11 Et pour aller encore plus loin ?

Le but de ce guide est de vous emmener le plus loin possible dans la maitrise des cartes à microcontrôleur et les chapitres précédents ont bien rempli ce rôle. Mais le chemin ne s'arrête jamais et il y a toujours un domaine à explorer. C'est ce que je vais vous démontrer maintenant tout en vous proposant un voyage fictif vers un futur qui ne l'est peut-être pas tant que cela.

11.1 / S'INTÉRESSER À CE QU'IL Y A SOUS LE CAPOT

Il n'est pas question, dans le cadre de ce guide, de rentrer dans les détails mais il n'est pas inutile de donner quelques points de repère de ce qui constitue un microcontrôleur (ou micro, noté µC). Vous comprendrez pourquoi un programme écrit pour une carte ne fonctionne pas avec une autre. Peut-être même aurez-vous envie d'approfondir vous-même certains des points que je vais aborder.

Pour comprendre la structure d'un microcontrôleur, on peut comparer ce dernier à une maison. Il y a de petites maisons (µC à 8 bits) et de grandes maisons (µC à 32 bits), mais ce qu'on trouve à l'intérieur est toujours la même chose : une cuisine où on va préparer les repas en suivant une recette, un salon ou une salle à manger, des chambres, une ou plusieurs salles de bain, des placards pour ranger ses affaires, d'autres pièces spécialisées (buanderie, atelier, garage). La maison a aussi des portes et des fenêtres qui communiquent avec l'extérieur, et une horloge bien placée pour que chacun puisse voir le temps qui passe. Le micro, c'est un peu pareil. Il a aussi une **horloge** qui délivre des signaux carrés et chaque créneau donne le rythme de travail. Il a des placards qui lui servent à ranger des informations, ce qu'on appelle **mémoire**, une cuisine qui lui sert à effectuer sa tâche en suivant une recette appelée programme avec des ingrédients bien rangés dans des placards, appelés **instructions ou données**. Le micro a également des **unités spécialisées** qui servent à certaines tâches bien spécifiques, tout comme une buanderie peut être équipée d'une machine à laver le linge (qui rend le linge propre) et d'un sèche-linge (qui rend le linge sec). Enfin, pour communiquer avec l'extérieur, le micro a aussi des portes appelées **PORT**, chacun étant constitué d'un ensemble de lignes d'entrée-sortie ; lorsque la porte d'une maison est ouverte, c'est pour laisser entrer ou sortir quelqu'un, et de la même façon, **un PORT est soit une entrée soit une sortie pour une information**. De même qu'une maison peut avoir une porte principale et des portes fenêtres qui donnent sur le jardin, le micro a **plusieurs PORT qui communiquent avec l'extérieur** (capteurs ou actionneurs). On va maintenant voir en détail chacun des sous-ensembles qui constituent un microcontrôleur, tout en gardant à l'esprit notre analogie avec une maison.

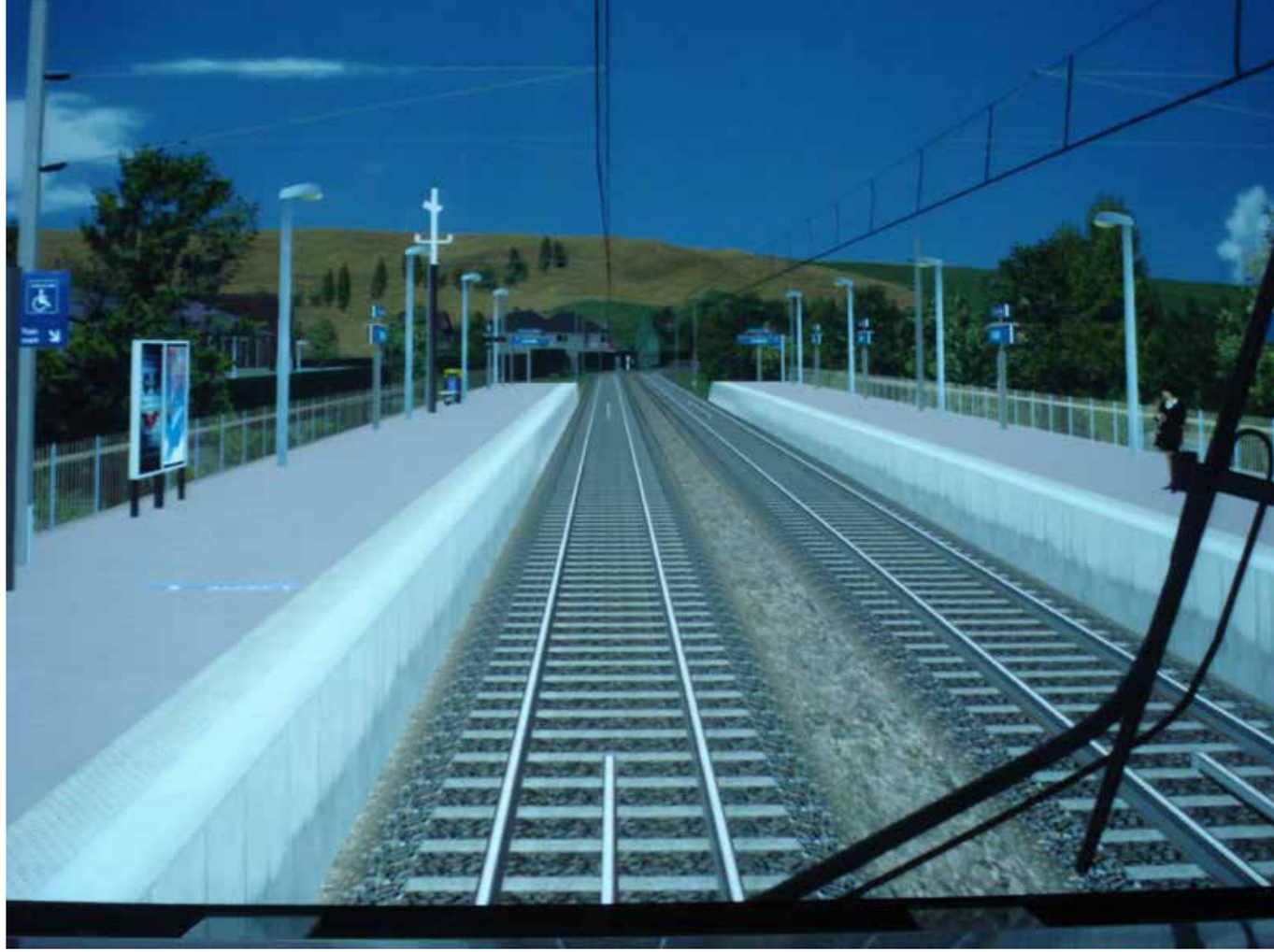

Image de conduite d'un simulateur réel à la SNCF

Un microcontrôleur est constitué de plusieurs unités, chacune spécialisée dans une tâche, et travaillant ensemble (**figure 11-1** reprenant la structure de l'ATmega328P). L'**ALU** (Arithmetic and Logic Unit) est l'unité qui est capable d'effectuer des **calculs arithmétiques et logiques** sur les octets de données (addition, soustraction, incrémentation, ET, OU, NOT, etc.). C'est le programme qui indique à l'ALU quelles opérations effectuer et sur quelles données. L'ALU est en quelque sorte la cuisine, car c'est là qu'on travaille pour préparer un plat (une tâche) avec des ingrédients (**variables**) en suivant une recette (**programme**).

Le programme étant constitué d'instructions, il faut stocker ces dernières ainsi que les données. Pour ranger ses affaires dans une maison, on dispose de placards qui peuvent avoir plusieurs formes (penderie, commode, etc.). De même, les affaires de Madame ne sont généralement pas au même endroit que les affaires de Monsieur, mais cette règle est un peu sexiste et on pourrait tout à fait mettre tous les vêtements dans la même penderie. Pour un micro, les affaires sont les informations qui sont de deux natures : **instructions et données**. Pour les stocker, c'est le rôle de la mémoire qui peut être **permanente ou bien volatile**. Le micro a une architecture **Von Neumann** si les données et les instructions sont stockées dans un **unique espace mémoire** (lui-même constitué de mémoire permanente ou volatile) : il n'y a donc qu'un seul bus d'adresse permettant de localiser le bon octet au bon endroit. Le micro a une architecture **Harvard** si la **mémoire pour les instructions est séparée de la mémoire pour les données** : c'est le cas pour l'ATmega328P des cartes Uno et il y a deux bus d'adresses puisque les espaces de stockage sont différents. Les données sont stockées en **mémoire volatile (SRAM)** alors que les instructions sont stockées en **mémoire permanente (Flash memory)** pour ne pas avoir à recharger le programme à chaque utilisation.

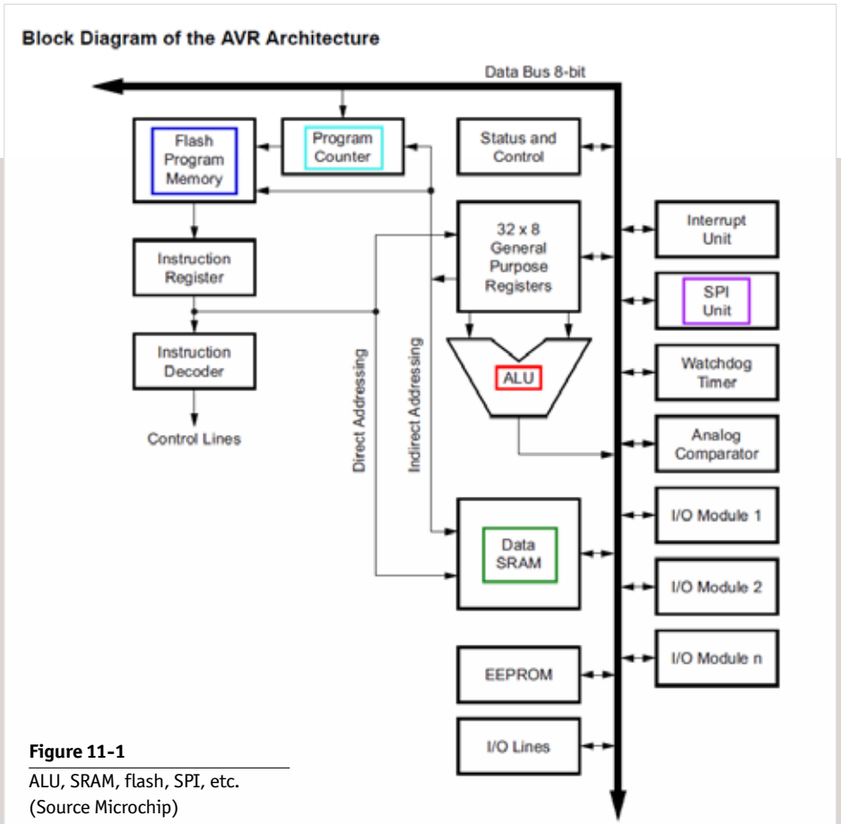

Figure 11-1
ALU, SRAM, flash, SPI, etc.
(Source Microchip)

Le **program counter** est un compteur qui permet de repérer l'instruction en cours, un peu comme si nos recettes de cuisines avaient des lignes numérotées. Lorsqu'une instruction a été exécutée, le program counter est incrémenté d'une unité et on va chercher l'instruction suivante pour l'exécuter elle aussi. En cas de saut vers un sous-programme, **il faut le sauvegarder** de manière à reprendre le programme principal (là où on s'était arrêté) lorsque le sous-programme est entièrement exécuté.

Une maison dispose d'une sonnette, et lorsque le facteur sonne, vous interrompez ce que vous êtes en train de faire pour aller réceptionner le courrier, et lorsque c'est fait, vous reprenez ce que vous étiez en train de faire. L**e micro aussi peut être interrompu dans sa tâche pour aller exécuter une tâche encore plus urgente** ; c'est ce qu'on appelle une **interruption** et dans ce cas, il faut aussi sauvegarder le program counter ainsi que d'autres registres importants. Parfois, on a envie d'être tranquille et on inhibe la sonnette de la maison ; on peut faire la même chose et **inhiber les interruptions** si la tâche qu'effectue le micro ne doit pas être interrompue (respect d'un timing précis par exemple). Tout cela est à régler dans les différents registres du micro et il faut donc apprendre à les connaître.

Comme un micro doit communiquer avec l'extérieur, **ces broches sont organisées en PORT** avec différents registres pour les contrôler : ce sont les portes fenêtres de notre maison avec les serrures pour les ouvrir ou les fermer. Tant qu'on utilise les fonctions d'Arduino (digitalWrite par exemple), le logiciel IDE sait que telle broche correspond à tel registre et le programmeur n'a à s'occuper de rien. Mais on peut aussi travailler directement avec les registres des PORT, ce que j'avais fait dans le tome 1 pour la croix de pharmacie. Et comme les PORT ne sont pas organisés de la même façon d'un micro à un autre, **un programme qui fonctionne sur Uno ne fonctionnera plus sur Mega** ! Pour la croix de pharmacie, j'utilisais le PORTD de la carte Uno qui correspond aux broches 0 à 7. Sur la carte Mega, les broches 0 à 7 font partie des PORT E, G et H, comme le montre la **figure 11-2** tirée de la documentation « pinout » des cartes.

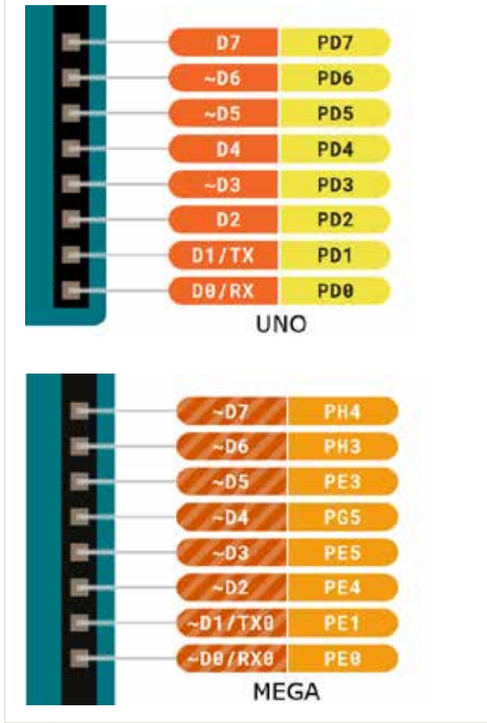

Figure 11-2

Un micro possède aussi des **timers** pour compter le temps (tout comme il y a plusieurs pendules dans une maison) et on peut les programmer en écrivant directement dans les registres qui leur sont associés. Les timers ont la propriété d'effectuer leur **comptage du temps indépendamment du programme, en tâche de fond**. Mais d'un micro à l'autre, les timers ne sont pas les mêmes, ce qui peut encore expliquer qu'un programme pour une carte ne fonctionne pas sur une autre. Pour éviter ces désagréments, le mieux est soit de respecter les fonctions d'Arduino, soit de passer

par des bibliothèques. Une bibliothèque n'est pas pour autant universelle : quand on l'utilise, on doit se renseigner sur les cartes qu'elle prend en charge. Par exemple, Servo ne fonctionne pas sur une carte ESP et il faut charger la bibliothèque ESP32Servo, comme on l'a vu un peu plus haut. Certaines fonctions (ou bibliothèques) d'Arduino utilisent les timers, ce qui peut inhiber d'autres tâches : **il faut bien regarder la documentation des fonctions ou bibliothèques**. Par exemple, Servo empêche de produire de la PWM sur les broches 9 et 10, qu'un servomoteur y soit branché ou non.

Un timer a un rôle particulier : le **WDT pour Watch Dog Timer ou encore chien de garde**. Dans un programme, on peut l'utiliser ou pas, et si on l'utilise, son rôle est d'éviter qu'un programme soit bloqué parce qu'il tourne en rond (boucle sans fin, blocage dû à un parasite). Le principe est simple, le WDT décompte son compteur et s'il arrive à zéro, il effectue un reset de la carte. Pour que le programme fonctionne normalement, il faut relancer la minuterie régulièrement pour l'empêcher d'arriver à zéro. Dans une boucle sans fin, on ne relance plus la minuterie donc le WDT finit par arriver à zéro : un reset a lieu et le programme peut redémarrer et s'exécuter normalement.

D'autres unités spécialisées existent au sein du micro (l'équivalent de la buanderie d'une maison), **par exemple pour communiquer** (SPI, I2C, UART, WiFi, Bluetooth, etc.) ou **pour produire de la PWM** ou encore pour **transformer un signal analogique en numérique ou réciproquement**. Pour chaque unité, on trouve des registres de contrôle dont il faut connaître l'usage. Le mieux est de se plonger dans la datasheet, mais certaines font plusieurs centaines de pages. Tout cela ne se justifie que pour des besoins très particuliers : en modélisme ferroviaire, l'utilisation des fonctions d'Arduino suffit amplement, et si ma croix de pharmacie a utilisé le PORTD, c'est juste pour rendre le programme plus lisible (je gagne aussi en temps d'exécution mais cela ne se voit pas à l'œil nu).

Une dernière chose à avoir en tête, c'est que la mémoire d'un microcontrôleur n'est en rien comparable à la mémoire d'un micro-ordinateur. On ne parle pas en Giga-octets et il faut donc **apprendre à économiser cette mémoire** ; cela peut se faire en choisissant le bon type de variable ou en stockant les chaînes de caractères en mémoire programme (PROGMEM). De plus, la mémoire qui sert à stocker les variables sert aussi au microcontrôleur qui sauvegarde certaines données lors des appels de subroutines (interruptions par exemple). C'est ce qu'on appelle **la pile** et si nos précieuses données viennent l'écraser, c'est le dysfonctionnement assuré du programme et ce n'est pas toujours facile à comprendre.

Ce petit tour d'horizon nous a permis de comprendre que **la puissance d'un microcontrôleur est due à plusieurs facteurs** : sa taille en bits, sa fréquence d'horloge, la capacité de ses mémoires sont des éléments essentiels. Mais il faut aussi prendre en compte d'autres éléments comme les unités spécialisées qui sont à choisir en fonction de l'application qu'on veut obtenir (numérisation de signal, communication CAN ou I2C, nombre de broches pour communiquer avec l'extérieur, présence du WiFi, etc.). Les premières questions à se poser concernent donc le projet de modélisme ferroviaire qu'on a en tête et à partir de là, on se pose la question du µC qui peut le mieux convenir : c'est généralement celui qui dispose des interfaces utiles au projet (CAN, WiFi par exemple) car la vitesse n'est pas forcément un élément bloquant pour nos trains miniatures. Mais il faut aussi tenir compte des logiciels délivrés avec le µC, notamment les bibliothèques qui vous permettront de faire fonctionner des actionneurs particuliers à votre projet.

11.2 / APPRENDRE D'AUTRES LANGAGES DE PROGRAMMATION

Il existe d'autres langages de programmation que le langage Arduino qui est lui-même du langage C ou C++. On a évoqué **Processing** dans le chapitre 9 et comme l'environnement de Processing ressemble à l'environnement d'Arduino, vous ne devriez pas être trop perdu à consacrer un peu de temps à apprendre ce langage. Cela permet de faire des **interfaces graphiques à un programme**, comme par exemple un TCO pour un réseau, ou bien une interface de commande. Cette dernière peut aussi être faite avec le langage HTML (Hyper Text Markup Language) et vous trouverez sur le site de LOCODUINO quelques exemples à ce sujet (le HTML complétant le programme en C/C++).

En 2024, le langage **Python** était le plus utilisé dans le monde. Les cartes Arduino n'ont pas échappé à ce phénomène et peuvent maintenant être programmé en **MicroPython**, à condition d'être compatibles avec ce langage. Le Python est un langage interprété et non compilé, donc un peu plus lent : pour le train, cela n'a pas une grosse incidence. On peut trouver de nombreuses applications en MicroPython et s'en servir. Enfin, **le Python est le langage idéal pour commencer la programmation car il est plus simple dans sa syntaxe** ; il n'y a pas besoin de définir le type de variable par exemple, et il n'y a pas besoin de terminer une ligne par un point-virgule, le retour chariot suffisant. On trouve des boucles for ou while et des tests if-else (comme en C/C++) et il est aussi possible de définir des fonctions. Les fortes similitudes avec le C/C++ font que la connaissance d'un des deux langages permet d'acquérir facilement le deuxième sans être trop perdu. Je vous encourage donc à essayer, ne serait-ce que pour varier les plaisirs.

Enfin, si on recherche la rapidité d'exécution et la compacité du code, **l'assembleur est le langage idéal car c'est celui qui suit au plus près l'intimité du microcontrôleur**. Hélas, la rédaction des programmes et leur mise au point est un **véritable casse-tête**. Si vous vous lancez dans l'aventure, attendez-vous à de nombreuses heures à chercher les bugs. Comme je l'ai dit dans une série d'articles publiée sur LOCODUINO (**https://locoduino.org/spip.php?article280**), l'assembleur n'apporte pas grand-chose dans notre hobby. Dans certains cas cependant, l'assembleur peut avoir une utilité comme, par exemple, respecter des timings très précis ou gagner en vitesse d'exécution. Quant à la compacité du code, faire clignoter une LED avec le programme Blink demande 924 octets de programme, alors qu'en assembleur, **il n'en faut que … 30 !** Malgré ces avantages, l'assembleur a beaucoup de défauts : vous devez penser à tout car personne ne le fait à votre place. Il faut parfaitement connaître le microcontrôleur et son architecture. Il faut un logiciel pour générer le code (par exemple MicrochipStudio 7) et il est préférable d'avoir un programmeur (on en trouve chez Microchip, anciennement Atmel). Il faut aussi apprendre à utiliser tout cela. Bref, c'est beaucoup de temps consacré pour une cause qui ne se justifie pas dans notre domaine. À vous de voir …

11.3 / IMAGINER LE TRAIN MINIATURE DE DEMAIN

Le train miniature a beaucoup évolué ces vingt dernières années : on est passé de l'analogique au numérique. Pourquoi ? Et bien déjà pour résoudre certains problèmes propres à l'analogique. Par exemple, pour garer un engin moteur sur une voie, il faut créer une section de garage non alimentée. En conséquence, cela complique le câblage du réseau. En plus, on ne peut pas faire circuler sur une même voie deux locomotives indépendamment l'une de l'autre : impossible donc d'envoyer une locomotive en chercher une autre pour s'y atteler et créer une unité multiple. Enfin, ce qui est garé en analogique ne reçoit plus de courant donc pas d'éclairage des feux ou des voitures, sauf à ajouter un système électronique qui ajoute du courant de haute fréquence sur la voie. **Tout cela a été résolu par l'arrivée du numérique** : les locomotives sont commandées indépendamment les unes des autres, la voie est constamment alimentée ce qui permet de garder l'éclairage, et au final il n'y a que deux fils à relier aux rails pour que cela fonctionne. Oui, c'est plus cher mais cela permet plus de choses. Mais si on veut que des trains se suivent en toute sécurité, alors il faut recréer des cantons, donc un câblage plus compliqué. Le numérique n'a pas résolu le court-circuit des boucles de retournement mais un module peut s'en occuper. Enfin, la captation de courant se fait mieux car la voie, étant alimentée en alternatif, s'encrasse moins qu'en analogique, mais il reste tout de même des problèmes de captation, notamment sur les aiguilles pour des engins courts, ce qui oblige à polariser la pointe de cœur : il faut s'y connaître et encore une fois, cela complique le câblage. Pour repérer la position des trains sur le réseau, il faut des capteurs ou bien un logiciel coûteux qu'il faut calibrer.

Le numérique a donc résolu beaucoup de choses mais **des problèmes, il en reste encore !** Le train du futur sera imaginé justement pour résoudre les problèmes restants. La meilleure solution pour résoudre les problèmes de câblage du réseau, le court-circuit des boucles de retournement et les problèmes de captation dus à l'encrassement ou aux appareils de voie, c'est d'avoir **une batterie à bord du train** (concept « dead rail » aux USA). Plus besoin de soigner la pose de la voie, elle ne sert qu'à guider les roues. Plus de problème de câblage, la voie n'est pas alimentée. Plus de court-circuit, on peut dessiner le réseau qu'on veut. Bien sûr, on rajoute le problème de la recharge des batteries, mais celles-ci ont fait de gros progrès et en feront encore : les batteries de demain auront **de plus en plus de capacité et le temps de recharge sera abaissé**. C'est déjà le cas des **batteries à graphène**. Cette recharge pourra d'ailleurs se faire par induction, lorsque le train fait une halte par exemple ou lorsque le réseau ne sert pas.

Mais comment les décodeurs recevront-ils les ordres ? Et bien avec des techniques qui ressemblent à la **radio** (cela existe déjà) ou au **WiFi** (objet connecté) ou encore au **Bluetooth**. Une transmission **OTA (Over The Air)** permettra d'ailleurs un bien plus gros débit que celui du numérique d'aujourd'hui. Avec l'ordre reçu sans fil et l'énergie déjà à bord, les décodeurs fonctionneront de la même manière et permettront sans doute encore plus de choses qu'aujourd'hui.

Mais il faudra toujours des capteurs pour localiser les trains ? Pas forcément, si on utilise le **GPS RTK (Global Positioning System Real Time Kinematic)** qui permet une précision de l'ordre du cm. Avec cette précision, on sait exactement où se trouve le train sur le réseau. Aujourd'hui, ces techniques coûtent encore cher mais elles se démocratisent chaque jour un peu plus et on trouve déjà sur YouTube des tutos pour le GPS RTK. La communication des décodeurs sera très certainement **dans les deux sens**, ce qui fait que le train pourra envoyer à la centrale sa position et celle-ci pourra l'afficher sur un écran de TCO et

Un projet n'est pas toujours simple : ici le Viaduc de Garabit

aubiner la signalisation lumineuse. On peut donc aussi imaginer que **la centrale DCC du futur fera plus de choses que certains logiciels de contrôle de réseau actuels, pour un prix inférieur**.

Enfin, il est à parier, vu la miniaturisation qui existe déjà à l'heure actuelle, que toutes les locomotives seront **équipées d'une micro-caméra à l'intérieur des postes de conduite**, et que l'image de notre propre réseau sera renvoyée sur notre smartphone ou tablette. La conduite se fera alors comme si on avait pu rentrer dans notre modèle réduit, ce qui sera plus intéressant que regarder les trains passer comme on le fait actuellement. Les aéromodèles et les drones permettent déjà cette fonctionnalité, ce qui a donné un nouvel engouement pour le pilotage des modèles réduits.

Tout cela est-il concevable ? Il y a une vingtaine d'années, les aéromodélistes volaient en thermique avec des moteurs capricieux, polluants et bruyants. Et si quelqu'un évoquait le vol en électrique, il se faisait railler car les batteries étaient à l'époque trop lourdes. Aujourd'hui, quasiment tous les aéromodélistes volent en électrique car batteries et moteurs ont fait des progrès et pourtant la consommation de ces derniers est bien supérieure à la consommation d'un train à l'échelle H0. Concevable est donc le terme qui convient, mais qui connaît l'avenir ? Pour ma part, j'y crois fortement et comme l'avenir se construit dès aujourd'hui, alors il me reste du pain sur la planche. Un peu d'aide de votre part serait la bienvenue !

RÉSUMÉ DU CHAPÎTRE 11

Plus vous programmerez des microcontrôleurs et plus vous aurez envie de connaître leurs secrets. La meilleure façon de progresser, c'est de ne pas s'en tenir à ce qu'on connait mais de découvrir de nouveaux langages et de nouvelles techniques. Quant au train miniature du futur, on peut se contenter d'attendre sa venue ou bien précéder le mouvement et participer à son élaboration. Certains s'y sont déjà mis.