bas (utilisation du NMOS uniquement, collecteur ouvert ou *open-drain*)

GPIO (sortie numérique) (2/2)

- La vitesse de commutation de la sortie est limitée, et parfois configurable (STM32 par exemple), plus la vitesse est rapide, plus cela consomme d'énergie et plus les fronts sont raides et peuvent causer des perturbations électromagnétiques
- Il existe une limite de courant par GPIO et une limite de courant pour l'ensemble du circuit
- Le mode collecteur ouvert permet d'utiliser des bus de communication utilisant ce mode (comme I^2C avec une résistance de pull-up)

Exemple de GPIO sur STM32

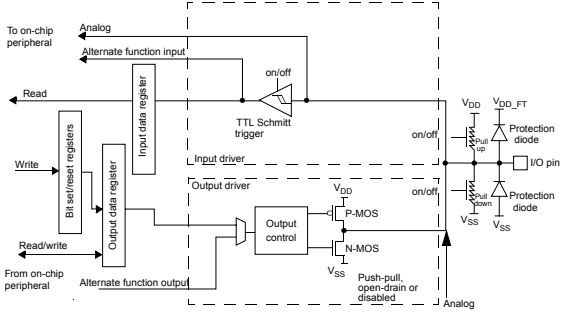

FIGURE 4 – Schéma d'un GPIO générique sur STM32 (source ST)

Dans ce schéma, déjà vu, on remarquera qu'on peut mesurer l'état du GPIO lorsqu'on le pilote en mode collecteur ouvert.

Erreurs fréquentes d'utilisation des GPIO

- Ne pas choisir une fréquence de sortie suffisante pour l'utilisation demandée
- Utiliser le mode *push-pull* (valeur imposée) depuis plusieurs composants à la fois et provoquer des courts-circuits
- Demander trop de courant pour l'ensemble du circuit à travers les GPIO ; l'utilisation de transistors externes ou de *drivers* pour piloter des charges gourmandes est recommandé

Convertisseur numérique-analogique

- *DAC* : *digital to analog converter*
- Génère un signal analogique à partir de valeurs discrètes
- Peut nécessiter d'être complété par un filtre passe-bas pour éliminer les harmoniques liés à la quantification

PWM (1/2)

- *PWM* : *pulse-width modulation*
- Principe : signal rectangulaire purement binaire 0V ou V_{DD}, périodique de période P, avec un *duty-cycle* d entre 0 et 1 proportionnel à la tension moyenne souhaitée ; $V_{avg} = V_{DD} * d$
- La partie haute du cycle peut être centrée au sein de la période, alignée à gauche ou alignée à droite

PWM (2/2)

- À partir d'une sortie purement binaire, permet après application d'un filtre passe-bas d'extraire la composante continue du signal (moyenne)
- Le signal est composé d'une composante continue et de porteuses modulées en phase autour des harmoniques de $1/P$; une fréquence élevée augmente la fréquence de coupure nécessaire pour le filtre passe-bas
- Le filtre passe-bas peut être électronique ou physique (moteur, œil, oreille)

Autres capteurs et effecteurs

- Beaucoup d'autres périphériques sont disponibles, en entrée ou en sortie (écrans, écrans tactiles, son, cartes SD)
- Ils se connectent via un bus de communication (I^2C, SPI, USB, BlueTooth)
- Plusieurs cas de figure sont possibles :
  - Le microcontrôleur gère le protocole au niveau logique et électrique
  - Le microcontrôleur gère le protocole uniquement au niveau logique : un *transceiver* sera nécessaire (Ethernet par exemple)
  - Le microcontrôleur ne gère pas le protocole : il faut soit l'implémenter logiciellement, soit utiliser une puce supplémentaire qui fera l'interface entre un protocole connu et le protocole (module WiFi par exemple)

Capteurs, effecteurs et automatique

- Piloter un système n'est souvent pas aussi simple que lire une entrée et produire une sortie
- Les systèmes en boucle ouverte sont souvent moins précis ou moins performants : on ne fait pas décoller un quadricoptère en envoyant juste un signal identique aux quatre hélices
- Les systèmes en boucle fermée (ou bouclés) calculent l'erreur entre la consigne et le résultat obtenu et pilotent les effecteurs en conséquence.
- Les systèmes bouclés peuvent présenter des états instables : oscillations, amplification incontrôlée (effet Larsen pour l'audio par exemple)
- C'est ce qu'étudie la discipline de « l'automatique »

Automatique (1/2)

L'automatique s'intéresse aux systèmes dynamiques :

- leur modélisation : comment le système réagit-il à un stimulus ou à une perturbation ?
- leur identification : quels sont les paramètres du système dynamique en présence (moment d'inertie par exemple) ?
- leur commande : quelle commande appliquer en fonction de l'état dynamique pour obtenir l'effet voulu en sortie ?

Automatique (2/2)

Les outils principaux sont les suivants :

- Les équations différentielles temporelles représentent l'évolution du système
- La transformée de Laplace permet d'obtenir une fonction de transfert plus simple à manipuler
- Pour les systèmes à temps discret, on utilise la transformée en Z

Fonction de transfert

Les fonctions de transfert permettent de représenter facilement des concepts physiques fréquemment rencontrés :

- la multiplication du signal par une constante
- un retard du signal (déphasage)
- l'intégration ou la dérivation du signal
- un filtre
- la combinaison d'autres fonctions de transfert par addition ou multiplication (enchaînement)

Lorsque la fonction de transfert est représentée sous la forme d'une division de polynômes, l'étude de ses pôles (dénominateur à 0) et de ses zéros (numérateur à 0) permet de déterminer certaines propriétés de stabilité.

Exemple de contrôle bouclé : le PID

- PID : proportionel, intégral, dérivé, correspondant aux trois coefficients K_p, K_i et K_d
- Système dynamique : on calcule l'erreur $e(t)$ entre la consigne (par exemple une position à atteindre) et le résultat (par exemple la position mesurée)
- On applique au moteur une commande de la forme

$$u(t) = K_p e(t) + K_i \int_{\tau=0}^{t} e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

- Le coefficient K_p détermine la raideur de la pente de la réaction, K_d l'amortissement à l'approche de la consigne et K_i la correction de l'erreur résiduelle
- On peut déterminer ces coefficients de manière analytique, en connaissant les propriétés du système, ou expérimentales (par exemple méthode de Ziegler & Nichols)

Autour de nous (1/2)

Les systèmes bouclés sont présent tout autour de nous :

- contrôle de vannes de radiateur pour atteindre la température cible ; peut être amélioré en caractérisant l'inertie thermique de la pièce et les transferts avec l'extérieur en connaissant la température
- contrôle d'une chaudière en addition des vannes de radiateur afin de minimiser la consommation d'énergie tout en conservant une vitesse de réaction acceptable

Autour de nous (2/2)

- véhicules autonomes ou à conduite assistée ; systèmes ABS qui maintiennent l'adhérence lors du freinage (dérapage optimal autour de 20%)
- ventilateur, machine à laver le linge, réfrigérateur, manette à retour de force, gyropode, . . .
- modèles réduits à pilotage assisté (drones, bateaux, voitures)

Pensez à une garantir une commande fiable et stable lorsque vous concevez votre système.

Fonctions internes

Agir avec l'extérieur, c'est bien, mais...

L'avantage d'un micro-contrôleur est de disposer de périphériques et interfaces en interne :

- interfaces avec les bus de communication
- compteurs, timers
- gestionnaire d'interruption
- etc.

Comment cadencer tout ça ? Grâce aux horloges.

Horloges (1/2)

Un microcontrôleur fonctionne grâce à une ou plusieurs horloges :

- HSE (*high-speed external*) : horloge rapide externe, en général fournie par un quartz ou un oscillateur, mais également par un autre circuit via une sortie d'horloge (*clock out*)
- HSI (*high-speed internal*) : horloge rapide interne, plus ou moins précise, souvent sensible aux variations de température à l'intérieur du boîtier
- LSE (*low-speed external*) : horloge lente externe, généralement utilisée lors des modes de veille pour préserver quelques fonctionnalités comme la RTC (*real-time clock*)
- LSI (*low-speed internal*) : la LSI est à la LSE ce que la HSI est à la HSE
- des horloges externes dédiées pour certains périphériques (USB, Ethernet) qui ont des exigences fortes
- des horloges dérivées en interne d'autres horloges

Horloges (2/2)

La création des horloges dérivées se fait :

- en divisant la fréquence d'une horloge existante en ne considérant que certains fronts
- en multipliant la fréquence d'une horloge existante en utilisant une PLL (*phase locked loop*), construite à partir d'un VCO en boucle fermée (tiens donc !)

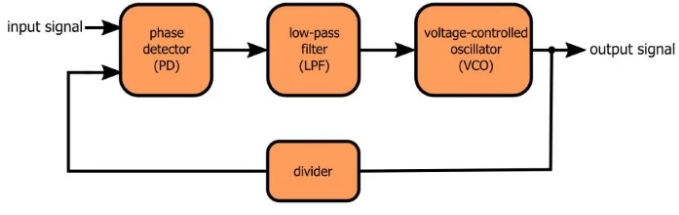

FIGURE 5 – PLL (source R. Keim, all about circuits)

Le VCO contrôlé sans rétroaction ne serait pas assez précis.

Arbre d'horloges sur un STM32 (1/2)

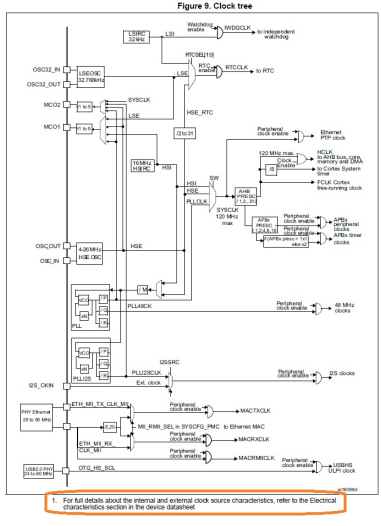

FIGURE 6 – Exemple d'arbre d'horloges

Arbre d'horloges sur un STM32 (2/2)

On notera que :

- HSE et HSI peuvent être utilisées comme source d'une PLL avec diviseurs pour produire PLLCLK et PLL48CK (à 48MHz, pour l'USB)
- L'horloge système SYSCLK est choisie entre HSE, HSI et PLLCLK
- Des horloges sont générées à partir de SYSCLK pour cadencer les bus AHB (*advanced high-performance bus*) et APB (*advanced peripheral bus*)
- LSE, LSIRC ou une division de HCE peut être utilisé pour cadencer la partie RTC
- LSIRC est systématiquement utilisée pour cadencer le watchdog indépendent (retenez cela, on en reparlera un peu plus tard)
- Il est possible d'activer ou désactiver des domaines d'horloge au besoin

Et surtout, comme indiqué en bas du schéma, il faut vérifier les caractéristiques dans la documentation (*datasheet*) du microcontrôleur concerné. Certaines vitesses ou plages de vitesse sont imposés pour un fonctionnement satisfaisant du microcontrôleur.

Démarrage typique d'un système avec HSE

1. Le microcontrôleur démarre avec HSI comme source d'horloge système
2. Le code active HSE ; une fois que HSE est stable, un bit est positionné automatiquement
3. Le code choisit HSE comme source de la PLL ; une fois que la PLL est stable, un bit est positionné automatiquement
4. Le code configure les *wait states* à utiliser pour la flash pour la vitesse du système choisie
5. Le code choisit la sortie de la PLL comme horloge système

Une défaillance de HSE au démarrage peut être détectée lors de l'étape 2 si, par exemple, celle-ci ne devient jamais stable.

Et si cela arrive en cours de fonctionnement ? Un peu de patience, revenons aux interfaces et nous verrons cela plus tard.

Interfaces avec les bus de communication (1/2)

- Le microcontrôleur implémente la machine à états du protocole en s'appuyant sur l'horloge appropriée
- Par exemple, pour échanger un mot avec un esclave en SPI, il faut
  - forcer le signal SS (*slave select*, ou CS pour *chip select*) à l'état bas
  - générer un signal d'horloge SCLK autant de fois qu'il y a de bits dans un mot (en fonction de la configuration de ce bus SPI)
  - au fur et à mesure présenter les bits du mot à transmettre sur MOSI (*master out slave in*) dans l'ordre et à l'instant prévu par la configuration
  - en même temps lire les bits du mot entrant sur MISO (*master in slave out*)
  - placer le résultat dans un register à disposition de l'utilisateur et lui signaler par une interruption que le mot est disponible
  - remonter le signal SS

Interfaces avec les bus de communication (2/2)

Les fonctions peuvent même être plus évoluées :

- Le microcontrôleur peut transmettre plus d'un mot à la fois en allant chercher le contenu à transmettre en DMA (*direct memory access*)
- Ce contenu peut être placé dans un buffer circulaire, le microcontrôleur signale lorsqu'il arrive à la moitié de ce buffer ou à la fin, ce qui permet son remplissage concurrent
- Les mêmes fonctionnalités sont accessibles pour la partie réception

Interfaces : avantages

- Le CPU est déchargé des détails de la machine à états
- La vitesse d'échange peut être plus rapide qu'en écrivant la machine à états en logiciel
- Dans le cas de l'utilisation de DMA, un très grand nombre d'opérations (par exemple l'envoi d'un buffer de 1500 caractères à une interface réseau par exemple) peut être faite en une seule fois en tâche de fond
- Lorsque c'est applicable, l'interface peut être certifiée conforme à une norme existante, ce qui permet de s'afficher comme officiellement compatible sans besoin de passer une certification séparée

Timer

Les timers fournissent des sources de temps et d'interruption :

- Ils ont une précision donnée (typiquement 8, 16, 32 ou 64 bits)
- Ils comptent ou décomptent en fonction d'une division de la vitesse d'horloge
- Ils peuvent être en mode *free-running* (comptage non-stop) ou être reliés à un comparateur
- Ils peuvent s'arrêter ou continuer en se remettant à leur valeur initiale lorsque la valeur cible est atteinte
- Ils peuvent refléter le résultat de la comparaison ou son complémentaire sur une sortie physique
- Ils peuvent être combinés pour générer des signaux pour ponts en H
- Ils peuvent commencer à compter à partir du changement d'état d'une entrée

Watchdog (1/2)

Le watchdog est un timer spécial, autonome, qui s'assure que le système n'entre pas dans un état irrécupérable :

- Une fois déclenché, le watchdog ne peut plus être arrêté
- Il faut signaler régulièrement au watchdog que le système se porte bien
- Si on dépasse le délai fixé (*timeout*), le système redémarre ; en général, un registre permet de connaître la cause du redémarrage (reset externe, watchdog, *brown out*, redémarrage logiciel)
- Certains watchdogs obligent à respecter une fenêtre (*window*) de temps entre deux signalements
- Certaines interfaces de débogage peuvent suspendre le watchdog si la configuration de celui-ci le permet

Le redémarrage n'est pas forcément une solution idéale, mais il permet de remettre le système dans un état stable et connu.

Watchdog (2/2)

Il est possible d'utiliser un watchdog externe si le microcontrôleur choisi n'en dispose pas en interne.

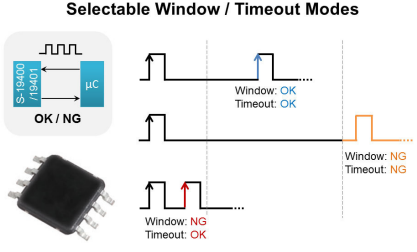

FIGURE 7 – Watchdog externe (source Business Wire)

Un système dans un état inconnu est inutile voire nuisible. Si tout se passe bien, le watchdog ne servira jamais, dans le cas contraire, il est indispensable.

Watchdogs multiples (1/2)

Certains STM32 disposent de deux watchdogs :

- Un watchdog indépendent (*IWDG*, *independent watchdog*) simple qui est cadencé grâce à son horloge indépendente du reste des horloges du microcontrôleur (circuit RC)
- Un watchdog fenêtré, qui utilise un compteur connecté à l'arbre d'horloges interne et peut déclencher une interruption avant de se déclencher (et il peut être signalé à ce moment)

Question : en production, pourquoi peut-il être utile d'utiliser les deux watchdogs en même temps ?

Watchdogs multiples (2/2)

Réponse (mais vous aviez déjà trouvé) :

- Si jamais l'horloge externe (HSE) défaille, le système arrête d'avancer ou ralentit exagérément
- Malheureusement, le watchdog fenêtré suivra ce rythme
- À l'inverse, le watchdog indépendant continuera de fonctionner
- Après un redémarrage, il sera possible de remarquer à l'initialisation du système le manque de stabilité de HSE et de continuer à fonctionner sur HSI, ne serait-ce que pour signaler la défaillance à l'utilisateur ou, en présence d'un serveur, au système de supervision

Et si on parlait de quand ça se passe bien ?

Quand ça se passe bien, on a quand-même envie d'en faire le moins possible. Plutôt que de surveiller un port d'entrée ou le bit d'un registre de statut, on préfère recevoir une interruption.

- La survenance d'une interruption (IRQ, *interrupt request*) provoque la mise à 1 d'un bit *pending* correspondant à cette IRQ
- Entre chaque instruction, le cœur du microcontrôleur vérifie s'il y a une interruption à traiter, et si oui la traite
- Il est possible de masquer une interruption pour inhiber son traitement, puis de la démasquer pour qu'il ait lieu
- Certaines interruptions sont non masquables (NMI, *non-maskable interrupts*)

Conditions de traitement d'une IRQ

Entre deux instructions, le cœur du microcontrôleur

- détermine l'interruption *pending* non masquée la plus prioritaire (priorités configurables ou non)
- vérifie que sa priorité est compatible avec l'état actuel du système
  - Certains systèmes permettent de modifier le niveau de priorité minimal
  - Certains disposent d'un seul bit pour masquer ou non toutes les interruptions
- vérifie, si une interruption est déjà en cours de traitement, que la nouvelle a le droit de l'interrompre

Si ces conditions sont réunies, au lieu d'exécuter l'instruction suivante, le cœur du microcontrôleur exécute la routine de traitement d'interruption (ISR, *interrupt service routine*) appropriée après avoir sauvegardé une partie du contexte sur la pile (ou une pile dédiée).

ISR et retour

En général, l'*interrupt service routine* se trouve à une adresse trouvée dans une table de vecteurs configurable par l'utilisateur.

Selon les architectures

- Il est nécessaire ou non de nettoyer le bit *pending* de l'interruption dans l'ISR (l'acquitter), sous peine de rerentrer immédiatement dans l'ISR ; certains systèmes le font automatiquement
- Il faut utiliser ou non une instruction spéciale pour revenir de l'ISR plutôt que de l'instruction de retour usuelle
- Si au retour d'une ISR une autre ISR (ou la même) doit être exécutée immédiatement, certaines architectures permettent d'éviter la restauration et la resauvegarde immédiate du context depuis la pile (*interrupt tail-chaining*, ou *back-to-back processing*)

IRQ composites

Dans certains cas, une même IRQ est générée à partir de plusieurs sources :

- Une même IRQ peut être générée si un port série est prêt à envoyer un caractère ou s'il a reçu un caractère ;
- Un contrôleur d'interruptions secondaire peut multiplexer plusieurs sources d'interruption et génère une IRQ commune

Dans ce cas, il est indispensable d'acquitter la vraie source de l'IRQ car même un cœur de microcontrôleur avec acquittement automatique lors du retour de l'ISR ne peut pas deviner quelles sources ont été réellement prises en compte lors de cette exécution de l'ISR, surtout si des sources additionnelles ont été déclenchées pendant l'exécution de l'ISR.

Et maintenant ?

Avec tous ces périphériques, internes ou externes, on fait quoi ? Il faut écrire du code pour traiter les données.

Stockage : mémoire non volatile

Mémoire non volatile

- Sert à stocker
  - le programme (firmware)
  - ses données
  - des données liées à l'appareil (calibration, configuration)
  - avec une alimentation, des données changeantes (horloge temps-réel ou RTC pour *real-time clock*) ou qu'on ne souhaite pas inscrire sur un support (clés cryptographiques)
- Types de NVM (*non volatile memory*)
  - mask-ROM / OTP-ROM
  - EPROM / EEPROM
  - mémoire flash / carte SD/MMC/etc. / SSD
  - RAM + batterie

Nous allons nous intéresser à la mémoire flash.

Mémoire flash

La mémoire flash est la mémoire volatile la plus couramment rencontrée.

- Principe de base : un transistor MOS avec une grille flottante (isolée) entre la grille classique et le canal drain-source
- Lorsque la grille flottante est chargée en électrons, elle isole partiellement la grille classique, la tension de commande usuelle est ignorée et le transistor n'est pas passant ; il faut une tension de commande plus élevée pour le rendre passant.

Mémoire flash

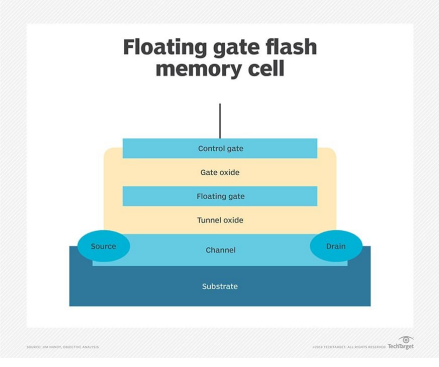

FIGURE 8 – Cellule de mémoire flash (source Kim Mandy, Objective Analysis)

Cellule flash : programmation et effacement

Exemple de chargement d'électrons par effet tunnel

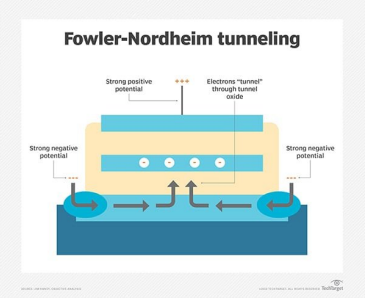

FIGURE 9 – Programmation (mise à 0) par effet tunnel (source K. Mandy, *op cit*)

Le déchargement est similaire, en inversant les tensions.

Cellule flash : fonctionnement

Le transistor étant commandé avec les tensions usuelles :

- si aucun électron n'est stocké dans la grille flottante, la grille classique commande le transistor et peut le rendre passant (état 1, ou effacé)
- si des électrons sont stockés dans la grille flottante, la grille classique est isolée (la tension de seuil est devenue plus élevée) et le transistor ne devient jamais passant (état 0, ou programmé)
- dans tous les cas, une tension plus élevée peut le rendre passant

Note : avec un contrôle précis de la charge de la grille flottante, on peut stocker plus d'un bit en testant ensuite le transistor avec des tensions de grille différentes.

Mémoire flash : organisation physique

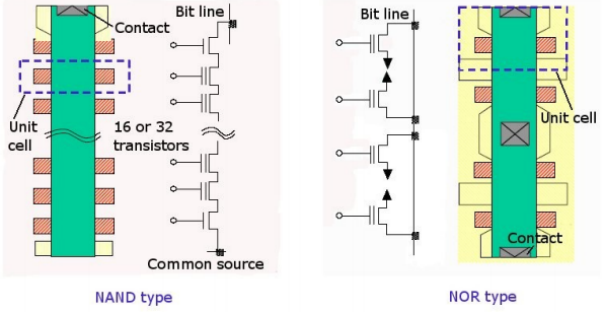

FIGURE 10 – Flash NAND et NOR (source Ridgerun)

Mémoire flash : organisation physique

- La flash NOR permet un aspect direct à chaque bit par une interface électrique similaire à une interface mémoire
- La flash NAND est beaucoup plus dense mais nécessite un contrôleur dédié pour lire son contenu
- Certains microcontrôleurs intègrent une interface bus vers ce type de contrôleur et peuvent booter sur de la flash NAND externe
- La flash NAND peut contenir des erreurs et utilise des bits supplémentaires pour contrôler l'intégrité de chaque mot

Mémoire flash NAND : utilisation (1/2)

- La mémoire flash est découpée en blocs (de tailles identiques ou non)
- Le bloc est la plus petite unité effaçable
- Chaque bloc est découpé en pages (de tailles identiques ou non)
- La programmation se fait par pages entières depuis un buffer intermédiaire en RAM vers une page non utilisée

Comment faire pour réécrire une page utilisée sans altérer le reste du contenu du bloc ?

Mémoire flash NAND : utilisation (2/2)

- Consequence : pour écrire de nouveau une page utilisée, il faut sauvegarder le contenu des autres pages du même bloc afin de pouvoir l'effacer
- La lecture se fait par pages entières vers un buffer intermédiaire en RAM
- Des octets supplémentaires (souvent 1/32) sont disponibles dans chaque page (par exemple 512+16 ou 2048+64) pour permettre le contrôle de l'intégrité

Mémoire flash NOR : utilisation

- La mémoire flash est découpée en blocs (de tailles identiques ou non)
- Le bloc est la plus petite unité effaçable
- Chaque bloc est découpé en pages (des fois appelés secteurs) (de tailles identiques ou non)
- La programmation se fait octet par octet, ou même bit à bit : les bits ne pouvant passer que de l'état 1 (effacé) à 0 et chaque bit étant adressable directement, écrire un 1 ne change pas l'état, écrire un 0 force un 0

Comparaison flash NAND / flash NOR

propriété	**flash NAND**	**flash NOR**
lecture	par page	aléatoire
écriture	rapide (par page)	lente (mot par mot)
effacement	lent	très lent
fiabilité	ECC (facteur typique 1/32), fautes par pages	pas d'ECC mais fautes individuelles
utilisation	données, code si supporté	code (interface SRAM)
coût	modéré	élevé

Précautions à prendre

- La mémoire flash s'use : le nombre d'écriture supporté se trouve dans le documentation
- Les techniques de *wear leveling* (répartition de l'usure) permettent de ne pas réécrire sans cesse les mêmes blocs
- Des systèmes de fichier dédiés intègrent le *wear leveling* (JFFS2 par exemple)
- Certains contrôleurs de flash gèrent seuls le *wear leveling* et la gestion des blocs avec erreur en les remplaçant par des blocs de secours libres
- Du code reprogrammant la flash ne peut pas toujours s'exécuter depuis la mémoire flash, même s'il s'agit de blocs différents (recopie en RAM nécessaire de code relocalisable)

Programmation de la mémoire flash en production

Lorsque vous livrez votre produit à vos consommateurs, la mémoire flash (qui contient notamment le code) doit avoir été programmée. Cela peut se faire

- avant montage, avec des programmateurs ; gestion des flux et des stocks compliquée, mise à jour compliquée
- après montage, *in situ*
  - avec une interface JTAG, SWD ou équivalent : peu cher mais lent
  - avec un lit à clous : coûts fixes élevés, rapide, peut se combiner avec les tests électriques de la carte et permet également des tests fonctionnels du logiciel
  - dans tous les cas, doit être prévu dès la conception de la carte (schémas, placement, routage)

Autres solutions de stockage externe

La mémoire flash interne au microcontrôleur ou connectée à un bus dédié n'est pas la seule solution pour ajouter de la mémoire non volatile à un système :

- sur la carte, mémoire flash externe en I²C ou SPI
- carte SD équivalent, amovible ou non
- SSD (interface SATA ou NVMe)
- disque dur ou SDD (interface SATA ou à travers un bus USB)
- stockage réseau
- etc.

Stockage : mémoire volatile

RAM interne

Les microcontrôleurs intègrent de la RAM interne

- en quantité limitée (de quelques dizaines d'octets jusqu'au Go)
- parfois en plusieurs RAM avec des propriétés différentes, comme :
  - nombre de cycles pour l'accès
  - accessible ou non au contrôleur DMA
  - utilisable ou non pour les buffers de certains périphériques (USB)
  - alimentée ou non en fonction des modes de veille du microcontrôleur

L'utilisation judicieuse de la RAM, notamment grâce à l'utilisation des sections et un *linker script* approprié, permet d'améliorer les performances et le rendement énergétique.

Caches

Les microcontrôleurs peuvent également intégrer de la mémoire cache

- pour accélérer les accès successifs à de la mémoire plus lente (RAM, flash)
- pour précharger du code ou des données
- lorsque l'interface le nécessite (flash NAND)

Parfois la mémoire cache est également utilisable comme mémoire généraliste.

RAM externe

Il est possible de rajouter de la RAM externe à un microcontrôleur

- à travers une interface de bus dédiée lorsqu'elle existe
- à travers une interface généraliste comme SPI sinon

La RAM externe sera généralement beaucoup plus lente que la RAM interne. Il vaut mieux l'éviter lorsque c'est possible.

RAM externe (suite)

Ceci n'est pas un cours sur les RAM, mais il faut savoir que

- il existe plusieurs sortes de RAM (dynamique, statique)
- leur mode d'accès varie (synchrone, asynchrone, avec ou sans mode burst, avec ou sans retournement de bus)
- leur interface également (simple port, multi-ports)

Lorsque cela sera nécessaire (applications vidéo, traitement du son, big data) sachez qu'il vous faudra chercher un cours sur les RAM.

Choix du micro-contrôleur

Contexte

- On s'intéressera ici aux systèmes embarqués à base de microcontrôleurs
- Certains « gros » systèmes ressemblent plus à des mini-PC
- Dans certains cas, pour des prototypes notamment, on fera appel à des cartes existantes remplissant les bonnes caractéristiques plutôt que de construire la nôtre

Caractéristiques des microcontrôleurs

- Archicture, performance et jeu d'instructions (calcul flottant, DSP)
- Taille des mémoires (flash, RAM, caches)
- Périphériques intégrés, interfaces externes, sources d'interruption
- Outils de développement (langages, debug)
- Sécurité (protection mémoire, gestion des secrets, TrustZone)
- Prix et disponibilité
- Tensions d'alimentation, consommation, mise en veille
- Type de boîtier, facilité de soudage
- Impact sur l'environnement (rarement pris en compte)

Performances et jeu d'instructions

Il est nécessaire d'avoir les capacités de faire les calculs requis

- Les processeurs avec unité flottante sont plus chers
- Faire des calculs en virgule flottante sans unité flottante demande plus de temps et d'énergie
- Les instructions DSP (*multiply-accumulate*, arithmétique saturée) font gagner beaucoup de cycles mais complexifient le processeur

Architecture

- Choix entre architecture RISC (ARM, AVR, MIPS, PIC, RISC V) ou CISC (Intel)
- Selon le langage de programmation choisi, on peut avoir besoin de peu de registres (Forth, Java, langages à pile interprétés, donc CISC suffit) ou de beaucoup de registres (RISC)
- Bref : pour de l'IoT, 99%+ de chances de choisir un processeur RISC

Nous allons faire un focus particulier sur les cœurs Cortex-M de ARM.

Architecture : ARM (1/2)

- ARM : société britannique appartenant à SoftBank / NVIDIA (septembre 2020)
- Cœurs de 32 ou 64 bits, license vendue par ARM pour intégration, microcontrôleurs construits par d'autres (NXP, ST)
- Équipent la majorité des téléphones mobiles et quelques ordinateurs portables

Architecture : ARM (2/2)

- Se décline en plusieurs gammes et famille, par exemple les Cortex
  - Cortex A : 32 bits ou 64 bits, MMU, multicœurs, unité mathématique puissante, pour ordinateur (dont les futurs Apple), TV, téléphone mobile, etc.
  - Cortex R : gamme temps-réel avec des cœurs redondants et des latences garanties, domaines médical ou du transport
  - Cortex M : 32 bits, dédiés à l'embarqué à faible coût et faible consommation, avec ou sans FPU, avec ou sans instructions DSP

ARM : Cortex-M

Dans notre domaine (IoT), nous nous intéresserons particulièrement aux cœurs Cortex-M :

- Architecture ARMv6-M : Cortex-M0 (le plus petit), Cortex-M0+ (le plus efficace énergiquement), Cortex-M1 (pour intégration dans des FPGA)
- Architecture ARMv7-M : Cortex-M3 (milieu de gamme), Cortex-M4 (avec DSP et FPU facultatif), Cortex-M7 (encore plus puissant, avec SIMD et arithmétique saturée)
- Architecture ARMv8-M : Cortex-M23 (le plus petit avec TrustZone), Cortex-M33 (plus puissant), Cortex-M35P (avec protections matérielles contre les attaques), Cortex-M55 (calcul vectorisé)

Nous utiliserons un Cortex-M4 en SE302a et un Cortex-M33 en SE302b.

ARM : quelques propriétés des Cortex-M

Les Cortex-M nous intéressent

- pour leur haute densité de code (jeu d'instructions thumb 2)
- pour leurs interruptions rapides, priorisées et chaînées
- pour leur support interne des systèmes d'exploitation (`SVC`, `PendSV`)
- pour leur faible coût
- pour la bonne intégration avec les outils de développement libres (GCC, GDB, Rust)

Cortex-M : tous les constructeurs ne se valent pas

Des microcontrôleurs à base de Cortex-M sont disponibles chez plusieurs constructeurs. Pour choisir un modèle, il faut aussi regarder, en plus des autres critères listés :

- la souplesse d'utilisation et les performances du contrôleur d'interruption externe
- la capacité de router les périphériques internes du microcontrôleur sur les broches du microcontrôleur

Nous utiliserons un microcontrôleur ST en SE302a, un microcontrôleur NXP en SE302b et un microcontrôleur Nordic pour certains projets de fin d'études.

Quelques autres microcontrôleurs pour l'embarqué

- 68HCxxx (Motorola) : CISC, historique
- AVR (Atmel) : riches en périphériques, utilisés notamment dans les Arduino
- MIPS (Imagination Technologies) : 32 bits ou 64 bits, basse consommation
- PowerPC (IBM) : utilisé dans les missions vers Mars
- SuperH (Renesas) : RISC, utilisé historiquement pour le multi-media dans l'automobile, remplacé progressivement par ARM (notamment Android)
- x86, x86_64 (Intel) : CISC, peu utilisé dans l'embarqué

Mais tout ça, on le met dans quoi ?

Les microcontrôleurs viennent dans des boîtiers (*packages*) variés destinés à

- assurer la jonction électrique avec l'extérieur
- assurer la dissipation thermique
- protéger la puce de l'environnement (chocs, poussière, rayonnements)
- adapter le composant aux contraintes de fabrication
- détecter et empêcher les attaques physiques, possiblement en collaboration avec le microcontrôleur lui-même

Le boîtier

Influe sur :

- le routage
- le nombre de couches
- le soudage
- la dissipation

Par exemple, on évitera pour des projets locaux d'utiliser des boîtiers BGA (*ball grid array*) ou disposant d'un pad thermique en dessous, car on ne dispose pas de l'équipement permettant de vérifier la qualité des soudures aux rayons X.

Et on l'alimente comment ?

Les boîtiers nécessitent

- une ou plusieurs tensions d'alimentation, fixes ou dans une plage
- des tensions stables, sous peine de *brown out* en cas d'écroulement de l'alimentation
- de capacités de découplage (réserve d'énergie), par boîtier et par broche d'alimentation

On prendra particulièrement soin de ne pas mélanger les alimentations des parties numérique, analogique et de puissance, sous peine de propager les perturbations de l'une à l'autre.

La qualité de l'alimentation est un des aspects les plus négligés (à tort !) des systèmes embarqués.

Et on le programme comment ?

Il faut s'assurer

- du support du ou des langages de programmation envisagés
- de la disponibilité des outils de développement sur les plates-formes qu'on utilise
- du besoin d'utiliser des logiciels propriétaires pour configurer, programmer ou deboguer le système

Mais quel langage de programmation choisir ?

Les langages de l'embarqué

D'innombrables langages sont utilisables :

- C et assembleur sont omniprésents mais offrent de nombreuses possibilités pour faire des erreurs
- C++ est fréquemment utilisé, mais fait peur en raison de sa complexité et du *dispatching* dynamique
- Ada et Rust offrent de nombreuses garanties pour éviter les erreurs
- Java est peu utilisé
- Python (et micro-python) permettent le développement rapide au prix des performances
- etc.

Le tout est d'être à l'aise, de disposer des bibliothèques requises, et si possible de se faire plaisir.

Le cours SE302b et certains projets seront l'occasion d'expérimenter la programmation embarquée avec un autre langage de programmation que le C (Rust).

Et maintenant ?

Conclusion

- De nombreux paramètres doivent être pris en compte lors de la conception d'un système embarqué
- Ce cours ne fait qu'effleurer un sous-ensemble des sujets importants
- Ce qu'il faut retenir, c'est qu'aucun choix n'est simple, mais que les solutions existent

FIGURE 11 – L'heure des choix