메모(Technical)

Document No.: MJL-LD-AN-4
Author: 하태욱 [twha@mjl.co.kr]
Version: 1.0
Date: 2000 년 3 월 31 일
Subject: **Microprocessor 을 이용한 FPGA Passive Serial Configuration.**

---

1. FPGA 에 Configuration data 을 실어주는 전형적인 방법은 FPGA 전용 Serial PROM (EPC1441, EPC1, EPC2, 기타등등)을 이용하는 방법일 것이다.

2. PCB Board 상에 Microprocessor 가 없을 경우에는 반드시 위에서 언급한 1 번 방식을 사용 해야 하지만, Microprocessor 가 있고 여분 (FPGA configuration data 의 용량)의 System ROM 이 있을 경우에는 Microprocessor 를 사용하여 FPGA 에 Configuration data 를 실어 줄 수가 있다.

3. Altera 의 FPGA 계열 (APEX 20K/E, FLEX 10K/E/S/, FLEX 6000)도 Microprocessor 를 이용한 Passive serial configuration 을 이용할 수가 있다.

4. 이 자료는 특정 CPU (Intel 8051 계열)를 사용하여 FLEX 6000 Series FPGA 에 Configuration data 를 실어 주는 방법의 한 예를 기술하고자 한다.

5. 만일 사용자가 위에서 언급한 Microprocessor 와 다른 것을 사용할 경우에도 해당되는 Microprocessor Object Code 를 만들어 주는 Compiler 를 사용하고 아래에서 기술하는 순서에 입각해서 Microprocessor 와 FPGA 를 연결하는 Passive Serial Communication Program 을 만들어 주면 쉽게 소정의 목적을 달성할 수 있을 것이다.

6. Timing information 및 순서에 대한 자세한 자료는 아래의 문서와 Web site 를 참조하길 바란다.

   - 문서 명: AN 116 (Configuring APEX 20K, FLEX 10K & FLEX 6000 Devices), ver. 1.02, December 1999
   - 위치: http://www.altera.com/document/an/an116.pdf
   - 문서 명: Here is a “altera.PLDWorld.com”… / Tool Manual…
   - 위치: http://www.pldworld.com/@altera/html/technote/toolmanual.htm

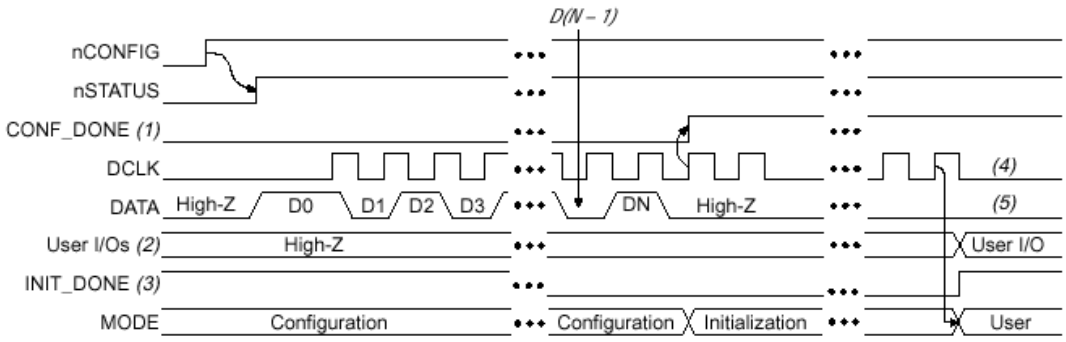
Device Configuration Overview

디바이스가 동작되고 있는 동안에는, APEX 20K, FLEX 10K, 그리고 FLEX 6000 디바이스들은 Configuration data 를 디바이스 내의 SRAM Cell 안에 저장해 둔다. SRAM Memory 는 휘발성이기

때문에, SRAM Cell 들은 디바이스에 전원이 인가될 때 마다 Configuration data 를 반드시 실어와야만 될 것이다. APEX 20K, FLEX 10K, 그리고 FLEX 6000 디바이스들이 한번 구성(Configuration)이 되면, 디바이스의 Register 와 I/O pin 들은 반드시 초기화가 될 것이다. 초기화가 진행이 된 후, 디바이스는 In-system operation 을 위해 사용자 방식(User mode)으로 들어 갈 것이다.

그림 1 은 Configuration, Initialization, 그리고 User mode 동안의 디바이스의 상태를 보여준다.

Figure 1. APEX 20K, FLEX 10K & FLEX 6000 Configuration Cycle



Notes:

(1) During initial power-up and configuration, CONF_DONE is low. After configuration, CONF_DONE goes high. If the device is reconfigured, CONF_DONE goes low after nCONFIG is driven low.

(2) User I/O pins are tri-stated during configuration. APEX 20K and FLEX 10KE devices also have a weak pull-up resistor on I/O pins during configuration. After initialization, the user I/O pins perform the function assigned in the user's design.

(3) When used, the optional INIT_DONE signal is high when nCONFIG is low before configuration and during approximately the first 40 clock cycles of configuration.

(4) DCLK should not be left floating. It should be driven high or low, whichever is more convenient.

(5) DATA (FLEX 6000 devices) and DATA0 (APEX 20K and FLEX 10K devices) should not be left floating. It should be driven high or low, whichever is more convenient.

APEX 20K, FLEX 10K, 그리고 FLEX 6000 디바이스에 대한 Configuration data 는 Active 또는 Passive configuration scheme 을 이용하여 실어질 수 있다. Configuration device 를 가지고 Active configuration scheme 을 이용할 때는, Target device 와 Configuration device 둘 다 Control 과 Synchronization signal 을 발생한다. 두 디바이스가 Configuration 을 시작할 준비가 될 때, Configuration device 는 Data 를 APEX 20K, FLEX 10K, 그리고 FLEX 6000 디바이스에 전송한다.

Passive configuration scheme 를 사용할 때는 APEX 20K, FLEX 10K, 그리고 FLEX 6000 디바이스는 Configuration process 를 조정하는 Intelligent host 또는 Microprocessor 를 가진 시스템에 통합되어서 운용된다. Host 는 저장디바이스(Hard Disk, RAM, 또는 다른 시스템 메모리)에서 Configuration data 를 공급 받는다. Passive configuration 을 사용할 시, 사용자는 시스템이 운용중이라도 configuration data 를 변경시켜 다시 디바이스를 Reconfigure 함으로써 Target device 의 기능을 변경시킬 수 있다. 또한 시스템 사용자들에게 변경된 (새롭게 만든) Programming file 을 배포함으로 In-field upgrade 를 구축할 수 있다.

PS(passive Serial) Configuration with a Microprocessor

Microprocessor 를 가지고 PS Configuration 을 함에 있어, Microprocessor 는 Storage device 에서 Data 를 Target APEX 20K, FLEX 10K, 또는 FLEX 6000 device 에 Programming hardware 를 통해서 전달한다. Configuration 을 초기화하기 위해, Microprocessor 는 반드시 nCONFIG pin 상에 하나의 Low-to-high transition 신호를 발생시켜야 하며 Target device 는 반드시 nSTATUS pin 을 Release

해야만 한다. Microprocessor 또는 Programming hardware 는 Target device (the DATA0 pin for APEX 20K and FLEX 10K devices, and the DATA pin for FLEX 6000 devices)의 DATA pin 상에 한 Time 에 Configuration data one bit 를 위치시킨다. 각각의 Data byte 의 Least significant bit(LSB)가 반드시 첫번째로 보내져야만 한다. Data 는 CONF_DONE 신호가 High 로 올라갈 때까지 Target device 안으로 지속적으로 Clocking 되어져 보내진다.

모든 Data 가 전송 되어진 후, DCLK 는 FLEX 10K 그리고 FLEX 6000 device 에 대해서는 부가적 인 10 번의 DCLK 을 보내야만 하며 APEX 20K device 는 부가적인 40 번의 DCLK 을 보내야만 한다. 이렇게 하는 것은 Device 를 초기화하는 데 필요한 동작이기 때문이다. Device 의 CONF_DONE pin 은 초기화를 시작하는 것과 성공적으로 Configuration 이 된 것을 알리기 위해 High 로 올라간다.

Max+plus II 또는 Quartus software 는 초기화를 위한 여분의 Bits 를 Configuration file 을 만들 때 첨가 시킨다. Configuration 후에 Device 에 DCLK 를 구동하는 것은 디바이스 동작에 아무런 영향을 주지 않는다. 그러므로 전체 Configuration file 을 Target device 에 전송하는 것은 Target device 를 Configuration 하고 Initialize 하는 데 충분하다.

Handshaking signal 은 PS Configuration mode 에서는 사용이 되질 않는다. 그러므로 Configuration clock speed 는 정확한 Configuration 을 충족시키기 위해 정의된 Frequency 보다 낮게 반드시 동작시켜야 한다. 최대 DCLK 주기는 존재하지 않는다. 사용자는 DCLK 를 Halt 함으로 Configuration 을 잠시 중단 할 수 있다.

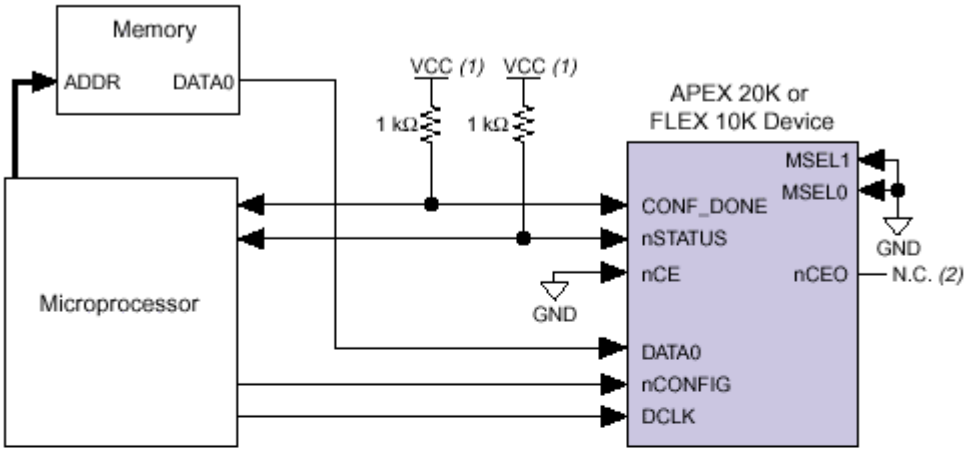
만일 Target device 가 Configuration 동안에 Error 를 감지하면, Target device 는 Microprocessor 에 경고를 하기 위해 자기의 nSTATUS pin 을 Low 로 구동 시킨다. Microprocessor 는 그때 Configuration process 를 다시 시작하기 위해 nCONFIG pin 에 한번의 Low-to-high pulse 를 만들 수 있어야 한다. 첨부적으로 MAX+plus II 또는 Quartus software 에서 “Auto-Restart Configuration on Frame Error” option 이 만일 사용하도록 선택되어 졌다면, Target device 는 한번의 Reset time-out period 후에 nSTATUS 를 Release 한다. nSTATUS 가 Release 된 후에 Microprocessor 는 nCONFIG 를 Low 로 한번 Pluse 신호를 만들 필요 없이 Target device 를 Reconfiguration 할 수 있다.

Microprocessor 는 성공적인 Configuration 을 충족시키기 위해 CONF_DONE 과 INIT_DONE pin 을 항상 감시 할 수 있어야 한다. 만약 Microprocessor 가 모든 data 을 다 보내고 Initialization clock 를 보내기 시작하지만 CONF_DONE 이 High 로 가지 않으면, Microprocessor 는 Target device 를 반드시 다시 Reconfiguration 해야만 한다.

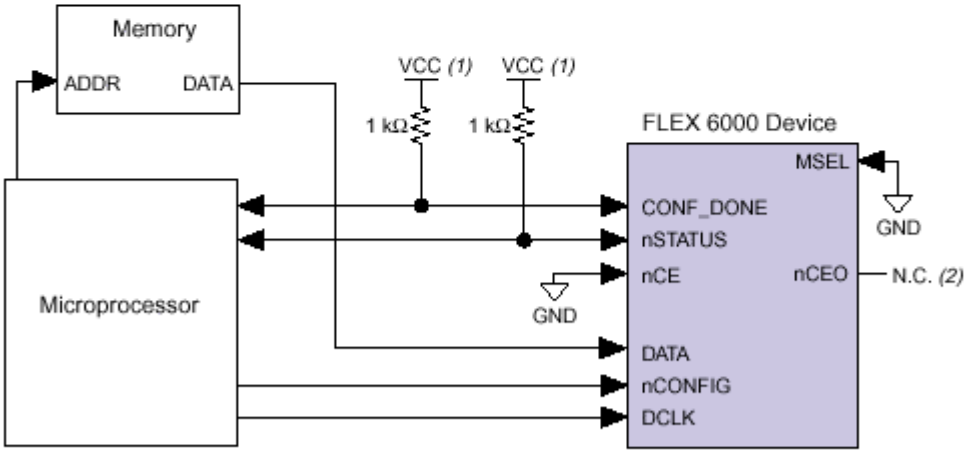
아래의 그림을 참조하면 Configuration diagram 을 볼 수 있다.

Figure 11. PS Configuration Circuit with Microprocessor

APEX 20K or FLEX 10K Devices



FLEX 6000 Devices



Notes:

(1) The pull-up resistor should be connected to any V_{CC} that meets the device high-level input voltage (V_{IH}) specification.

(2) The nCEO pin is left unconnected.

iNTEL 8051 계열 Microprocessor 와 FLEX6000 을 사용한 Passive Serial Configuration 의 실전 예제

위에서 기술한 내용들은 AN116.PDF 에 있는 내용을 한글화 작업을 한 것이다. 이것만 가지고 바로 작업을 하기에는 초보자들이 보기에는 어려움이 있을 것이다. 해서 하나의 예제를 가지고 작업을 진행하고자 한다.

1) 하나의 EPF6016AQC240-3 과 8051 Microprocessor 를 사용한다.

2) 우선 FPGA 설계가 다 완료되었으면 Microprocessor 가 사용할 Configuration data 를 변환하는 작업이 필요하다.

3) 최종 디자인을 Compile 하면 *.sof 와 *.pof file 이 만들어 질 것이다. 이 파일을 바로 사용할 수 는 없고 데이터를 변환하는 작업이 필요하다.

4) MAX+plus II 에서 Pull-down menu 의 File -> Convert SRAM Object Files... 을 선택하여 아래 그림을 참조하여 작업을 진행한다.

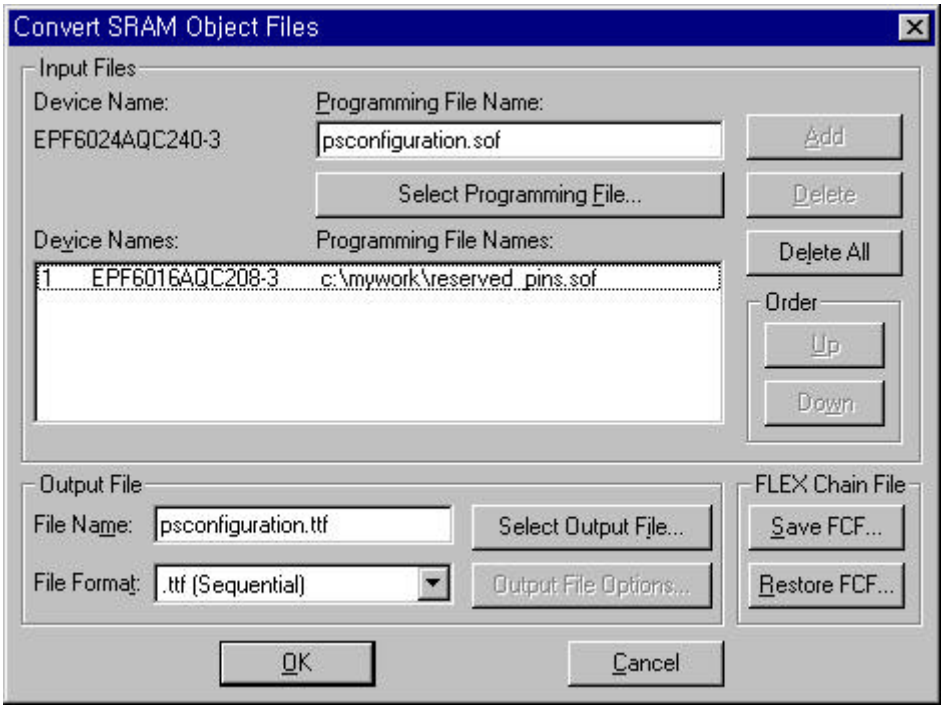


그림 1 Convert SRAM Object Files

5) 위의 그림 3 에서 보는 것과 같이 Output File 에서 File Format 을 .ttf(Sequential)로 선택한 후 OK button 을 누른다.

6) 실제적으로는 MAX+plus II Software 는 디자인 Complie 시 3 가지 형태의 Configuration data 가 만들어 진다.

*.sof ; ByteBlasterMV cable 을 사용하여 Configuration 할 때 사용하는 Data
*.pof ; Configuration PROM 을 사용할 때 사용하는 Data
*.ttf ; Passive Configuration with a Microprocessor 를 사용할 때 이용하는 Data

그러나 여기서 만들어 지는 *.ttf 를 바로 사용하지 말고 4), 5)과정을 거쳐 Sequential type 의 ttf file 를 만들어 낸다.

7) 지정된 위치에 *.ttf file 이 만들어 졌는지 확인한다.

8) Microprocessor 의 Compiler 가 사용할 수 있는 형태의 Data 로 변환하기 위해 "hex_conv.exe" Program 을 사용하여 *.ttf file 을 DB (Define Byte) 형태의 file 로 변환한다.

a. Windows 98 / Windows NT 의 바탕화면상에 DOS 창을 띄운다.

b. DOS 창에서 hex_conv.exe 가 있는 Directory 로 이동한다.

c. 만들어진 *.ttf file 을 hex_conv.exe 가 있는 Directory 로 Copy 한다.

d. DOS prompt 에서 hex_conv.exe 를 실행시키면 입력파일명을 물어본다. 여기서 사용자가 만든 *.ttf 파일명을 넣어준다. 자동으로 출력파일은 fpga.ttf 가 된다.

9) 상기 작업까지 진행을 하면 Microprocessor 와 SYSTEM ROM 에 저장할 FPGA Configuration data 는 준비가 된 것이다.

이제 남아 있는 작업은 "C language 또는 Assembler language"를 사용하여 Microprocessor 와 FPGA 간의 통신 프로그램을 만드는 것이다. 아래 한 예를 보여주고 있다.

```
;;;*** altera Configuration interface
pga_dclk    bit             p1.0
pga_data    bit             p1.1
pga_status          bit             p1.2
pga_conf_done       bit             p1.3
pga_init_done       bit             p1.4
pga_conf    bit             p1.5
;;;;;;;;;;;;;; u-com Emulator Test sub-rutine Kevin
;;
;;              jmp     restart
;;
;;;;;;;;;;;;;; u-com Emulator Test sub-rutine Kevin
fpga_init:
                mov     r0,#05
                mov     r1,#03
                clr     pga_dclk
                clr     pga_data
fpga_conf_init_loop:
;;              call    pga_clock
;;              djnz    r0,fpga_conf_init_loop
fpga_conf_init_clr:
                clr     pga_conf
;;              clr     pga_status
fpga_conf_init_clr_loop:
;;              call    pga_clock
                djnz    r1,fpga_conf_init_clr_loop
                nop
                nop
                nop
                nop
                nop
                nop
                setb    pga_conf
;;              setb    pga_status
fpga_status_wait:
                jnb     pga_status,fpga_status_wait
                mov     r0,#0ffh
fpga_fir_data_wait:
                djnz    r0,fpga_fir_data_wait
                mov     dptr,#fpga_code_data_end
                mov     temp1,dpl
                mov     temp2,dph
fpga_data_loading_begin:
                mov     dptr,#fpga_code_data
                clr     a
```

```
fpga_data_loading_loop:
                    mov     r0,#08
                    clr     a
                    movc    a,@a+dptr
fpga_data_down_loop:
                    rrc     a
                    mov     pga_data,c
                    setb    pga_dclk
                    clr     pga_dclk
;;                  call    pga_clock
                    djnz    r0,fpga_data_down_loop
                    inc     dptr
                    mov     a,dpl
                    jnb     pga_status,fpga_init
                    cjne    a,temp1,fpga_data_loading_loop
                    mov     a,dph
                    cjne    a,temp2,fpga_data_loading_loop
;;                  jnb     pga_conf_done,fpga_init
                    mov     r0,#45
fpga_down_init_done:
                    setb    pga_dclk
                    clr     pga_dclk
                    djnz    r0,fpga_down_init_done
                    setb    pga_dclk
;;                  jnb     pga_init_done,fpga_init
restart:
                        clr     TR0         ; run tmr 0
                        clr     TR1         ; TMR 1 DO NOT RUN
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; FPGA.ASM FILE
;; FILE INCLUDE RUTINE
fpga_code_data:

$include(fpga.ttf)

fpga_code_data_end:
```

위의 프로그램은 iNTEL8051 Assembler language 를 사용하시는 분은 해석할 수 있으리라 생각하고 설명은 달지 않는다.

가장 중요한 것은 FPGA 와 Microprocessor 와의 통신규약을 정확하게 이해하는 것이라 생각이 든다. 순서를 이해가 될 때까지 반복적으로 암기한 후에, 통신 프로그램을 짜면 별다른 문제점이 발생할 부분이 없을 것으로 생각이 된다.

위의 그림 11 을 참조하여 Microprocessor 와 FPGA 간의 결선을 하여 작업을 진행한다.

Revision History

- 2000-3-31 - Ver 1.0: Initialize Release…
- 2000-4-14 – Ver 1.1: **내용보완 및 추가**… (Modified by C.W.Yang)