Montez à bord du bus Avalon : une interface de FPGA simplifiée

Auteur : Noe Quintero, Ingénieur d'application, groupe Signaux Mixtes, Linear Technology.

De nombreuses conceptions de FPGA modernes utilisent une version de processeur embarqué pour la commande. Une solution traditionnelle implique l'utilisation d'un processeur « softcore » embarqué tel que NIOS. Une autre solution consiste à utiliser un dispositif SoC qui comprend un processeur intégré « hardcore ». La figure 1 montre un système de FPGA d'Altera typique qui contient le processeur et un assortiment de périphériques connectés par l'intermédiaire du bus Avalon Memory Mapped (MM). Ces processeurs simplifient largement l'application finale, mais nécessitent une expérience solide en programmation et la connaissance de chaînes de compilation compliquées. Cela peut entraver le débogage, en particulier si un ingénieur « hardware » doit lire et écrire vers les périphériques par un moyen simple sans avoir à déranger l'ingénieur « software ».

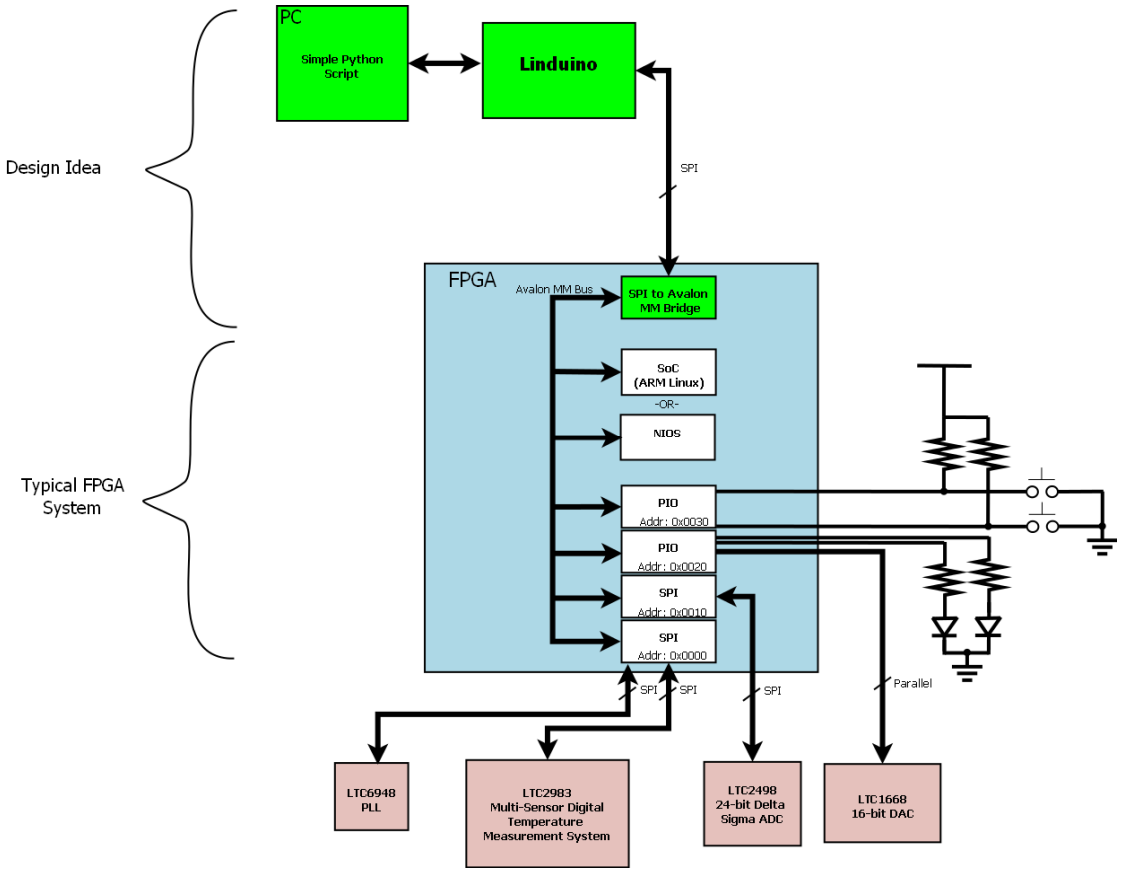


Figure 1. Système de FPGA d'Altera classique connecté par l'intermédiaire du bus Avalon Memory Mapped (MM)

Ici, l'idée est d'utiliser pour la conception un pont SPI esclave vers Avalon MM (*SPI Slave to Avalon MM Bridge)* d'Altera offrant une manière simple de sauter dans le bus Avalon. Cette technique présente deux avantages : elle ne compromet pas la conception du système original,

et le pont peut coexister avec le processeur embarqué. Pour le système présenté dans la figure 1, le pont SPI-Avalon MM permettrait à l'ingénieur de contrôler directement la fréquence de la PLL fractionnaires N LTC6948, de régler la tension du CNA LTC1668, de lire une tension en provenance du LTC2498, ou de lire des températures en provenance du LTC2983, exactement comme le processeur.

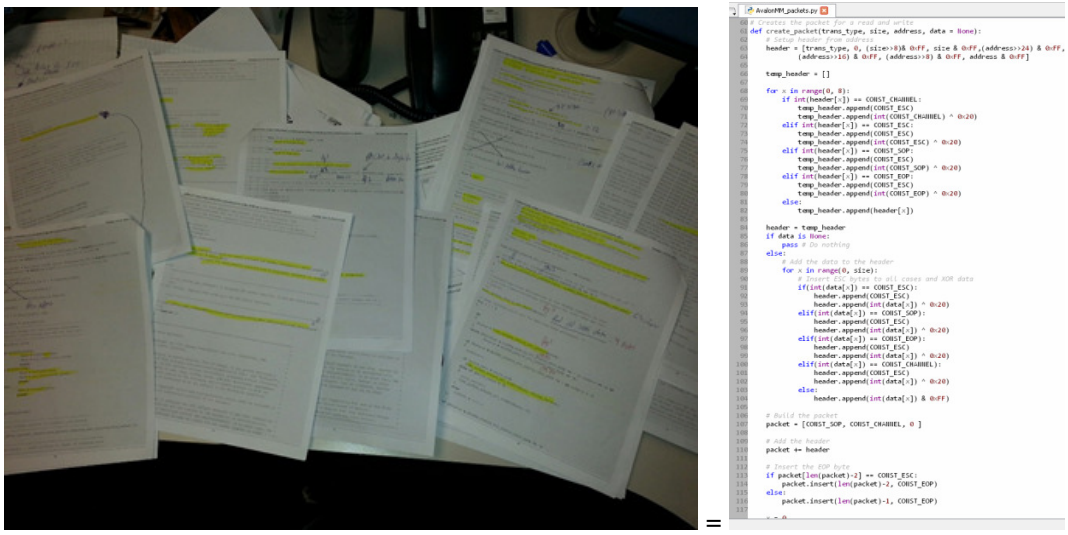


Figure 2. Surligneur + exemple de code + rétro-ingénierie = script Python

Altera fournit une conception de référence pour le pont SPI-Avalon MM. Malheureusement, la documentation est maigre, quand elle existe, et utilise un processeur NIOS comme SPI maître. Cela est contradictoire avec l'utilisation du pont SPI, puisque le processeur NIOS peut s'interfacer directement avec le bus Avalon MM. Une SPI maître pratique est le microcontrôleur Linduino® de Linear Technology, un clone de l'Arduino avec des caractéristiques supplémentaires pour s'interfacer avec les cartes de démonstration LT. Une de ces caractéristiques est un port SPI à niveau zéro rétabli. Cette fonction de rétablissement du zéro est particulièrement utile lors de l'interfaçage avec la banque d'E/S de l'FPGA à des tensions pouvant descendre jusqu'à 1,2V. Le micrologiciel Linduino peut être utilisé pour accepter des commandes par l'intermédiaire d'un port COM virtuel et traduire les commandes en transactions SPI.

Après la rétro-ingénierie de l'exemple de conception Altera, le côté gauche de la figure 2, une bibliothèque Python a été développée pour créer des paquets que le pont pourrait accepter. Ces paquets sont ensuite traduits en commandes Linduino. Un simple script Python permet alors à l'ingénieur « hardware » d'avoir le contrôle total du projet sans avoir besoin de réinventer les protocoles d'interfaçage. Un exemple de script Python pour contrôler la fréquence d'un générateur de schémas numériques pour un CNA LTC1668 est donné dans le dossier LinearLabTools Python disponible sur http://www.linear.com/solutions/linearlabtools. La figure 3 présente le montage de démonstration.

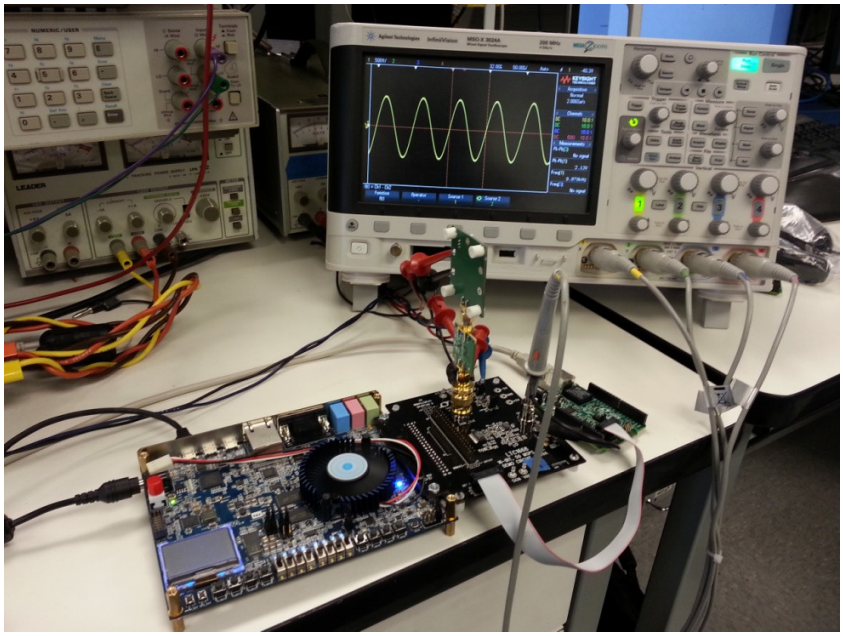


Figure 3. DC2459A en action

La figure 4 présente le schéma de principe du système de FPGA. Notons que l'oscillateur contrôlé numériquement (numerically controlled oscillator (NCO)) peut être contrôlé par le registre à décalage ou le cœur PIO. Le registre à décalage est inclus pour le débogage, car il permet de contrôler directement le NCO. Régler le niveau haut logique de la ligne du GPIO active le pont SPI-Avalon MM, qui à son tour commande un port PIO 32 bits par le bus Avalon MM. La sortie du PIO contrôle alors la fréquence du NCO.

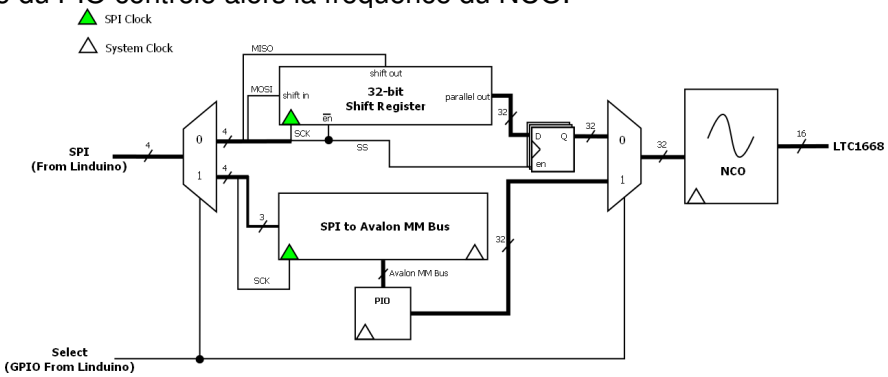


Figure 4. Schéma de principe du système de FPGA DC2459A

Lorsque le système le plus basique est opérationnel, d'autres cœurs IP périphériques Avalon peuvent être connectés au bus Avalon MM. Pour concevoir le système, Altera fournit un outil d'intégration de système appelé Qsys. Cet outil présente une IUG pour connecter les IP entre eux. Qsys est alors utilisé pour traduire le système de l'IUG en HDL verilog. L'IUG est représentée figure 5. Enfin, le système devra être ajouté au niveau le plus haut pour l'implémentation. Les adresses de l'IP sont entièrement configurables. Dans le cas donné en exemple, le PIO est établi à une position avec une adresse de base de 0x0.

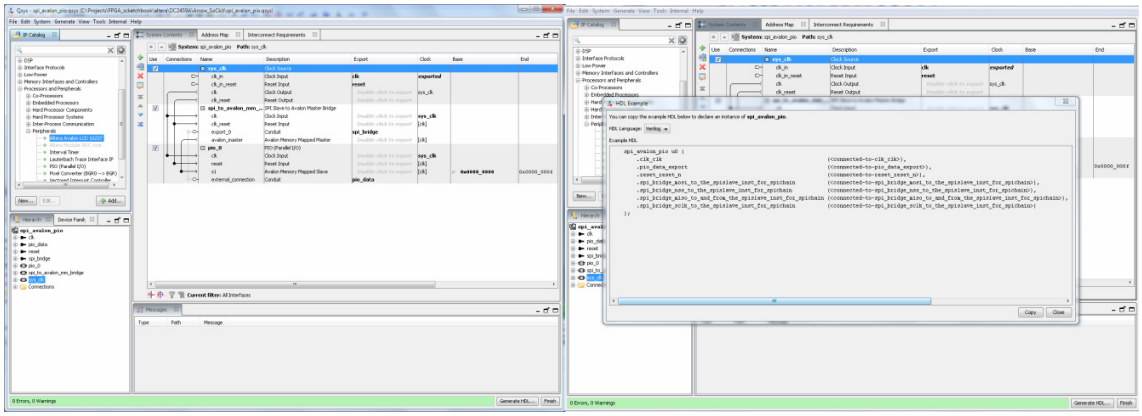


Figure 5. IUG Qsys

Une fois que la conception est implémentée dans le FPGA, la bibliothèque Python fournie dans LinearLabTools contient deux fonctions pour s'interfacer à la conception.

transaction_write(dc2026, base, write_size, data)
transaction_read(dc2026, base, read_size)

Le premier argument de ces fonctions est l'instance du port série du Linduino. Le deuxième argument est l'adresse du périphérique sur le bus Avalon. Les fonctions respectivement acceptent et retournent des listes d'octets. Ces deux fonctions sont écrites pour apporter de la flexibilité lors de l'écriture et la lecture vers IP. Pour régler le NCO dans l'exemple donné, la fonction transaction_write suffit. L'équation 1 sert à déterminer le mot de réglage.

Équation 1 : $$tuning\ word = \frac{desired\ frequency}{system\ clock\ frequency} \times 2^{32}$$

Pour régler le NCO à 1 kHz avec un taux d'échantillonnage de 50Méch/s, la valeur du mot de réglage est 85899. C'est-à-dire 0x00014F8B en hexadécimal, passé sous forme de liste de quatre octets. Le code Python pour régler le CNA à 1KHz est alors :

transaction_write (linduino_serial_instance, 0, 0, [0x0,0x01,0x4F, 0x8B])

Remarque : l'adresse de base du PIO est un zéro en provenance de la conception logique.



```
    The views and conclusions contained in the software and documentation are
    those of the authors and should not be interpreted as representing official
    policies, either expressed or implied, of Linear Technology Corp.

    Description:
        The purpose of this module is to use the DC2026C as a Avalon MM Bus
        interface to set the DC2459A frequency out.
"""

################################################################################
# Libraries
################################################################################

import sys
sys.path.append('../../utils')
import connect_to_linduino as duino
import ltc_spi_avalon as avalon


if __name__ == "__main__":
    linduino = duino.Linduino() # Find the Linduino
    linduino.port.write("M3")   # Set to SPI Mode 3
    linduino.port.write('G')    # Set the GPIO HIGH
    try:
        print "Command Summary"
        print "    1-Send raw code"
        print "    2-Set frequency"
        print "    3-Exit program"
        user_input = input("Enter a command: ")
        while(user_input != 3):
            if(user_input == 1):
                code = int(input( "Enter raw 32 bit code: "))

                # Send the data to the Avalon MM bus addr 0
                avalon.transaction_write(linduino, 0, 4,
                                         [code & 0xFF ,(code>>8) & 0xFF ,
                                         (code>>16) & 0xFF, (code>>24) & 0xFF])
            elif (user_input == 2):
                freq = float(input("Enter desired frequency(Hz): "))
                float_code = freq/50000000*(2**32-1)
                code = int(float_code)

                # Send the data to the Avalon MM bus addr 0
                avalon.transaction_write(linduino, 0, 4,
                                         [code & 0xFF ,(code>>8) & 0xFF ,
                                         (code>>16) & 0xFF, (code>>24) & 0xFF])
            else:
                print "**** Invalid Command ****"
            print "Command Summary"
            print "    1-Send raw code"
            print "    2-Set frequency"
            print "    3-Exit program"
            user_input = input("Enter a command: ")
    finally:
        linduino.close() # Close the port
```

```
Looking for COM ports ...
Available ports: [(23, 'COM24')]

Looking for Linduino ...
    Found Linduino!!!!
Command Summary
    1-Send raw code
    2-Set frequency
    3-Exit program
Enter a command:
```

Figure 6. Exemple de bus Python Avalon

Un script Python simple montré figure 6 est fourni pour faire la démonstration de l'interface de la conception de l'FPGA et du script Python. Il contient une interface texte simple pour configurer le NCO. Une remarque importante à faire est que le pont SPI Avalon utilise un mode SPI 3. Celui-ci a été déterminé comme étant le mode correct, péniblement, par essais successifs, et vérifié en analysant l'interface SPI du processeur NIOS dans l'exemple d'Altera.

En conclusion, cet exemple de projet démontre la possibilité de contrôler un système sans toucher du tout au processeur embarqué. Cela permet à l'ingénieur « hardware » de faire avancer le projet sans avoir à déranger l'ingénieur « software ». Les avantages de cette méthode peuvent être incorporés discrètement à l'FPGA sans que cela affecte la conception originale. L'ingénieur « hardware » peut se concentrer sur le « hardware ».

Le code Python est disponible sur : http://www.linear.com/solutions/linearlabtools
HDL pour la carte de démonstration du LTC1668 est inclus dans les fichiers de conception du DC2459, disponibles sur http://www.linear.com/demo/DC2459