Arduino **Portenta Machine Control** et Arduino **Portenta H7**

démonstration avec une passerelle CAN vers MQTT

Figure 1. Arduino Portenta Machine Control.

Mathias Claussen (Elektor)

L'Arduino Portenta Machine Control peut-il rivaliser avec un automate programmable (PLC) ? Arduino met le pied dans la porte de l'industriel. Voici un bref aperçu de l'Arduino Portenta Machine Control et de l'Arduino Portenta H7.

Arduino en environnement industriel : c'est la voie que la série Arduino Pro aimerait emprunter. Avec l'Arduino Portenta Machine Control (**figure 1**), on découvre une plate-forme munie d'E/S compatibles avec les classiques 24 V ou 4-20-mA. On y trouve en plus les systèmes de bus Ethernet, CAN-FD, RS-232, RS-485 et RS-422. Le wifi, le Bluetooth et l'USB sont présents pour encore plus de connectivité. Bien entendu, l'I²C et les connexions pour thermocouples ne sont pas oubliés. Les branchements sont très faciles sur l'Arduino Portenta Machine Control grâce aux borniers, visibles sur la **figure 2**.

Ceux qui se souviennent maintenant des connexions Siemens S7 n'ont pas tort. L'Arduino Portenta Machine Control est destiné à offrir une alternative aux développeurs dans les usines existantes - alternatives qui ne seraient pas réalisables avec des PLC classiques. Il n'est pas non plus surprenant que l'Arduino Portenta Machine Control dispose d'un boîtier fonctionnel pour rail DIN.

Arduino Portenta Machine Control

En quoi l'Arduino Portenta Machine Control est-il différent d'un PLC classique ? En cherchant un peu, vous trouverez des informations sur le fonctionnement interne d'un Siemens S7-1200 [1]. Mais qu'est-ce qui rend l'Arduino Portenta Machine Control meilleur ? Il n'y a pas de Raspberry Pi ou de Beagle Bone Black à l'intérieur. Le cœur de la machine est l'Arduino Portenta H7 et son microcontrôleur (MCU) STM32H747XI. Le MCU est de type double cœur asymétrique avec un ARM Cortex-M7, qui peut être cadencé jusqu'à 480 MHz, et un ARM Cortex-M4, qui peut être cadencé à 240 MHz. En plus des 2 Mo de Flash et 1 Mo de SRAM internes du STM32, on dispose de 8 Mo de SD-RAM et 16 Mo de QSPI-Flash (*Quad Serial Peripheral Interface Flash*) [2].)

Le MCU STM32 Dual-core de l'Arduino Portenta H7 fournit ainsi suffisamment de ressources pour exécuter plus que de simples applications d'automate. Dans cet univers, le système est parfaitement apte à mettre en œuvre des réseaux neuronaux et permettre de nouvelles applications, grâce à l'apprentissage automatique. L'Arduino Portenta H7 est directement pris en charge par Edge Impulse pour une introduction facile dans l'apprentissage automatique. Mieux encore, l'Arduino Portenta H7 dispose d'un port USB-C qui peut non seulement fonctionner en tant qu'hôte ou esclave USB, mais qui, grâce à une conception intelligente, est capable de fournir une

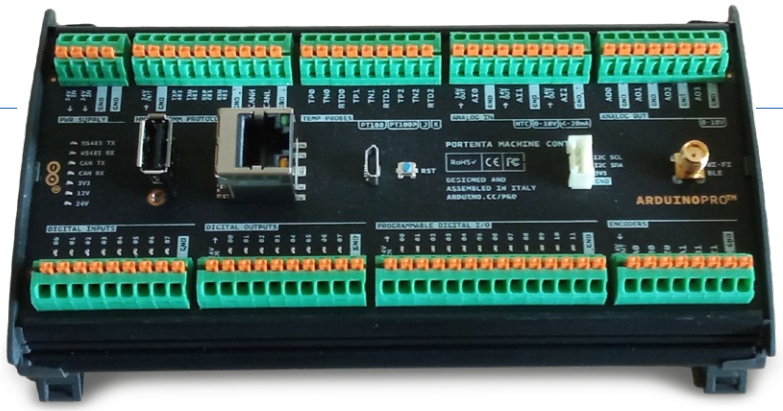

Figure 2. Les borniers de l'Arduino Portenta Machine Control.

sortie vidéo jusqu'à 720 p (1280×720 pixels). Il est donc possible d'installer directement un moniteur comme interface homme-machine (IHM) avec l'Arduino Portenta H7. Malheureusement, le port USB-C qui fournit le signal vidéo n'est pas directement accessible sur l'Arduino Portenta Machine Control.

Avec toutes ces nouvelles options, vous aurez probablement des idées pour vos propres applications. Dans cet article, je vais vous montrer comment aborder un petit projet avec l'Arduino Portenta H7 et l'Arduino Portenta Machine Control.

Atelier logiciel et EDI

On peut utiliser l'environnement Arduino pour l'Arduino Portenta H7 avec l'EDI Arduino en version 1.x ou le nouvel EDI en version 2.x. L'éditeur en version 1.x n'est pas le meilleur de sa catégorie, mais la version 2.x apporte des améliorations significatives. L'interface utilisateur de la version 2.x est présentée à la **figure 3**.

Pour communiquer avec l'Arduino Portenta H7 ou l'Arduino Portenta Machine Control avec l'EDI Arduino et commencer à développer, il suffit d'installer le package de cartes approprié (**figure 4**).

Vous pouvez également considérer PlatformIO [3]. L'Arduino Portenta H7 est ici aussi directement pris en charge, avec Visual Studio Code comme éditeur. L'EDI Arduino et l'interface de programmation d'application (API) cachent en arrière-plan d'autres manipulations logicielles. Il s'agit du système d'exploitation Mbed OS [4] et de sa bibliothèque d'abstraction matérielle (HAL) pour le matériel. On trouve en plus la bibliothèque *openAMP* [5], pour la communication entre les processeurs. Pour les pilotes spécifiques au matériel, STM32Cube de STMicroelectronics permet d'utiliser du code éprouvé provenant directement du fabricant. Les couches logicielles sont présentées à la **figure 5**.

Bibliothèques Arduino utiles pour l'Arduino Portenta Machine Control et l'Arduino Portenta H7

En plus du noyau de l'EDI Arduino et du package de cartes, il y a un certain nombre de bibliothèques qui permettent d'accélérer le développement. Pour l'Arduino Portenta Machine Control, c'est la bibliothèque nommée *Arduino Portenta Machine Control*. Cette bibliothèque fournit des fonctions appropriées pour l'interface du capteur de température de l'Arduino Portenta Machine Control et des commandes adaptées pour les modules de pilotage RS-485 et les GPIO 24 V. On y trouve aussi la commande de l'interface CAN. Pour se servir du wifi intégré pour installer un nouveau micrologiciel à distance, vous pouvez

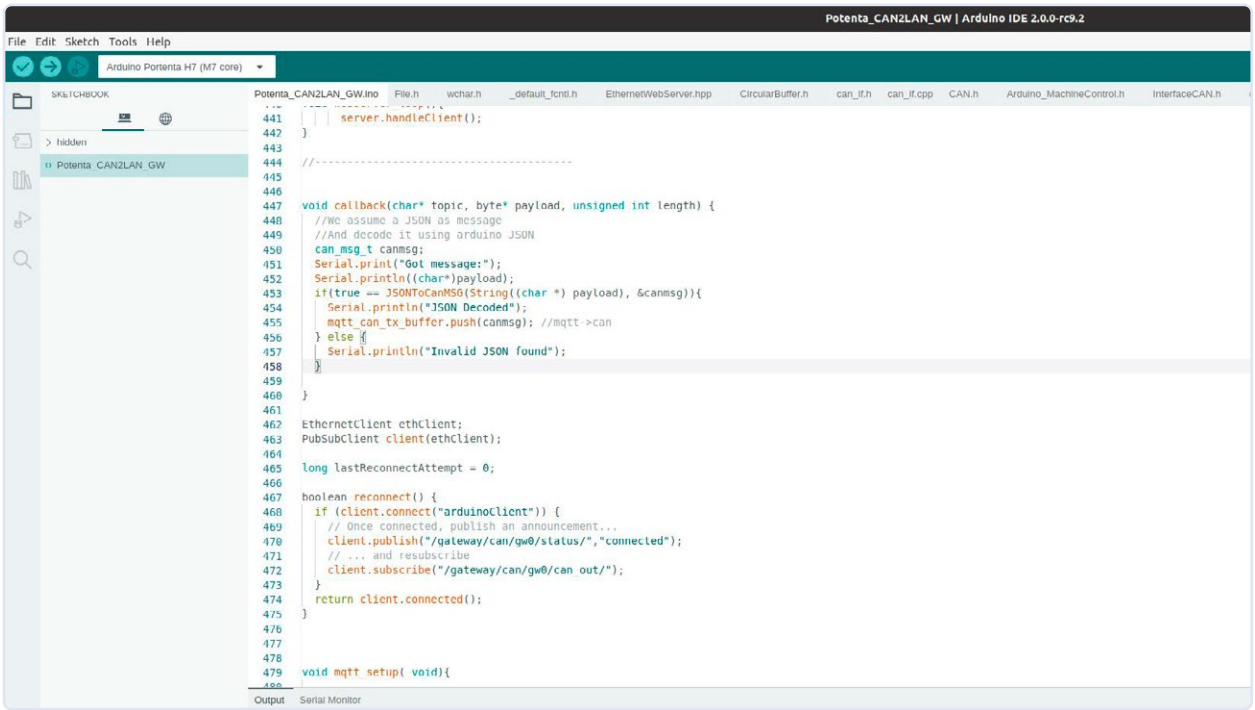

Figure 3. Arduino EDI version 2.

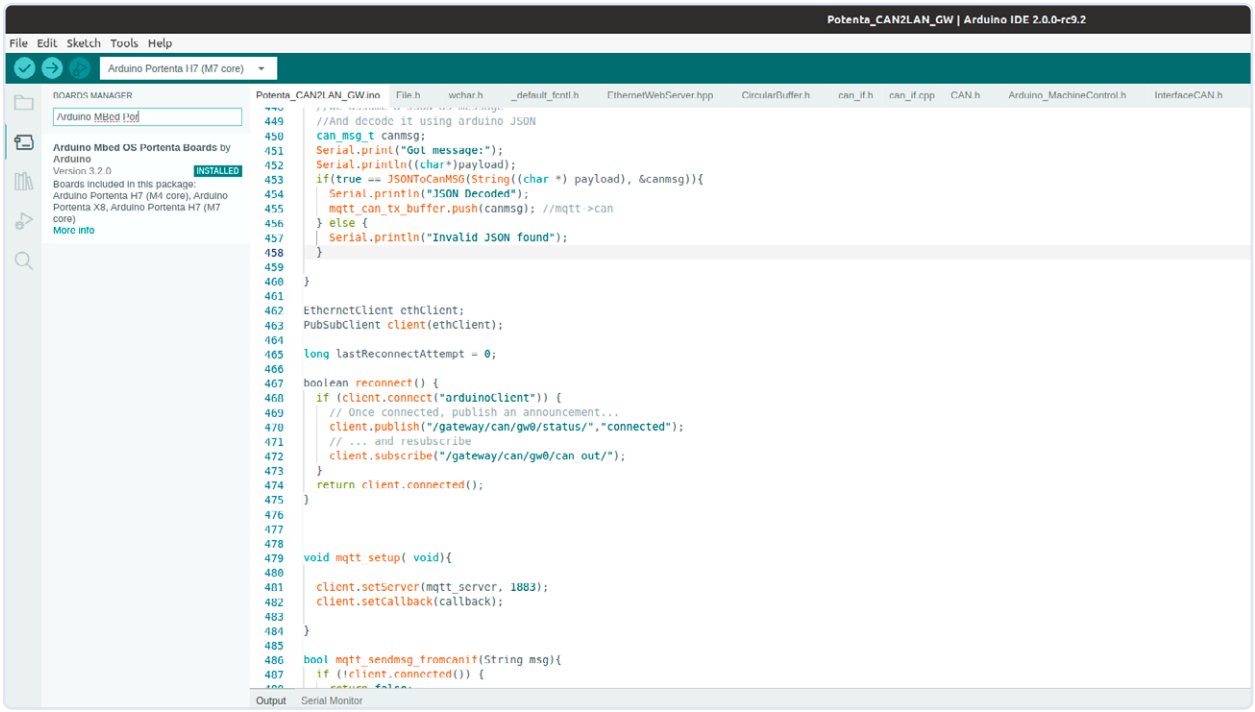

Figure 4. La carte dans le gestionnaire de cartes.

utiliser la bibliothèque *Arduino_Portenta_OTA*. Elle permet d'écrire un nouveau micrologiciel sur la mémoire flash externe ou une carte SD. Le micrologiciel sera mis à jour au prochain lancement du chargeur d'amorce.

Comme le *Bluetooth Low Energy* (BLE) est aussi pris en charge, il faut une pile BLE. Celle-ci s'appelle ArduinoBLE et peut être téléchargée via le gestionnaire de bibliothèques.

En ce qui concerne la sortie graphique, l'Arduino Portenta H7 s'appuie sur une bibliothèque qui organise comme il faut les pixels. C'est le cas de la bibliothèque *Little Versatile Graphics Library* (LVGL) qui, dans sa version actuelle, est une bonne base pour votre propre développement d'IHM.

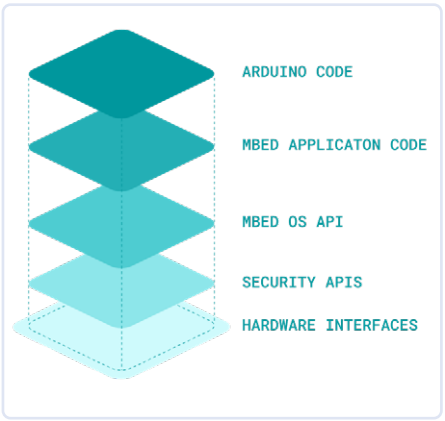

Figure 5. Couches logicielles pour l'Arduino Portenta H7. (Source : Arduino, https://bit.ly/arduino-mbed-stack)

Presqu'un automate classique : Arduino Portenta Machine Control et IEC 61131-3

Au moment de la rédaction de cet article, la prise en charge de la norme IEC 61131-3 [6] était encore en version bêta et les captures d'écran proviennent d'une version bêta (**figure 6** et **figure 7**). Mais qu'est-ce que la norme IEC 61131-3, et pourquoi est-ce une bonne chose que l'Arduino Portenta Machine Control la prenne en charge ?

Quiconque travaille avec des automates classiques tels que le S7-1200 de Siemens a déjà été confronté aux langages IL (liste d'instructions), LD (diagramme en échelle) ou ST (texte structuré). La création de contrôleurs avec ces langages de programmation est devenue un standard industriel. Ces langages et procédures sont décrits dans la norme IEC 61131-3. Prise en charge par l'Arduino Portenta Machine Control, les utilisateurs peuvent ainsi s'appuyer sur leurs acquis et mettre en œuvre des concepts éprouvés sans trop d'efforts.

Un MCU, deux cœurs de processeur

La programmation du MCU STM32 diffère de celle des autres Arduino. Ici, nous avons deux processeurs qui travaillent indépendamment, mais qui peuvent accéder aux mêmes ressources. Dans l'EDI Arduino, vous pouvez maintenant choisir si le projet doit fonctionner sur l'ARM Cortex-M4 ou -M7 et comment la mémoire flash disponible est divisée entre les processeurs. Par exemple, la supervision en temps réel de la machine peut s'exécuter sur l'un des cœurs, tandis qu'une instance de MicroPython fait son travail sur l'autre cœur. De même, vous pouvez faire calculer votre application d'intelligence artificielle sur le cœur Arm Cortex-M7 et utiliser le cœur M4 pour d'autres communications, telles que TCP/IP, CAN ou Modbus. Un schéma fonctionnel du MCU STM32 est présenté à la **figure 8**.

RAM externe, Flash et Secure Element

Comme mentionné au début, 8 Mo de SD-RAM sont connectés, et les deux UC peuvent y accéder. On peut accéder aux données, mais aussi exécuter du code à partir de cette RAM externe. De plus, le STM32 pourrait aussi monter directement dans sa zone de mémoire la Flash QSPI de 16 Mo. Cela permettrait à l'unité centrale de disposer de la Flash de 16 Mo comme mémoire interne et d'exécuter du code directement à partir de celle-ci (XiP - *Execution in Place*), mais cette option n'est pas prévue dans la

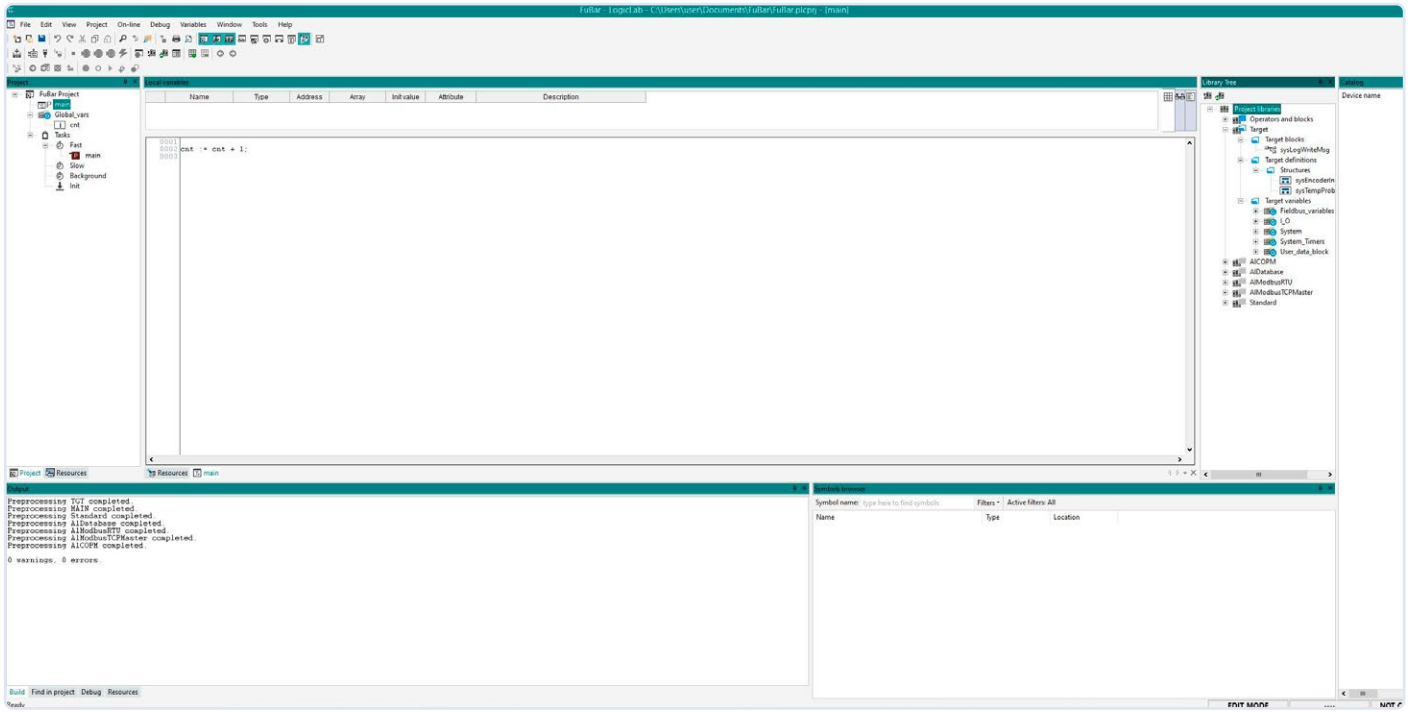

Figure 6. Compteur simple dans l'éditeur Arduino PLC.

pile logicielle Arduino. La flash de 16 Mo est traitée comme une flash QSPI externe et utilisée comme mémoire de masse avec un système de fichiers flash correspondant. Cela permet de stocker jusqu'à 16 Mo de données en mémoire, ce qui peut être très pratique pour les mises à jour *over-the-air* (OTA) ou d'autres données.

Lors de son développement, il faut tenir compte de l'endroit où le code est placé. L'exécution à partir de la mémoire flash interne ou de la mémoire SD-RAM obligera le MCU à attendre le code et les données pendant des périodes plus ou moins longues. Un placement maladroit peut entraîner une perte importante de puissance de traitement.

L'Arduino Portenta H7 possède un élément sécurisé. Il s'agit du *Secure Element NXP SE050*, dont une déclinaison réduite, le SE050E, a fait l'objet d'une critique sur le site Web d'Elektor. [7]

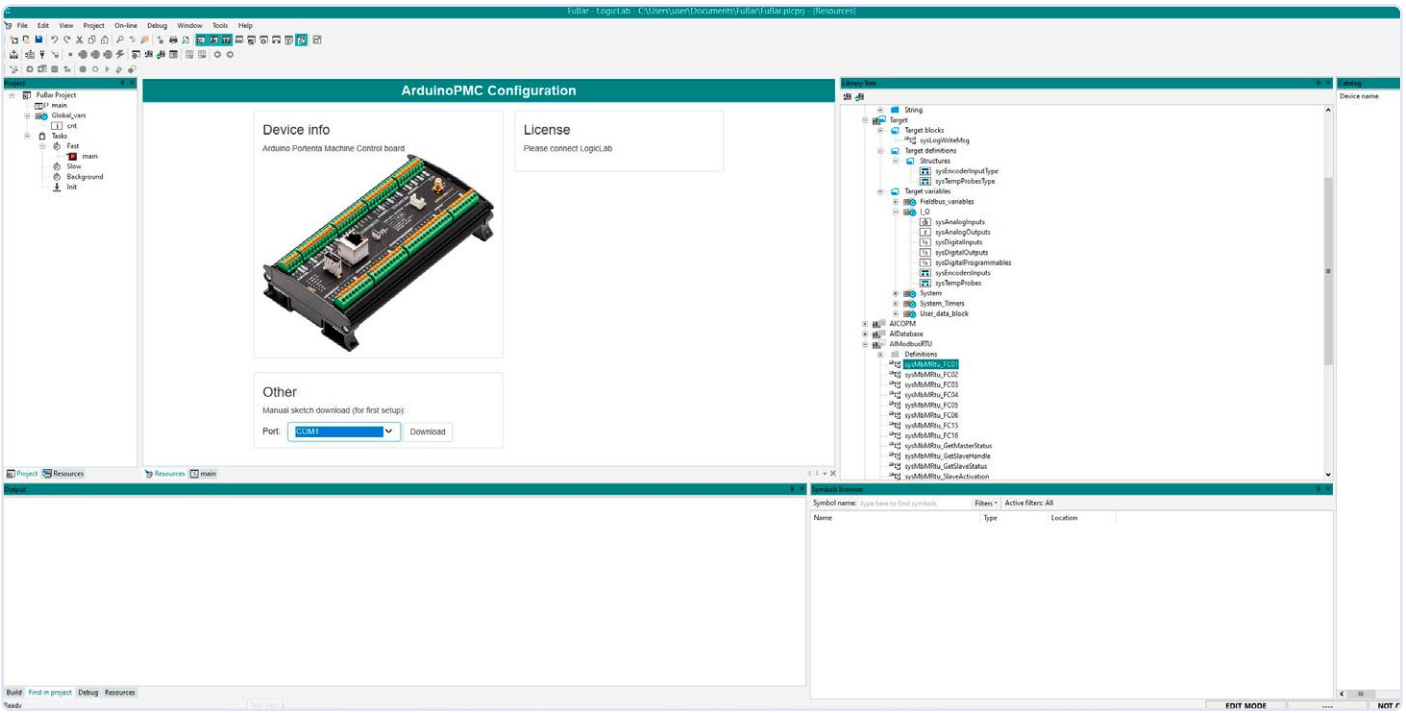

Figure 7. Vue d'ensemble du projet dans l'éditeur Arduino PLC.

Exemple de projet avec l'Arduino Portenta Machine Control

Maintenant que le matériel a été présenté, un petit exemple de projet vous montrera comment écrire du logiciel pour l'Arduino Portenta Machine Control. On doit réaliser ici une passerelle CAN vers MQTT. Cette passerelle traitera les messages CAN selon la norme CAN 2.0B et les transmettra sous forme de messages MQTT à un serveur connecté via un LAN. Ce logiciel n'est qu'une simple démonstration et le code source ne peut certainement pas être considéré comme apte à la production.

Pourquoi une passerelle CAN vers MQTT ?

Si vous voulez enregistrer et évaluer des données sur un bus CAN, vous pouvez le faire confortablement à distance. Les données peuvent ensuite être écrites dans une base de données ou examinées pour détecter des anomalies à l'aide d'autres méthodes (avec l'IA par exemple). Grâce au transport via MQTT, plusieurs participants peuvent évaluer les données CAN ou envoyer des messages au bus CAN. Et le CAN n'est pas seulement utilisé dans l'industrie ou les voitures ; on peut aussi le trouver dans les modèles réduits de chemin de fer. [8]

Bibliothèques CAN, MQTT et WebSocket

Pour envoyer ou recevoir des messages CAN, on utilise le contrôleur CAN intégré, qui gère également le CAN-FD (*Controller Area Network Flexible Data-Rate*) du côté matériel. Malheureusement, le contrôleur CAN est quelque peu ralenti ici par Mbed OS, qui agit comme une couche intermédiaire, et est actuellement limité à CAN 2.0B.

On utilise PubSubClient [9] version 2.8 pour la connexion MQTT et EthernetWebServer de Khoi Hoang pour la connexion WebSocket [10]. Celui-ci fournit aussi un serveur web qui peut fournir des pages web. En termes de fonctionnalité, il est très similaire à celui de l'ESP32, de sorte que les applications peuvent être facilement migrées à partir d'un ESP32 ou ESP8266, du moins en théorie. Le serveur web lui-même est inclus dans le projet et renvoie aussi une page statique avec l'erreur 404 (page non trouvée). En ce qui concerne la lecture du système de fichiers flash de l'Arduino Portenta, il existe quelques incompatibilités mineures entre l'API de la bibliothèque *EthernetWebServer* et l'API d'accès au système de fichiers

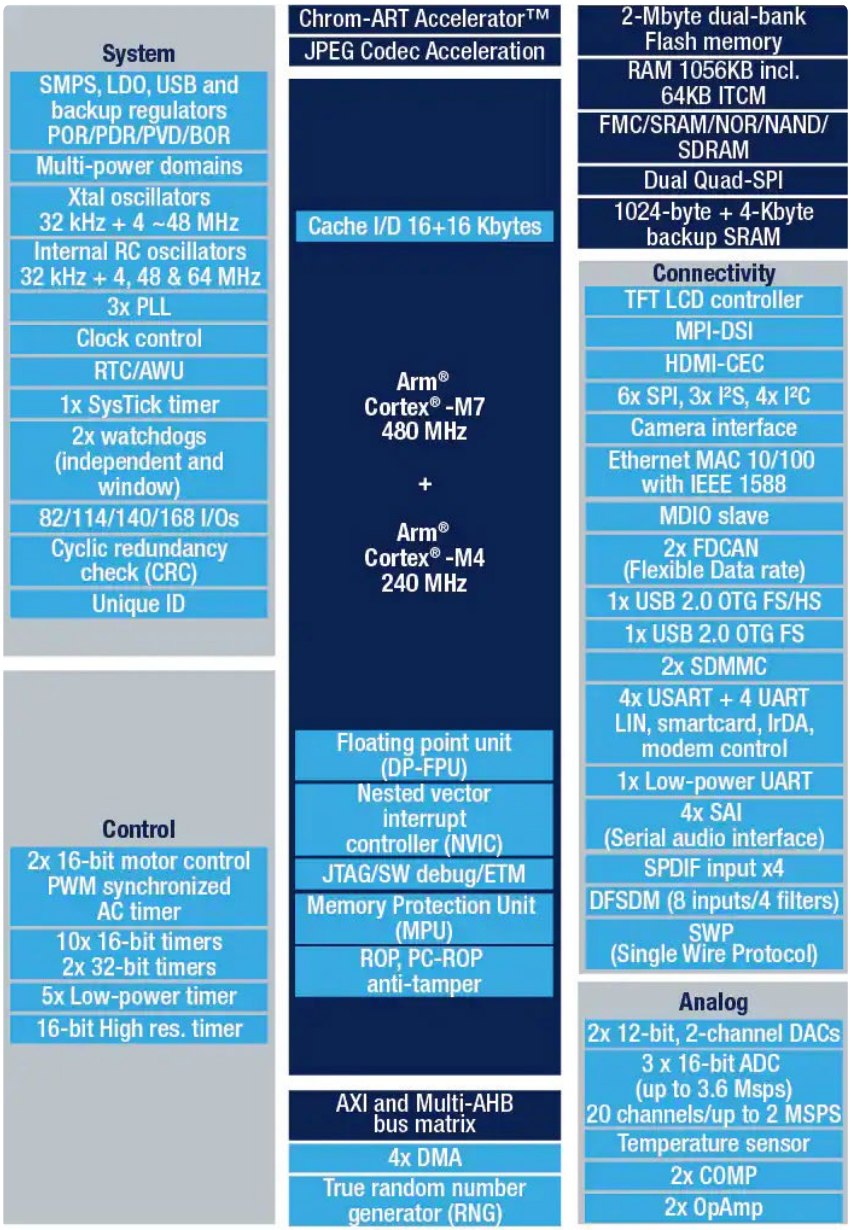

Figure 8. Vue d'ensemble du STM32H747XI (Source : STMicro, https://bit.ly/3xoRTDF)

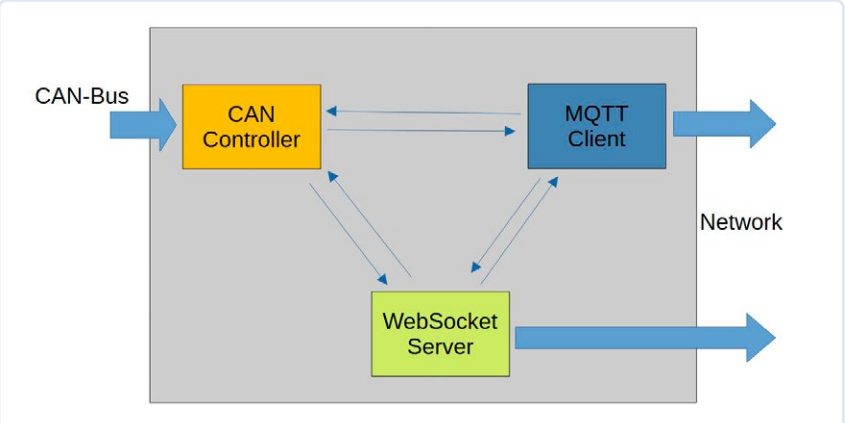

Figure 9. Schéma de principe du logiciel.

Figure 10. Configuration du CAN et du serveur MQTT.

de l'Arduino Portenta. Avec un peu plus de temps et d'effort, cependant, on peut le corriger en ajoutant une petite classe qui fournit alors au serveur web un accès adapté aux fichiers. Le principe du logiciel peut être vu dans la **figure 9**.

Le logiciel avait déjà été écrit avec l'idée d'un serveur web fonctionnel et fournit les messages CAN non seulement via MQTT, mais aussi via WebSocket. Cela permettrait aussi l'affichage et l'interaction avec le bus CAN via un navigateur Web.

Structure du logiciel

Pour la passerelle, les trois composants sont un client MQTT, un serveur WebSocket et le traitement des données CAN. Étant donné que ni WebSocket ni MQTT ne prescrivent la manière dont les données sont échangées, on utilise ici JSON.

La configuration du logiciel se fait ici entièrement dans le code source, à cause du court délai de développement du projet, ainsi que de complications mineures avec le serveur web. Le débit binaire approprié doit être réglé pour CAN et l'adresse IPv4 du réseau doit être saisie pour le serveur MQTT. On peut voir la partie du logiciel dédiée à la configuration à la **figure 10**.

Les messages CAN reçus par le contrôleur CAN sont transmis par le logiciel via MQTT et WebSocket. Si les données JSON contenant des messages CAN sont décodées avec succès via MQTT, elles sont transmises au contrôleur CAN et au serveur WebSocket. Pour les données WebSocket contenant des messages CAN, celles-ci sont transmises au serveur MQTT et au contrôleur CAN.

Le serveur WebSocket devait permettre d'afficher les données CAN via une interface Web ou d'envoyer des données. Cela n'aurait nécessité qu'un terminal avec un navigateur. Néanmoins, le serveur a été laissé dans le logiciel parce que d'autres systèmes peuvent se connecter directement à l'Arduino Portenta Machine Control pour échanger des données via WebSocket. Aucun serveur MQTT ne serait alors nécessaire ici.

JSON dans les messages MQTT et les connexions WebSocket

On utilise la notation d'objet JavaScript (JSON) [11] pour l'échange de données. La structure du JSON est assez simple et contient les données du message CAN ainsi que l'interface (CAN, MQTT ou WebSocket) par laquelle le message a été reçu. Un exemple d'une telle chaîne JSON se présente comme suit :

```
{
    "src": "can_if0",
    "type": "can",
    "extid": false,
    "canid": 2047,
    "length": 8,
    "data": [
        0,
        0,
        255,
        255,
        0,
        0,
        0,
        128
    ]
}
```

Pour la connexion MQTT, on utilise le *topic* « /gateway/can/gw0/can_in/ » pour les messages CAN reçus par le contrôleur CAN. Le message CAN est publié ici en tant que message JSON afin que tous ceux qui ont souscrit à ce topic MQTT reçoivent une copie du message. Les messages reçus via WebSocket sont publiés dans le *topic* « /gateway/can/gw0/ws_in/ » du serveur MQTT. Cela permet de distinguer les messages envoyés via MQTT de ceux envoyés via WebSocket à l'interface CAN. Le *topic* « /gateway/can/gw0/can_out/ » est souscrit sur le serveur MQTT. Si un message y est maintenant publié, le logiciel tente de le décoder en tant que message JSON et d'en générer un message CAN approprié. Si cela réussit, le message est transmis à l'interface CAN et au serveur WebSocket.

Et la suite ?

Le logiciel présenté ici n'est qu'un avant-goût et est loin d'être complet. Il est destiné à donner des indications sur la manière dont vos propres projets pourraient être conçus et à montrer comment vous pouvez rapidement écrire vos propres applications avec les bibliothèques existantes et quelques connaissances sur l'environnement Arduino. Le logiciel peut être téléchargé sur le site d'Elektor Labs. [12] L'Arduino Portenta Machine Control coûte 300 €. Lorsque vous établissez le budget de votre projet, gardez à l'esprit que, bien que le boîtier soit bien conçu, il est dépourvu d'antenne wifi, donc prévoyez en conséquence. Pensez aussi à prendre une antenne avec un gain d'au plus 3 dB pour respecter la réglementation.

Comme souvent, le logiciel peut être une épreuve. Mais commencer avec l'environnement Arduino est très simple et vous pouvez réutiliser beaucoup de vos connaissances. Avec l'Arduino EDI 2.x, l'éditeur a été beaucoup amélioré et obtenir tous les composants logiciels pour l'Arduino Portenta Machine Control est juste une question de minutes. À un moment donné, les bibliothèques ont besoin de l'apport des utilisateurs, donc si vous trouvez des bogues, veuillez les signaler. Cela permettra de garantir finalement une meilleure expérience pour tous.

Quant à l'Arduino Portenta Machine Control, l'impression est la suivante : beaucoup de potentiel. Son Arduino Portenta H7 est une base très solide en termes de matériel, et le prix d'environ 120 € le rend intéressant pour de nombreux projets. L'option permettant de réaliser une IHM est très attrayante. De plus, le fait d'avoir de la RAM et de la Flash connectées au STM32H7 sur l'Arduino Portenta H7 lui permet de contenir des données plus complexes et même d'exécuter des modèles d'intelligence artificielle plus importants sur le MCU. Avec le wifi, le Bluetooth et l'Ethernet, ainsi que le matériel pour les systèmes de bus courants, très peu d'attentes restent insatisfaites. L'Arduino Portenta H7 peut non seulement servir de base aux applications de contrôle industriel, mais en termes de matériel serait certainement aussi bien adapté comme cœur d'autres applications.

(220448-04) — VF : Denis Lafourcade

À propos de l'auteur

Mathias Claussen est ingénieur senior et rédacteur technique chez Elektor. N'hésitez pas à le contacter à l'adresse mathias.claussen@elektor.com. Vous pouvez lire plusieurs de ses articles sur www.elektormagazine.com/claussen. Vous pouvez également regarder Mathias lors du livestream mensuel Elektor Lab Talk sur YouTube (www.elektormagazine.com/elt), où vous pouvez lui poser vos questions en direct !

Produits

- **Arduino Portenta H7**
  www.elektormagazine.fr/arduino-portenta-h7
- **Arduino Portenta Machine Control**
  www.elektormagazine.fr/arduino-portenta-machine-control
- **Arduino Portenta Vision Shield**
  www.elektormagazine.fr/arduino-portenta-vision-shield

LIENS

[1] L'intérieur d'un Siemens S7-1200 : https://sec-consult.com/blog/detail/reverse-engineering-architecture-pinout-plc/
[2] Wikipedia : QSPI : https://en.wikipedia.org/wiki/Serial_Peripheral_Interface#Quad_SPI
[3] PlatformIO : https://platformio.org/
[4] ARM Mbed OS : https://os.mbed.com/mbed-os/
[5] Dépôt GitHub d'OpenAMP : https://github.com/OpenAMP/open-amp
[6] Wikipedia : IEC 61131-3 : https://en.wikipedia.org/wiki/IEC_61131-3
[7] M. Claussen, « Evaluation du Secure Element NXP EdgeLock® SE050E », Elektor, juin 2022 : www.elektormagazine.fr/news/evaluation-du-secure-element-nxp-edgelock-se050e
[8] Märklin CS2 Protocole CAN : www.maerklin.de/fileadmin/media/produkte/CS2_can-protokoll_1-0.pdf
[9] pubsubclient Dépôt GitHub : https://github.com/knolleary/pubsubclient
[10] Dépôt GitHub d'EthernetWebServer : https://github.com/khoih-prog/EthernetWebServer
[11] JSON : www.json.org/json-en.html
[12] Projet sur Elektor Labs : www.elektormagazine.fr/labs/can-to-mqtt-gateway-with-arduino