Conception et réalisation d'un émulateur de *smart grid*

Alexandre FAYE-BEDRIN – encadré par Roman Le Goff Latimier
Ecole Normale Supérieure de Rennes

INTRODUCTION

La gestion de réseaux électriques est une des composantes fondamentales du génie électrique. Elle permet d'assurer la stabilité d'un réseau sur lequel sont connectés des producteurs et des consommateurs. La multiplication d'acteurs non conventionnels sur ces réseaux -– énergies renouvelables, charges flexibles, véhicules électriques, et caetera -– ainsi que l'essor des technologies de communication ont donné lieu au concept de *smart grid* reposant sur l'interaction entre un réseau de puissance et un de communication.

Les travaux de recherche au sein de la thématique se heurtent à des difficultés : il est en effet inenvisageable d'utiliser les infrastructures réelles, servant au transport d'énergie, pour réaliser des expérimentations dont l'issue, en partie imprévisible, pourrait avoir des conséquences désastreuses en cas d'incident. De même, l'utilisation de démonstrateur à échelle réduite (fig. 1) avec des composants réels – batterie, éolienne, panneau photovoltaïque, *etc* – induit des coûts élevés en plus d'un risque de détérioration desdits composants.

L'utilisation de convertisseurs d'électronique de puissance, pour émuler le comportement de ces agents à une échelle très réduite, semble donc une piste intéressante pour implanter des stratégies sur un système réel tout en contrôlant les risques et en utilisant un système facilement déployable en laboratoire.

Des travaux antérieurs du laboratoire SATIE ont permis de mettre au point une structure d'électronique de puissance – pont en H – permettant d'émuler (à échelle réduite) tout consommateur ou producteur sur un réseau. L'objectif de ce projet de recherche est d'enrichir ce composant pour en faire un élément facilement pilotable qui puisse être fabriqué en de multiples exemplaires afin de les raccorder en réseau (fig. 2).

Il s'agit donc de réaliser la structure électronique, de la piloter à l'aide d'une structure de contrôle rapproché, et de commander le tout depuis un outil informatique.

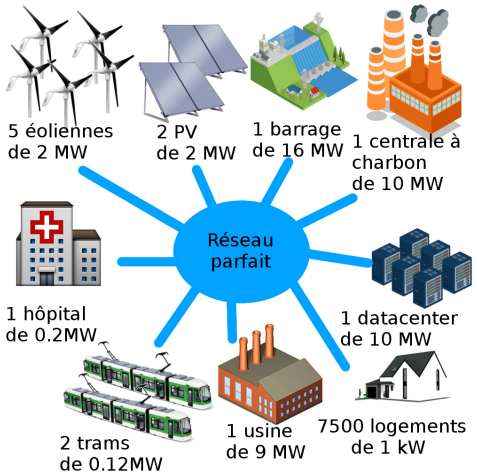

FIGURE 1: Exemple de réseau

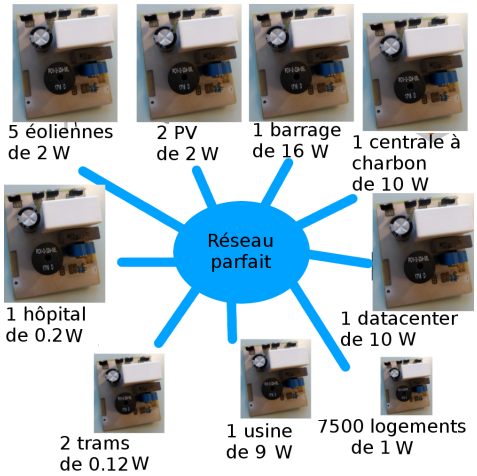

FIGURE 2: Réseau émulé

I. CONTEXTE ET OBJECTIFS

Pour devenir un agent complet sur un réseau de démonstration, le pont en H, composé d'une carte de puissance et d'une carte d'adaptation de signal, doit être accompagné d'une structure de contrôle rapproché (un processeur de signal par exemple), elle-même commandée par un dispositif où l'on peut implémenter une loi de gestion et que l'on peut placer sur un réseau d'information.

Ce projet de recherche a pour objectifs principaux (voir aussi fig. 3), afin de réaliser l'agent pilotable :

Réseau informatique	Lois de gestion	Raspberry Pi	Bus
	Asservissement, commande rapprochée	DSP	Signaux analogiques
	Adaptation du signal de commande	Carte signal	Commande
Réseau électrique	Réalisation de la puissance	Carte puissance (pont en H)	

FIGURE 3: Les différents étages du composant à enrichir, avec en rouge les objectifs de ce projet de recherche

— Réalisation et test de la structure électronique à partir d'une conception pré-existante.
— Conception des lois de contrôle rapproché (asservissement numérique des tensions et courants) par le DSP – processeur temps réel de traitement de signal – TI F28379D.
— Communication de ce DSP avec une Raspberry Pi – ordinateur léger basé sur une architecture ARM – réalisant le contrôle de haut niveau (calcul du point de fonctionnement – puissance active et réactive – en fonction de lois de gestion)

Le dispositif conçu doit être réalisé dans une démarche d'open hardware dont les spécifications, composants et codes seront rendus publics.

II. Réalisation et test de la structure électronique

La structure électronique est constituée de deux cartes indépendantes, l'une regroupant les composants de puissance et l'autre le nécessaire à la commande de ces composants.

A. Réalisation

Les deux cartes électroniques ont pour base chacune un circuit imprimé (PCB), réalisé dans les locaux de l'ENS. Un fichier de définition créé à l'aide du logiciel *Autodesk EAGLE* a servi à la réalisation de ces PCB, et contient les spécifications des composants à placer sur les cartes (**????** and fig. 4).

Les composants (ou leurs sockets) sont donc placés, et les contacts électriques réalisés par brasure avec un alliage d'étain, ce sur les deux cartes (fig. 5 and **??**).

B. Tests et retours

Les contacts électriques réalisés par brasure ont été testés par continuité à l'aide d'un multimètre, mais l'intégrité et le fonctionnement des cartes n'ont pas été vérifiés, ceci ayant fait l'objet de travaux antérieurs qui ont permis d'arriver à la version actuelle de la carte.

Cependant, la réalisation des cartes a déjà permis de repérer un léger défaut de conception de la version courante : un connecteur de la carte de puissance était placé sur la conception de telle manière qu'il était inaccessible (fig. 4). Cela a permis de corriger l'erreur pour une future version de la carte.

III. Prise en main du DSP et de ses outils de programmation

Le DSP utilisé (TI F28379D) est un matériel sur lequel l'équipe de recherche n'a pas encore de savoir faire. Il est donc nécessaire de tester ses possibilités et de mettre en place sa chaîne logicielle. Ce processeur cadencé à $200MHz$ possède deux coeurs capables d'opérations complexes (opérations sur des nombres flottants simple précision, fonctions trigonométriques par exemple). La lecture de la documentation du matériel ainsi que du module *MATLAB* adéquat apportent les éclaircissement nécessaires à la prise en main de l'ensemble.

A. Prise en charge par MATLAB

Le DSP est supporté par le logiciel *Mathworks MATLAB* et son module *Simulink*, à la condition d'installer le *hardware support package* consacré (pour le microcontrôleur C2000) ainsi que la suite logicielle fournie par le fabricant TI, *Code Composer Studio 8* (IDE et compilateur) accompagné de *controlSUITE* (gestion des DSP connectés à l'ordinateur).

MATLAB permet une programmation visuelle sous forme de schéma bloc, ce qui est un avantage par rapport à la programmation en C, car elle ne nécessite pas d'apprendre les spécificités du langage et de ses bibliothèques propres au microcontrôleur. Elle permet également à des personnes tierces de relire facilement les programmes.

L'utilisation de ce DSP nécessite d'être en mesure de réaliser trois tâches élémentaires qui seront ici présentées : la création d'un signal MLI en sortie du DSP, l'acquisition de signaux analogiques en entrée du DSP, la communication avec la Raspberry Pi réalisant le contrôle de haut niveau.

B. Utilisation du module MLI du DSP

Pour utiliser la MLI, on configure un bloc *Simulink* nommé ePWM (fig. 6). Il permet de contrôler 2 sorties MLI, soit indépendamment (tant au niveau de la fréquence que du rapport cyclique), soit en mode complémentaire (la sortie dite B est au niveau haut lorsque la sortie A est au niveau bas, et inversement).

Le bloc permet de définir, entre autres et pour chacune des sorties :

— la période de la MLI, en cycle processeurs ou en secondes
— un premier seuil de déclenchement (en étapes de compteur ou en pourcentage de cycle) et son comportement
— un deuxième seuil de déclenchement et son comportement
— le comportement du compteur (dent de scie montante, descendante, ou triangle)

Dans une configuration simple, on utilise le compteur en dent de scie montante. Le premier seuil de déclenchement définit la sortie à un niveau bas lorsqu'il est atteint sur un front montant, et à un niveau haut lorsqu'il est atteint sur un front descendant.

On peut noter que la période définie en secondes n'est pas respectée expérimentalement : la fréquence mesurée sur oscilloscope est égale au double de la fréquence demandée dans la configuration.

C. Détermination du gain d'un CAN

Les convertisseurs analogique vers numérique (CAN) du DSP ont vocation à être utilisés, notamment lors de l'asservissement du courant ou de la tension, pour récupérer les valeurs issues de capteurs et sondes.

Les CAN du DSP sont décrits dans la documentation comme ayant une précision sur 12bits, mais sans indications particulières sur l'échelle adoptée. Un programme *MATLAB* (fig. 7) est donc utilisé pour déterminer cette inconnue.

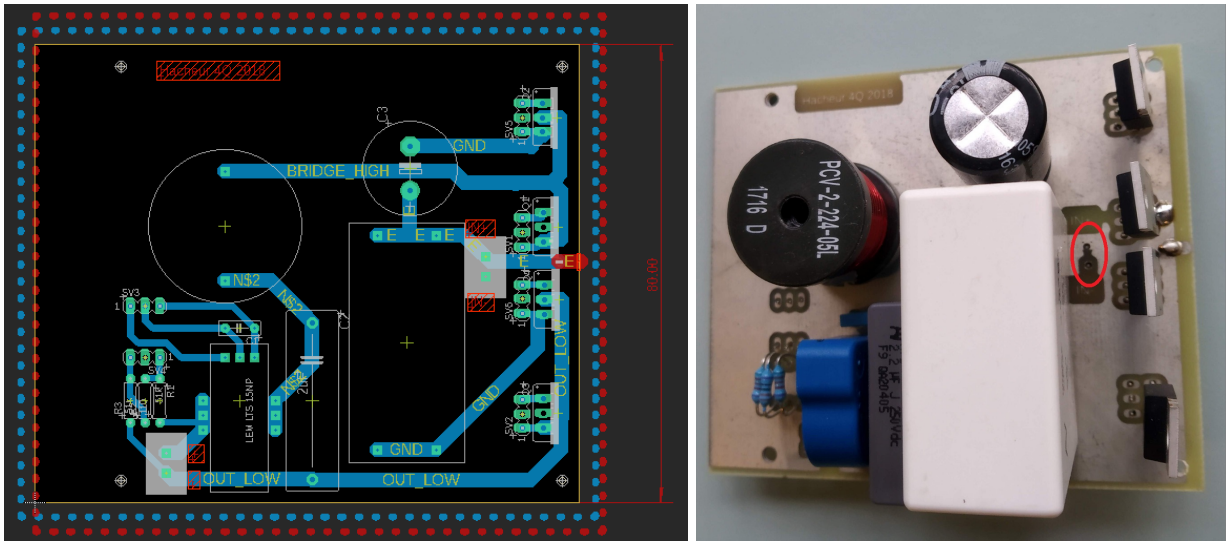

FIGURE 4: *Board EAGLE* de la carte de puissance ; et emplacement prévu du connecteur sur la carte puissance (entouré en rouge). On observe qu'il aurait été coincé entre d'autres composants, et donc inaccessible. Il a en conséquence été placé au verso de cette carte.

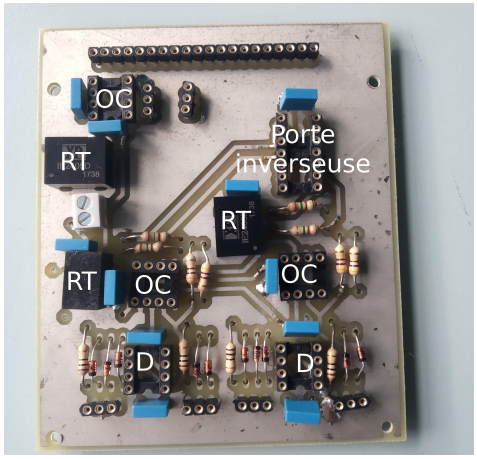

FIGURE 5: Recto de la carte signal. Légende : OC = Optocoupleur, RT = régulateur de tension, D = Driver de MOSFET

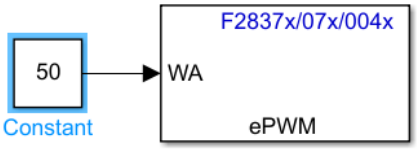

FIGURE 6: Programme *MATLAB* d'essai du module MLI

Les essais, qui consistent en l'envoi d'une tension continue à l'entrée du CAN et d'une observation de la MLI à l'oscilloscope, permettent de déterminer que le CAN accepte des tensions comprises entre 0 et $3V$, et que le gain est d'environ $12365V^{-1}$, lorsqu'on définit la sortie du bloc comme de type *single*.

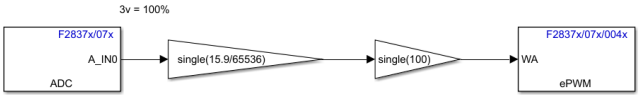

FIGURE 7: Programme *MATLAB* de détermination du gain du CAN. On peut y voir le gain déterminé expérimentalement.

IV. COMMUNICATION DU DSP ET DE LA RASPBERRY PI

Le système de décision (le réseau d'information) doit être lié avec le réseau électrique proprement dit, ainsi que ses agents. Alors que le DSP doit pouvoir asservir un hacheur pour contrôler tension et courant, il faut pouvoir lui indiquer une consigne : c'est le rôle de la Raspberry Pi. Cette dernière participe à la prise de décision concernant ladite consigne, puisqu'y est implémentée une loi de gestion et le nécessaire pour se mettre en accord avec les autres agents sur le réseau. La Raspberry Pi occupe de ce fait la couche décision et réseau d'information.

A. *Choix d'un protocole*

Trois protocoles série sont implémentés nativement sur les deux appareils :

- RS-232, protocole série datant des années 1960,
- I^2C (Inter-Integrated Circuit), conçu pour la domotique et l'électronique domestique,
- SPI (Serial Peripheral Interface), développé dans les années 1980 pour les systèmes embarqués.

Le protocole I^2C bénéficiant d'une simplicité de mise en place, et étant celui qui nécessite le moins de câbles (un pour l'horloge, un pour les données, ainsi qu'une masse commune), a été retenu pour cette implémentation. On peut cependant noter que les deux autres protocoles auraient pu satisfaire le besoin, et qu'aucun ne possédait d'avantage net sur les autres pour cette application.

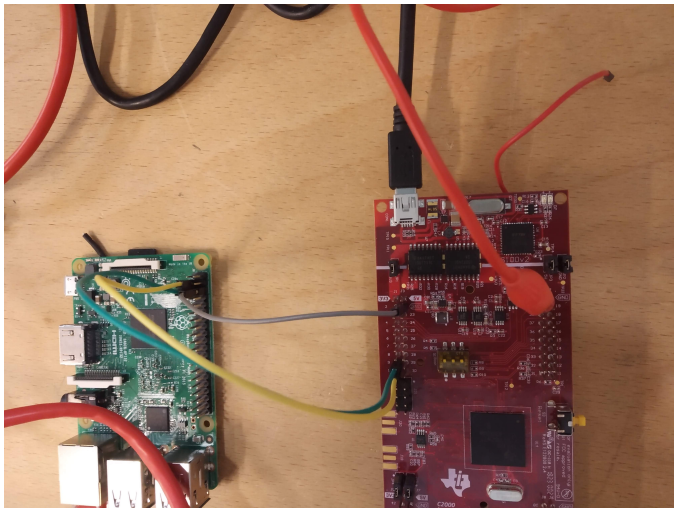

FIGURE 8: Raspberry Pi et DSP raccordés entre eux par un bus I^{2}C, ainsi qu'à un oscilloscope et un ordinateur (pour le DSP) et à un réseau informatique (pour la Raspberry Pi)

Dans le cas où l'on aurait voulu commander plusieurs DSP avec une seule Raspberry (ou un DSP et d'autres dispositifs), l'I^{2}C s'en serait en revanche sorti fort avantagé : RS-232 est une liaison point-à-point, c'est-à-dire qu'elle n'est pas conçue pour fonctionner avec plusieurs dispositifs communiquant sur le même canal, alors que SPI aurait besoin d'un fil sélecteur supplémentaire par agent esclave.

Ce protocole fonctionne sur un modèle maître-esclave, c'est-à-dire qu'un agent (nommé maître) se charge de commander et interroger les autres agents (les esclaves). Dans notre cas, la Raspberry Pi étant chargée de donner des ordres au DSP, on se propose de nommer maître la première et esclave le deuxième.

B. Câblage

Le câblage est fait suivant les indications des notices des deux appareils : on utilise un fil de masse, un fil d'horloge et un fil de données (fig. 8).

Le DSP et la Raspberry Pi sont reliés à un ordinateur servant pour la programmation, respectivement par USB et par Ethernet. Un oscilloscope est utilisé pour observer la sortie de la MLI du DSP, ainsi que pour surveiller divers signaux et tensions.

C. Programmation du DSP

Le DSP est programmé comme précédemment à l'aide du module *Simulink* de *MATLAB*. Il est, à cette occasion, configuré pour se comporter comme un esclave sur le bus. Le programme (fig. 9) est constitué d'un bloc de réception I^{2}C et du bloc de sortie MLI, ainsi que d'un bloc d'envoi I^{2}C accolé au bloc de lecture du CAN.

Le comportement attendu, et vérifié par la suite, est que la Raspberry puisse envoyer deux valeurs numériques sur un octet chacune, qui sont utilisées pour déterminer chacune un rapport cyclique du module MLI ; et lorsque la Rapsberry demande au DSP à lire une valeur, celui-ci réponde la mesure de tension effectuée par le CAN.

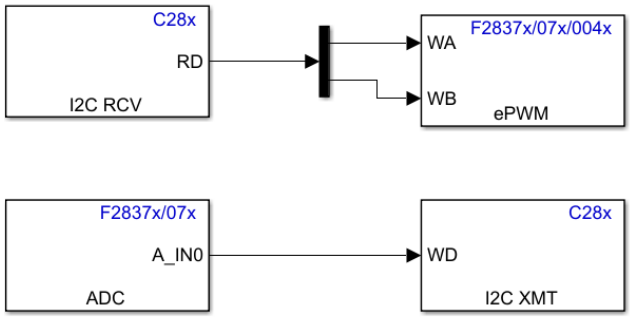

FIGURE 9: Programme esclave du DSP

D. Programmation de la Raspberry Pi

La Raspberry Pi dispose de bibliothèques Python pour le contrôle de son implémentation de l'I^{2}C, présentes dans le module *SMBUS*.[1] Elle est, par défaut, configurée pour se comporter en maître sur le bus.

Le programme dit maître, qui se charge d'envoyer les deux valeurs attendues par le programme esclave, puis de lui demander la valeur donnée par le CAN, est donc écrit en Python 3, et comporte moins de 10 lignes de code utile (voir annexe).

E. Problèmes rencontrés

La mise en place du bus a rencontré quelques obstacles.

1) Outils I^{2}C en ligne de commande: Des outils de manipulation en ligne de commande du bus I^{2}C sont fournis avec le système d'exploitation de la Raspberry Pi. L'un d'eux, *i2cdetect*, sonde toutes les adresses du bus pour tenter de détecter les agents présents. Une fois l'utilitaire exécuté, le DSP bloque le bus en maintenant le signal horloge à son niveau dominant. Les tentatives ultérieures d'utiliser le bus depuis la Raspberry se soldent donc systématiquement par un échec.

Pour débloquer le bus, il faut et suffit de redémarrer le DSP, en coupant et rétablissant sa source d'alimentation.

2) Paramètres du bus non maîtrisés: Apparemment aléatoirement, et environ une fois sur quatre, les deux valeurs envoyées par la Raspberry au DSP sont échangées.

Pour trouver la provenance exacte du problème, il faudrait observer le signal transmis par le bus avec un dispositif tiers, puis examiner le code C généré par *MATLAB* à partir du schéma bloc.

A la lumière des essais qui ont été menés, la communication entre le DSP et la Raspberry via un bus I^{2}C n'est donc pas en l'état une solution satisfaisante pour une utilisation fiable. Des comportements non répétables et non documentés par les constructeurs apparaissent trop régulièrement pour répondre aux besoins de l'application pour un démonstrateur de smart grid. Des travaux futurs seront donc nécessaire pour élucider ces comportements ou bien mettre en œuvre les autres protocoles disponibles.

1. SMBus est un protocole reprenant les caractéristiques de l'I^{2}C et précisant les délais entre deux fronts sur les signaux d'horloge et de données, ainsi que les niveaux de tension à utiliser. Il est compatible avec l'I^{2}C.

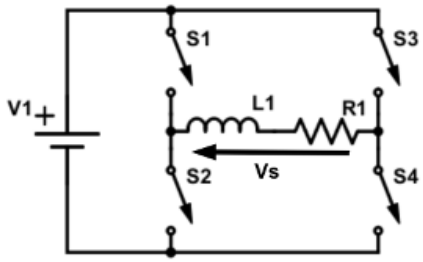

FIGURE 10: Schéma électrique simplifié du hacheur et de sa charge

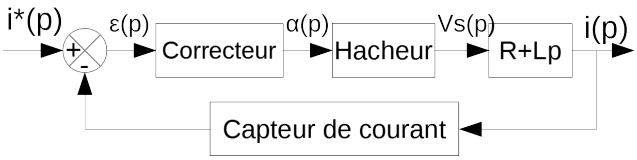

FIGURE 11: Schéma bloc simplifié du système souhaité. i^* est la consigne donnée par la Raspberry au DSP.

V. CONCEPTION DES LOIS DE CONTRÔLE RAPPROCHÉ

Une fois le gain du CAN déterminé et la transmission de commande au DSP par I^2C réalisable, il est possible de déterminer une loi de commande rapprochée, et surtout de l'implanter dans le DSP avant de faire des essais.

A. *Système à commander*

Le système à commander et asservir est composé, outre le pont en H, d'une charge résistive et d'une forte inductance (fig. 10). Une telle charge a pour propriété de composer un filtre passe-bas, et ainsi de transformer la tension modulée par MLI en courant à faibles oscillations.

Ainsi, on pourra essayer d'asservir le courant passant dans la charge en faisant varier le rapport cyclique du hacheur.

B. *Loi de commande*

On souhaite asservir le courant dans la charge, tel que présenté en Figure 11.

Si l'on dimensionne l'inductance correctement, elle n'aura pour effet que le lissage du courant à l'échelle d'une période de hachage, et sera sinon négligeable. On peut dans ces conditions considérer que le hacheur produit une tension Vs qui n'est autre que la tension moyenne sur une période de hachage.

Un correcteur proportionnel pourrait être utilisé. Une erreur statique serait présente, mais l'implémentation serait la plus simple à réaliser.

Le correcteur pourrait, dans un deuxième temps, être de type Proportionnel-Intégral, pour faire disparaître l'écart statique entre commande et sortie.

C. *Implémentation*

Un programme pour le DSP est réalisé avec *MATLAB*, qui implémente pour commencer un correcteur proportionnel (fig. 12).

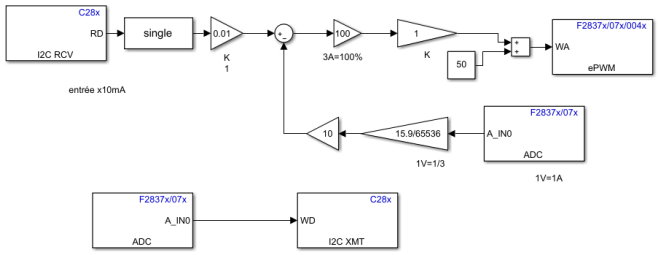

FIGURE 12: Programme du DSP pour l'asservissement du courant

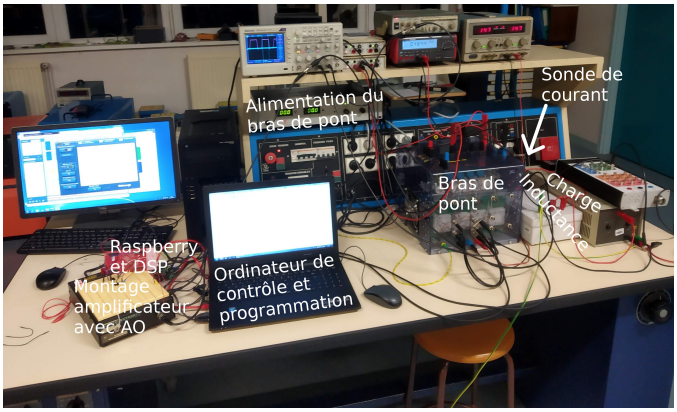

FIGURE 13: Montage du DSP, de la Raspberry, d'un pont en H et de la charge RL. On peut lire sur l'oscilloscope un signal rectangle, correspondant à la tension de commande du bras de pont.

Le montage est effectué, et le tout surveillé à l'oscilloscope (fig. 13).

Les observations des différents signaux ont montré que le DSP tente bel et bien d'asservir le courant dans la charge. Cependant, lors de l'essai, il s'est avéré que la sonde de courant utilisée comme capteur ne mesurait que les oscillations en éliminant les composantes basse fréquence. En conséquence, le comportement de l'ensemble était erratique, et ne permet pas de conclure fermement quant à la validité de ce montage expérimental.

L'implémentation reste donc à revoir et améliorer.

CONCLUSION

Durant ce projet de recherche, si tous les objectifs n'ont pas été remplis :

- la chaîne logicielle du DSP TI F28379D a été mise à l'essai et validée
- les capacités du même DSP on été testées, certains de ses défauts mis en évidence
- un bus de communication I^2C a été mis en place entre un DSP du même nom et un ordinateur léger type Raspberry Pi

L'enrichissement du composant d'émulation d'agent sur *smart grid* a donc connu plusieurs étapes d'avancement lors de ce projet, et poursuivra sa construction au travers d'autres travaux.

ANNEXE : PROGRAMME MAÎTRE

```
#!/usr/bin/python3
import sys
import smbus

# I2C channel 1 is connected to the GPIO pins
channel = 1
address = 80

# Initialize I2C (SMBus)
bus = smbus.SMBus(channel)

# Conversion des arguments de la ligne de commande
v=int(sys.argv[1])
w=int(sys.argv[2])

# Pour envoyer 1 seul octet
#bus.write_byte(address, v)

# Pour envoyer 2 octets
bus.write_block_data(address,2,[v,w])

# Lecture de l'octet envoy  par le DSP
print(bus.read_byte(address))
```