Control an FPGA bus without using the processor

Noe Quintero - April 27, 2016

Many FPGA designs use an embedded processor for control. A typical solution involves the use of a soft processor such as a Nios, though FPGA SoCs with a built-in hard processor have become popular too. Figure 1 shows a typical Altera FPGA system that contains the processor and a mix of peripherals that are connected via Altera’s Avalon Memory Mapped (MM) bus. These processors greatly simplify the end application, but require a strong programming background and knowledge of complicated toolchains. This can hinder debug, especially if a hardware engineer needs a simple way to read and write to the peripherals without pestering the software engineer.

Figure 1  Typical Altera FPGA system connected using the Avalon memory-mapped bus
This Design Idea uses Altera's SPI Slave to Avalon MM Bridge to provide a simple way to hop onto the Avalon bus. There are two advantages to this technique: It does not compromise the original system design, and the bridge can co-exist with the embedded processor. For the system shown in Figure 1, the SPI bridge allows the engineer to directly control the frequency of the LTC6948 fractional-N PLL, set the LTC1668 DAC voltage, read a voltage from the LTC2498 ADC, or read temperatures from the LTC2983, just like the processor can.

Altera provides a reference design for the SPI-Avalon MM bridge. Unfortunately, the documentation is sparse at best, and uses a Nios processor as the SPI master. This effectively defeats the purpose of the SPI bridge, as the Nios can interface directly to the Avalon MM bus. A practical SPI master is Linear Technology’s Linduino microcontroller, which is an Arduino clone with extra features to interface to LT demo boards. One extra feature is a level-shifted SPI port. This level-shifting function is especially helpful when interfacing to FPGA I/O banks with voltages as low as 1.2V. The Linduino firmware can be used to accept commands through a virtual COM port and translate the commands to SPI transactions.

After reverse engineering the Altera example design (the left side of Figure 2), a Python library was developed to create packets that the bridge would accept. These packets are then translated into Linduino commands. A Python script then allows the hardware engineer to have complete control of the project without needing to reinvent the interfacing protocols. An example Python script to control the frequency of a digital pattern generator for an LTC1668 DAC is provided in the LinearLabTools Python folder. Figure 3 shows the demo setup.
Figure 3  DC2459 DAC demo board (R) plugged into an FPGA board (L)

Figure 4 shows the system block diagram. Note that the numerically controlled oscillator (NCO) can be controlled by the shift register or the PIO core. The shift register is included for debug, as it allows direct control of the NCO. Setting the GPIO line high enables the SPI-Avalon bridge, which in turn controls a 32-bit PIO port over the Avalon bus. The PIO output then controls the NCO frequency.
With the basic system operational, additional peripheral cores can be connected to the bus. To design the system, Altera provides a tool called Qsys, which provides a GUI to connect the IPs to one another. Qsys translates the GUI-designed system (Figure 5) to HDL. Peripheral addresses are fully configurable. In this case, the PIO is set to a base of 0x0.
Once the design is implemented in the FPGA, the provided Python library in LinearLabTools contains two functions to interface to the design:

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

The first argument to these functions is the Linduino serial port instance. The second argument is the peripheral’s address on the Avalon bus. The functions accept and return lists of bytes respectively. These two functions are written to allow flexibility when writing and reading to IP. To set the NCO for the provided example, the `transaction_write` function is all is needed. Equation 1 is used to determine the tuning word.

Equation 1:

\[
\text{tuning word} = \frac{\text{desired frequency}}{\text{system clock frequency}} \times 2^{32}
\]

To set the NCO to 1kHz with a 50MSPS sample rate, the tuning value is 85899, or 0x00014F8B, which is passed as a list of four bytes. Thus, the python code to set the DAC to 1kHz is:

```python
transaction_write(linduino_serial_instance, 0, 0, [0x0, 0x01, 0x4F, 0x8B])
```
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 an Avalon MM Bus interface to set the DC2459A frequency out.

```
# Libraries
import sys
sys.path.append('.../utils')
import connect_to_linduino as linduino
import itc_spi_avalon as avalon

if __name__ == '__main__':
linduino = linduino.Linduino()  # Find the Linduino
linduino.port.write('MB')  # Set to SPI Mode 3
linduino.port.write('G')  # Set the SPI O HIGH
try:
    print "Command Summary"
    print "  1-Send raw code"  # 7-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_writes(linduino, 0, [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/500000000*(2**32-1)
            code = int(float_code)
            # Send the data to the Avalon MM bus addr 0
            avalon.transaction_writes(linduino, 0, [code & 0xFF, (code>>8) & 0xFF,
                                                    (code>>16) & 0xFF, (code>>24) & 0xFF])
        else:
            print "## Invalid command ##"
    print "Command Summary"
    print "  1-Send raw code"  # 7-Set frequency
    print "  3-Exit program"
    user_input = input("Enter a command: ")
finally:
linduino.close()  # Close the port
```

Looking for COM ports ...
Available ports: [{'COM24', 'COM23'}]

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

Figure 6  Python Avalon bus example
The Python script in Figure 6 illustrates the simple text interface that configures the NCO. An important note: the bridge uses SPI mode 3. This was painfully determined to be the correct mode by trial and error, and verified by analyzing the Nios processor’s SPI interface in Altera’s example.

This Design Idea provides the ability to control a system without touching the embedded processor, allowing the hardware engineer to progress on a project without bothering the software engineer, and with minimal impact to the hardware design.


HDL for the LTC1668 demo board is included in the DC2459 design files, available at http://www.linear.com/demo/DC2459.