© Bert Cuzeau / mailto:alse@compuserve.com

Walking Bit & Random Numbers on the XS40 Board

(Interfacing the FPGA and the Microcontroller)

Introduction

I have written this very simple example after having spent some time figuring out how to use the 8031-FPGA combination. (I did not have the Foundation software so I could not open any of the examples coming with the board).

I also experienced Xilinx Mapper problems while trying to implement the asynchronous solutions suggested in several examples.

This project does only require a(ny) VHDL synthesizer, and the Xilinx Alliance Base product.

For the board architecture please refer to the superb XESS documentation.

The Software

To test the bus interface and the design flow, I re-used the ideas from two example designs that can be found on the XESS CD-Rom (and Web) -but only as Foundation projects-…

This example, too, runs a small 8031 code which interacts with the FPGA :

(a) It walks a “one” in the A register 32 times, at a few Hertz pace, and writes register “A” to a memory-mapped register located in the FPGA at address F000 hex.
This register directly drives the 7-segment LED display located on the board.

(b) It then writes a seed (01) in another register, mapped at E000 hex.
This register is an LFSR which generates a pseudo-ransom sequence each time it is shifted. This shift occurs whenever a write (does not care what value) is done at D000.

(c) it writes dummy data at address D000 hex, at a few hertz pace, to shift the register,

(d) It reads the new LFSR value at E000 hex, and writes it to the LED register.

(e) When the sequence (c).. (d) has been done 32 times, it loops backs to (a).

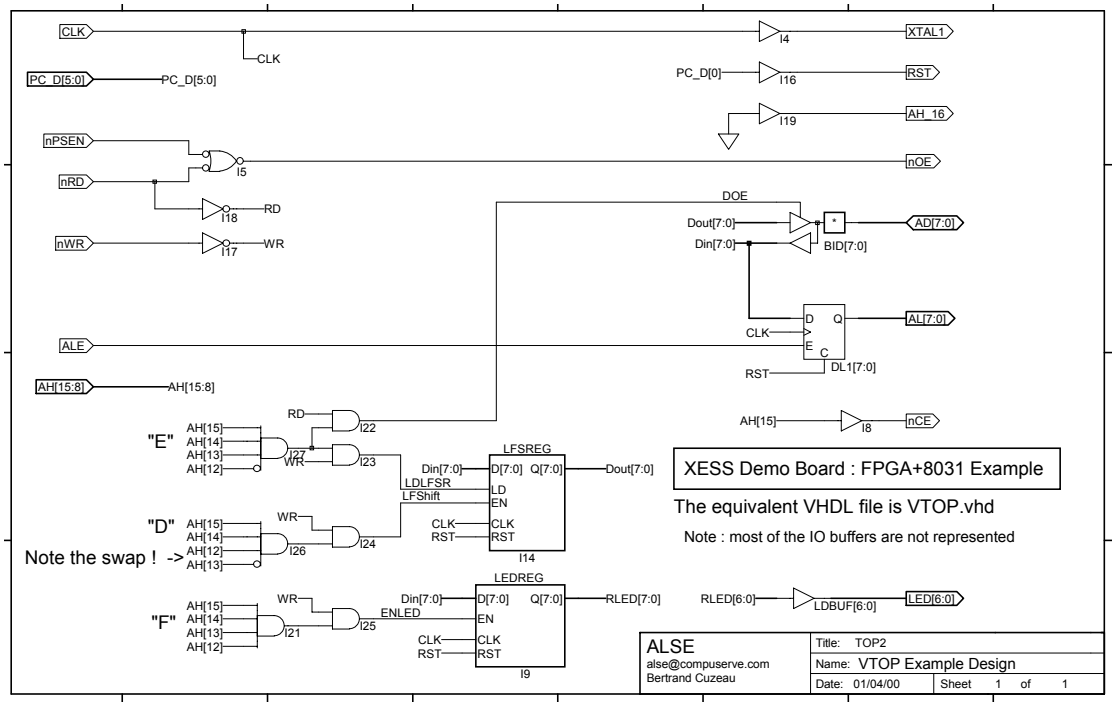
Though not very sphisticated, this program does test the FPGA interaction in many ways.

This program, in assembly language, is listed next page.

```
; WALK.asm
; ------------------------------------------------------
;   Example for XESS board 8031, linked with VTOP FPGA
; ------------------------------------------------------
; Bertrand Cuzeau
; alse@compuserve.com
;
; This works as follows:
;  - a bit is walked four times on the LED display
;  - a seed is then written in the LFSR
;  - the LFSR is shifted 32 times, read back,
;    and written to the LED
;  - the system cycles back to the beginning of the test
; This is probably not a very elegant code, but I am
; not familiar with the 8031...
;
; Don't forget to Pulse D0 in the parallel port (reset) !

        $MOD51

STACKTOP EQU    70H             ; start of stack (grows up)
LEDREG   EQU    0F000H          ; LED register address
LFSR     EQU    0E000H          ; LFSR R/W register address
LFShift  EQU    0D000H          ; LFSR shift command

        DSEG
        ORG     30H
CNTR:   DS      1               ; counter for wait routine
CNT2:   DS      1               ; counter for bit walk test

        CSEG
        ORG     0000H           ; program starts at 0 after reset
START:
        MOV     SP,#STACKTOP    ; initialize stack pointer ...

; --- Walking bit LED Test ---
START2:
        MOV     A,#1            ; A(0) <= 1
        MOV     DPTR,#LEDREG    ; point on FPGA LED register
        MOV     CNT2,#32        ; 4 passes
LOOP:
        RL      A
        MOVX    @DPTR,A         ; show A the LED digit
        CALL    WAIT            ; delay so we can see the bits ...
        DJNZ    CNT2,LOOP

; --- LFSR Test ---
        MOV     A,#1            ; Seed
        MOV     DPTR,#LFSR      ; point on E000=FPGA LFSR register
        MOV     CNT2,#32        ; 32 passes
        MOVX    @DPTR,A         ; Store Seed
LOOP1:
        MOV     DPTR,#LFSR      ; point on E000=FPGA LFSR register
        MOVX    A,@DPTR         ; Read LFSR
        MOV     DPTR,#LEDREG    ; point on FPGA LED register
        MOVX    @DPTR,A         ; Write to the LED digit
        CALL    WAIT            ; delay so we can see the bits ...
        MOV     DPTR,#LFShift
        MOVX    @DPTR,A         ; Dummy write to shift
        DJNZ    CNT2,LOOP1

        JMP     START2

WAIT: ; This subroutine waits about 1/8th second
        PUSH    ACC
        PUSH    B
        MOV     CNTR,#2         ; 1 = ~ 1/8th sec
WAIT1:
        MOV     B,#255
WAIT2:
        MOV     A,#255
        DJNZ    ACC,$
        DJNZ    B,WAIT2
        DJNZ    CNTR,WAIT1
        POP     B
        POP     ACC
        RET
        END
```

The FPGA

The solution adopted is fully synchronous. It is also entirely VHDL-Based, though we have reproduced below the equivalent schematics for the Top-Level :



Caveat : this design has not been checked against timing specifications from the 8031 chip. It works on my board, but this does not prove that the assumptions it is based on are correct, nor they would remain correct with another brand of microcontroller…

The module “LEDREG” is just simple G_DEC FlipFlops… We made a module of it just for the case where some decoding would have to be included.

The module LFSREG is a Linear Feedback shift register.

As we can see, the data is read back only from address E000 hex (LFSREG output). Otherwise, multiplexers (or tbufs) could have been used for that purpose.

This schematic is provided under its VHDL form, so **no graphical tool is necessary**.
The description is 100% portable among all (decent) synthesis tools available on the market.

Project Files

The archive includes the following files :

File Name	Description
G_BIDIR.vhd	optional : Bi-directional buffer
G_DEC.vhd	optional : D flipFlip with Enable & Clear
LFSREG.vhd	Random Number Generator
LEDREG.vhd	Led register
VTOP.vhd	Top-Level description
XL40.ucf	Pin Lock constraint file
XC4010XL.bit	Ready-to-use download file ***for XC4010XL***
WALK.asm	Assembler program file
WALK.hex	Binary program file
WALKBIT.doc	This documentation (Word-Format)
README.txt	Quick text-based description

Note : The two first vhdl modules (Bidir and D-FlipFlop) are not necessary. They can be used to demonstrate the use of "Generate" VHDL statements (which come commented out).

It would have been trivial (in fact easier) to include everything in a single file (top-level), but it is not the way to handle more complex projects...

This project can easily serve as a platform for more ambitious designs (enhancing the decoder process, adding other submodules, etc....). But PLEASE : before doing this, you should check carefully the timing information of the 8031 chip and ensure that the synchronous solution implemented here is absolutely correct.

Pinout

The pinout is indeed extremely important. Any error at this stage and the design won't work at all –if nothing worse happens-.

Notice the AH_16 pin added at the end, important if you have a 128k RAM.

Last : this pinout is okay for the XS40-010XL board that I own. Make sure it is appropriate for *your* board ! *XS95 boards have different pinout.*

```
NET CLK         LOC=P13; # CLOCK FROM EXTERNAL OSCILLATOR
#
# LED DRIVER OUTPUTS
NET LED(0)      LOC=P25; # d
NET LED(1)      LOC=P26; # c
NET LED(2)      LOC=P24; # e
NET LED(3)      LOC=P20; # g
NET LED(4)      LOC=P23; # b
NET LED(5)      LOC=P18; # f
NET LED(6)      LOC=P19; # a
#
# DATA BITS FROM THE PC PARALLEL PORT
NET PC_D(0)     LOC=P44;
NET PC_D(1)     LOC=P45;
NET PC_D(2)     LOC=P46;
NET PC_D(3)     LOC=P47;
NET PC_D(4)     LOC=P48;
NET PC_D(5)     LOC=P49;
//NET PC_D(6)   LOC=P32; # MUST USE SPECIAL-PURPOSE PINS FOR
//NET PC_D(7)   LOC=P34; #   ACCESSING PC_D(6) AND PC_D(7)
#
# MICROCONTROLLER PINS
NET XTAL1       LOC=P37; # uC CLOCK
NET RST         LOC=P36; # uC ACTIVE-HIGH RESET
NET ALE         LOC=P29; # uC ACTIVE-High ! ADDRESS LATCH ENABLE
NET nPSEN       LOC=P14; # uC ACTIVE-LOW PROGRAM-STORE ENABLE
NET nRD         LOC=P27; # uC ACTIVE-LOW READ
NET nWR         LOC=P62; # uC ACTIVE-LOW WRITE (ALSO CONTROLS RAM)
#
NET AD(0)       LOC=P41; # MULTIPLEXED ADDRESS/DATA BUS
NET AD(1)       LOC=P40;
NET AD(2)       LOC=P39;
NET AD(3)       LOC=P38;
NET AD(4)       LOC=P35;
NET AD(5)       LOC=P81;
NET AD(6)       LOC=P80;
NET AD(7)       LOC=P10;
#
# Ext RAM Pins (FPGA Outs) :
NET AL(0)       LOC=P3;  # DEMUXED Addr LOW
NET AL(1)       LOC=P4;
NET AL(2)       LOC=P5;
NET AL(3)       LOC=P78;
NET AL(4)       LOC=P79;
NET AL(5)       LOC=P82;
NET AL(6)       LOC=P83;
NET AL(7)       LOC=P84;
#
# Addr High from uC to FPGA & Ram
NET AH(8)       LOC=P59; #
NET AH(9)       LOC=P57;
NET AH(10)      LOC=P51;
NET AH(11)      LOC=P56;
NET AH(12)      LOC=P50;
NET AH(13)      LOC=P58;
NET AH(14)      LOC=P60;
NET AH(15)      LOC=P28;
#
# RAM CONTROL PINS
NET AH_16       LOC=P16; # Upper 64 k Enable
NET nOE         LOC=P61; # ACTIVE-LOW OUTPUT ENABLE
NET nCE         LOC=P65; # ACTIVE-LOW CHIP ENABLE
```

Design Flow

It is based on any VHDL synthesizer associated to Alliance 2.1i.
If you follow the instructions below step-by-step, everything should be done in a few minutes.

The Asm file is also delivered in .hex format so we don't have to asemble it.

- Create a **new directory** and unpack the files listed above.
- Launch your perferred **VHDL Synthesis** tool.
- Create a **new project** in your new directory.
- Add the sources files **LFSREG**.vhd, **LEDREG**.vhd and **VTOP**.vhd.
  VTOP should be the top-level (the last in the list if you use Synplicity)
- Select the proper **target** (device) for your board,
- **Synthesize** the project. (select the VTOP if you use FPGA Express)
  You should only have one normal warning (we have one unused bit in LEDreg).
  The Edif file should be done (**vtop.edf**).
- **Quit** your synthesis tool.

- Launch **Xilinx Design Manager**
- **File-New Project-Browse**.
  Go to your directory and pick **vtop.edf**
  Click on **Ok** (use the default work directory)
- In the **Constraints** File box, select "**Custom**"
- Browse back to your dir and select **XL40.UCF**
  Click on **Ok**.
  Click on **OK**.

Now we're back to the main screen.

- Click on the black arrow (**Design-Implement**). You should soon end up with a properly implemented chip, with a **bit file** ready to use !

- **Quit** Design Manager.

- Open an **MS-DOS window.**

- Type : `xsload walk.hex vtop.bit`

- Run the **GXSPORT** utility to toggle the D0 bit to 1 then back to 0, in order to reset properly the 8031.

You should see the walking bit followed by 32 pseudo-random patterns on the LED display.

PS: If your board doesn't seem to work, try first the GXSTEST. If it doesn't pass, you could then try to reprogram the clock generator as indicated in the documentation.

I hope this example will be useful for those who are focused on VHDL methodologies.

Bert Cuzeau
ASIC & FPGA Design Expert
Doulos HDL Course Leader
alse@compuserve.com