Project Report

on

# Developing an interface to download the FPGA configuration file to a EEPROM device via VME bus to reconfigure FPGA at power up.

Submitted by
Abhishek Surana
(Indian Institute of Technology, Delhi)

Under the supervision of
B.Satyanarayana and Mandar Saraf

Guided by
Prof. N.K. Mondal

Submitted on

29th December 2009

# Acknowledgment

I sincerely thank Prof. N.K. Mondal and B.satyanarayana.for providing me the lab facilities in INO lab for completing my project work. My special thanks to Mr Mandar Saraf for always being so patient with me and allowing me explore and do things as i wanted to. I would like to acknowledge all the lab members, especially Mr.R.R Shinde, Mr.Manas ,Mr.Shekhar, Ms. Darshna,Ms.Sonali for their help through out my project work.I would like to thank Mr. Deepak Samuel, Mr. Saleem, Mr. Sumanto Pal and Ms. Sudeshna for their constant support . A heartily thanks again for Mr. R Shinde and Mr. Manas for allowing me to participate in RPC building process. I would also like to thank Mr. Piyush Verma for making me part of Babha centenary celebrations which was really a learning experience for me.

This is essentially a report of a learning experience and I thank all those (many of those who are not listed here) who made my visit so enjoyable. It goes without saying that all mistakes in this report are mine alone. In the next few sections, I will set out details of the things I learned, including, the details and status of the project that i worked on.

Abhishek Surana
IIT Delhi

# Introduction

The modules being developed for the Data Acquisition system for INO VME module utilizing large FPGA's to implement tasks such as data transfer and signal processing. The desire was to be able to download the FPGA configuration information via the VME crate processor and VME bus to the serial EEPROM devices that program the FPGA at power up. The main issue here was in system programming (ISP) of EEPROM devices such that they don't need to be manually programmed every time a new configuration file has to be uploaded. The other issue was universality of configuration scheme such that it can be used with any type of FPGA(Flex10k to cyclone III). Also we wanted to avoid JTAG interface .

By the end of the my stay i was able to design a interface which completed all our requirements but due to time constraints couldn't implement it.

*Table 1–1. Configuration Scheme Device Family Support*

| Configuration Scheme | Device Family | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Stratix IV | Stratix III | Stratix II, Stratix II GX | Stratix, Stratix GX | Arria GX | Cyclone III | Cyclone II | Cyclone | APEX II | APEX 20K, APEX 20KE, | Mercury | ACEX 1K | FLEX 10K, FLEX 10KE, | FLEX 6000 |
| Passive Serial (PS) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Active Serial (AS) | ✓ | ✓ | ✓ | — | ✓ | ✓ | ✓ | ✓ | — | — | — | — | — | — |
| Active Parallel (AP) | — | — | — | — | — | ✓ | — | — | — | — | — | — | — | — |
| Fast Passive Parallel (FPP) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | — | — | ✓ | — | — | — | — | — |
| Passive Parallel Synchronous (PPS) | — | — | — | — | — | — | — | — | — | ✓ | ✓ | ✓ | ✓ | — |
| Passive Parallel Asynchronous (PPA) | — | — | ✓ | ✓ | ✓ | — | — | — | ✓ | ✓ | ✓ | ✓ | ✓ | — |
| Passive Serial Asynchronous (PSA) | — | — | — | — | — | — | — | — | — | — | — | — | — | ✓ |
| Joint Test Action Group (JTAG) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | *(1)* |

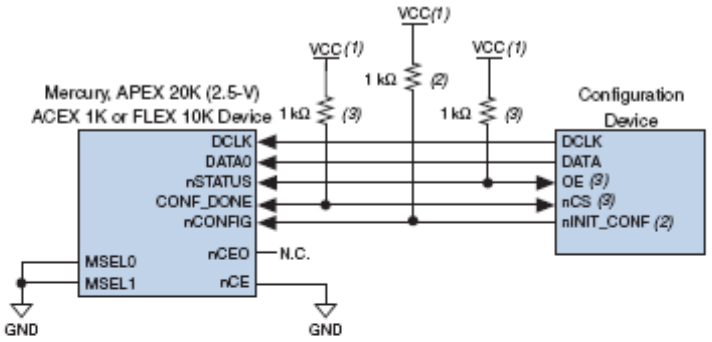
Table: Configuration scheme for different FPGAs.

## Enhanced Configuration Devices

We started off with Flex10k FPGAs as they were available already in the lab and were less expensive. .I zeroed on Passive Serials configuration scheme as it was universal with all FPGAs beside JTAG. In passive serial itself there were different ways to do the configuration . One was to use Enhanced Configuration devices (EPCs or EPCSs) by Altera itself. Another was to use microprocessors with a external flash memory which will configure the FPGA at the time of power up .

For the first few weeks it was thought that EPC1 or EPC1441 could be used with flex10k to see if this system work and later on other enhanced devices with higher memory could be tested with other FPGAs.

Fig: Enhanced configuration device (EPC 1, EPC1441 to configure FPGAs )




(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The nINIT_CONF pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning an external pull-up resistor is not required on the nINIT_CONF/nCONFIG line. The nINIT_CONF pin does not need to be connected if its functionality is not used. If nINIT_CONF is not used or not available (e.g., on EPC1 devices), nCONFIG must be pulled to $V_{CC}$ either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 devices' OE and nCS pins have internal programmable pull-up resistors. If internal pull-up resistors are used, external pull-up resistors should not be used on these pins. The internal pull-up resistors are used by default in the Quartus II software. To turn off the internal pull-up resistors, check the *Disable nCS and OE pull-ups on configuration device* option when generating programming files.

Fig: Enhanced configuration device (EPC 1, EPC1441 to configure FPGAs )

But in between i missed on that these devices were not reprogrammable. Even EPC2 could be reprogrammed only after taking out from circuit manually and programmed again by a programmer. Also, There is some confusion regarding whether these enhanced configuration devices could be used with Cyclone series FPGAs at all as there are some contradictory statements in different documents.

So finally after much discussion it was decided that we would work on use microprocessor and Flash memory to configure FPGA as it was universal. Also it was low on cost side. The idea to use Flex10k was also done away with and now we were working on with Cyclone series devices as it was to be used in final scheme of things.

## Final Design

A CPLD  Xilinx 9572 would be used to put the incoming bit stream from VME (presently from parallel port from PC) to to data flash. This bit stream would consist of configuration file for Altera Cyclone FPGA. A microcontroller would be used to control the transfer of configuration file from dataflash to FPGA.

Single-Device PS Configuration Using an External Host

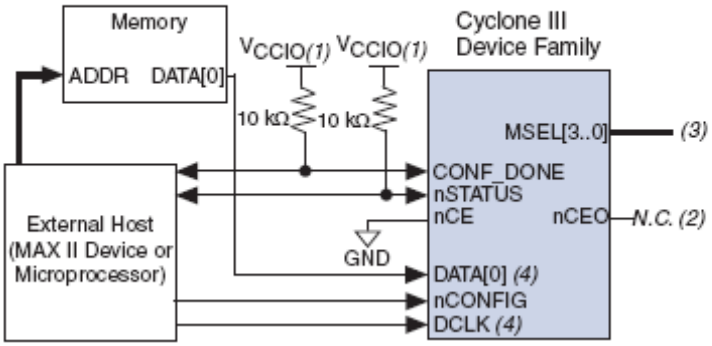
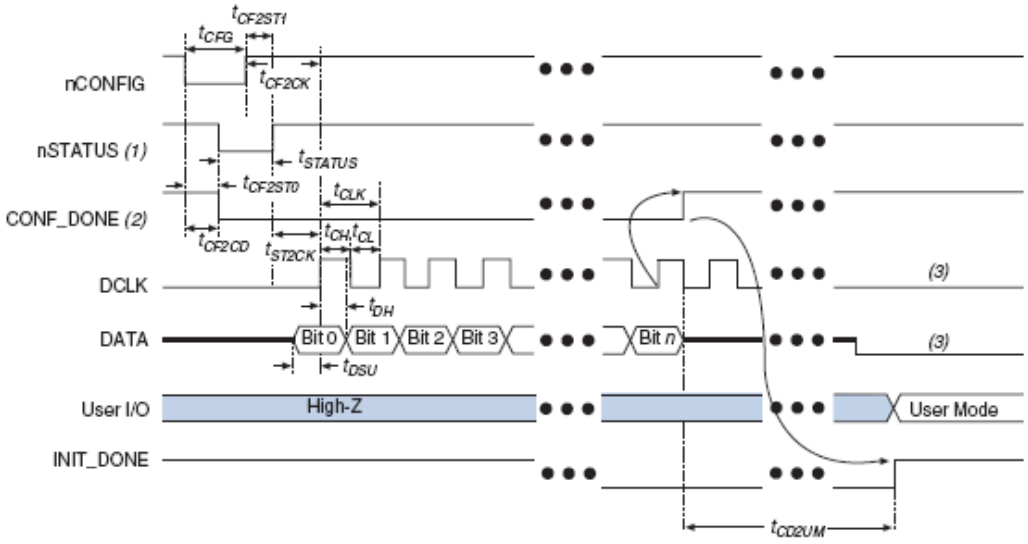



*Figure 8–12. PS Configuration Using a Microprocessor Timing Waveform*



## Dataflash- Atmel AT45DB081D

The data flash we are going to use is Atmel AT45DB081D. Its 8 megabit 2.5 volt with Serial Peripheral Interface compatible modes. Its 8,650,752 bits of memory are organized as 4,096 pages of 256 bytes or264 bytes each. In addition to the main memory, the AT45DB081D also contains two SRAM buffers of 256/264 bytes each. The buffers allow the receiving of data while a page in the main Memory is being reprogrammed, as well as writing a continuous data stream. EEPROM emulation (bit or byte alterability) is easily handled with a selfcontained three step read-modify-write operation. Unlike conventional Flash memories that are accessed randomly with multiple address lines and a parallel interface, the DataFlash uses a RapidS serial interface to sequentially access its data.

To allow for simple in-system reprogrammability, the AT45DB081D does not require high input voltages for programming. The device operates from a single power supply, 2.5V to 3.6V or 2.7V to 3.6V, for both the program and read operations. The AT45DB081D is enabled through the chip select pin (CS) and accessed via a three-wire interface consisting of the Serial Input (SI), Serial Output (SO), and the Serial Clock (SCK).All programming and erase cycles are self-timed.

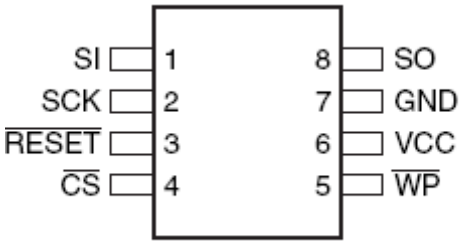



Fig: Pin out of Atmel AT45DB081D Dataflash

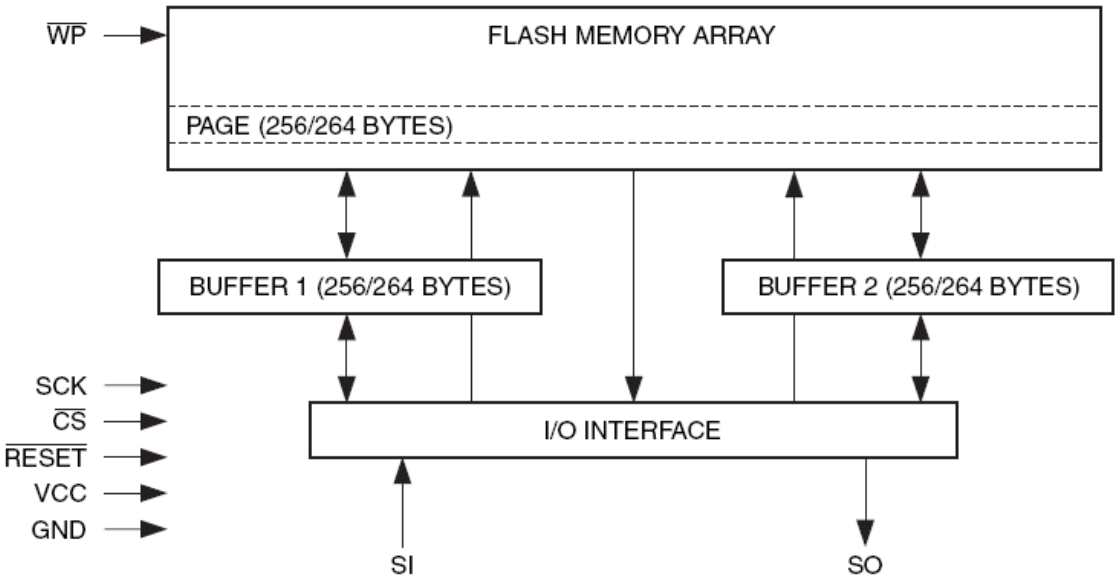



Fig: Memory Structure of dataflash

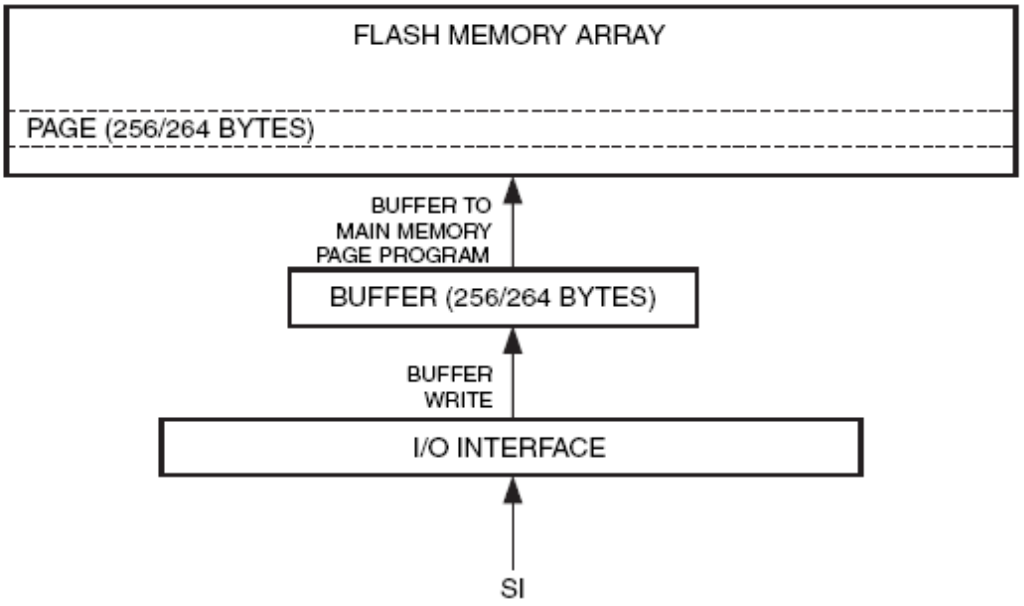



Fig: Write operation in dataflash

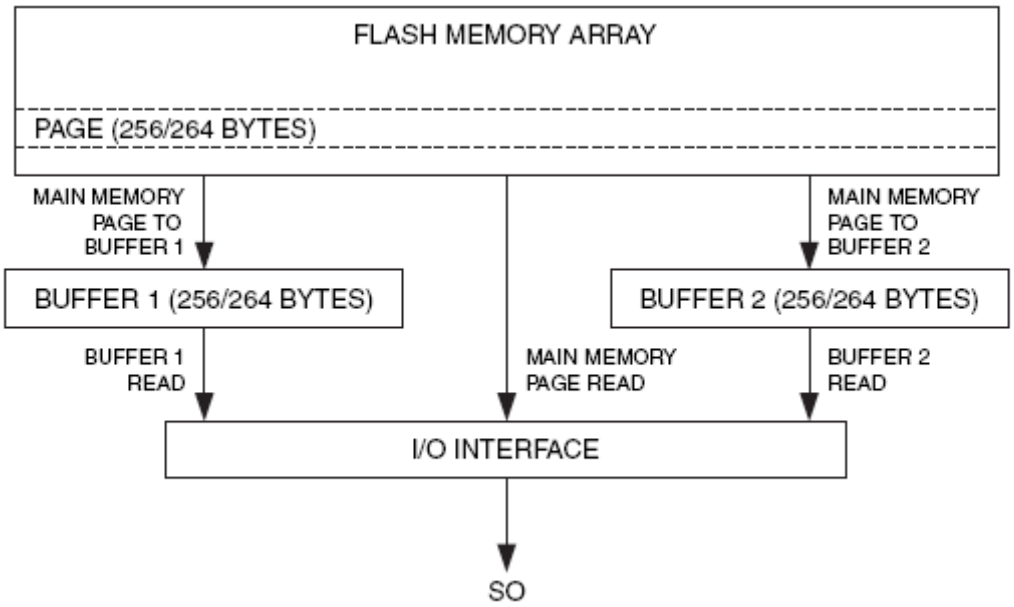



Fig: Read Operation from dataflash

## ATtiny 13

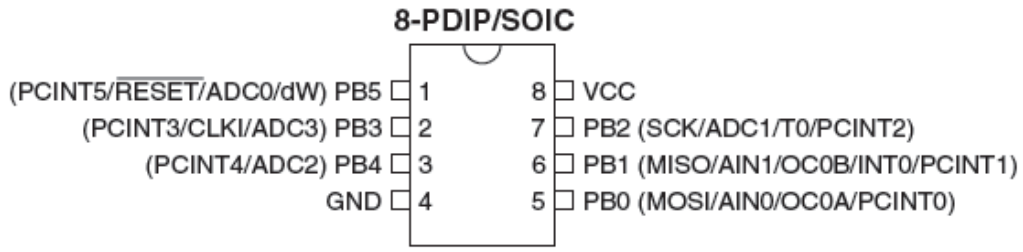



Fig: Pin out for ATtiny13

The ATtiny13 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATtiny13 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed. The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers. The ATtiny13 provides the following features: 1K byte of In-System Programmable Flash, 64 bytes EEPROM, 64 bytes SRAM, 6 general purpose I/O lines, 32 general purpose working registers, one 8-bit Timer/Counter with compare modes, Internal and External Interrupts, a 4-channel, 10-bit ADC, a programmable Watchdog Timer with internal Oscillator, and three software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counter, ADC, Analog Comparator, and Interrupt system to continue functioning. The Power-down mode saves the register contents, disabling all chip functions until the next Interrupt or Hardware Reset. The ADC Noise Reduction mode stops the CPU and all I/O modules except ADC, to minimize switching noise during ADC conversions.

## Xilinx CPLD XC9572

The XC9572 is a high-performance CPLD providing advanced in-system programming and test capabilities for general purpose logic integration. It is comprised of four 36V18 Function Blocks, providing 1,600 usable gates with propagation delays of 7.5 ns.

## Current Status

Presently i was trying to write into dataflash through a parallel port. The C code that i have written would hopefully do that. Initially there were some problems using C for parallel port interfacing in windows vista but later with the use of drivers(inpout32.dll) it was solved. Also I have written the VHDL code for CPLD that would transfer the parallel data to dataflash serially.

The C code and VHDL code are attached in appendix.

## RPC Building

During my stay here i also got a chance to be part of RPC making process. I helped in making RPCs of size 2 × 2 m2 . All glass used in this round was 2 mm glass. The glass was thoroughly cleaned with alcohol (propan-2-ol) on both sides. It was then washed with distilled water The water is thoroughly wiped off at once and the glass is once again cleaned with alcohol. Gloves and large amounts of tissue paper are a must. The glue used was a 3M Scotch-weld epoxy adhesive in a duo-pak cartridge and was applied using the branded 3M EPX applicator.

The edge spacers are designed in sections: a straight piece, and an angle or corner piece. This is because the edges of the glass are cut with a jig so that there is room for the input and output gas tubes. The corner piece has two wedges on either side that slot neatly into holes in the straight sections. It also contains the gas inlet/outlet pipes into which the gas tube fits.

## Future work

I have written C Code for parallel port interface which have to be run and checked. Also the VHDL code that i have written for the CPLD have to be implemented. We were not able to get the microcontroller so once we get that work has to be done to use that microcontroller to transfer data from Atmel Datflash to Altera FPGA. The code also needs to be looked into if it satisfies all our requirements, This interface is in very nascent stage and many things need to be refined such as the amount of time taken to write in dataflash, time taken to configure FPGA etc.

## Appendix

```
/*Program to program dataflash at45d81d using a xilinx 9572 cpld through a
parallel port in Win xp environment*/
// "pport.c" abhishek surana , IIT Delhi

#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<windows.h>

#define BASE 0x378 //Base address is for data port
#define ECR BASE 0x402

// byte port address have to be defined??

#define     D7    0x80
#define     D6    0x40
#define     D5    0x20
#define     D4    0x10
#define     D3    0x08
#define     D2    0x04
#define     D1    0x02
#define     D0    0x01

#define     S7    0x80  /* base address= base+1 */
#define     S6    0x40
#define     S5    0x20
#define     S4    0x10
#define     S3    0x08

#define     C3    0x08  /* base address= base+2 */
#define     C2    0x04
#define     C1    0x02  /* note: NOT(C1) is on the wire */
#define     C0    0x01

// C3 FOR CHIP SELECT CS
// C2 FOR READ/WRITE STROBE
// c2 high- write, low- read


typedef short (_stdcall *inpfuncPtr)(short portaddr);
typedef void (_stdcall *oupfuncPtr)(short portaddr, short datum);

void test_read8(void);
void test_write(void);
void test_write_datum(short datum);

 /* After successful initialization, these 2 variables will contain function
pointers. */
 inpfuncPtr inp32fp;
 oupfuncPtr oup32fp;
 /* Wrapper functions for the function pointers - call these functions to
perform I/O. */
 short Inp32 (short portaddr)
 {
 return (inp32fp)(portaddr);
 }

 void Out32 (short portaddr, short datum)
 {
```

```
 (oup32fp)(portaddr,datum);
 }


 int main(void)
 {

 char filename [300] // for getting filename
 FILE *Filepointer;
 unsigned char *r;
 int statusbits;
 int busystatus;


 HINSTANCE hLib;
 short x;
 int i;

 /* Load the library */
 hLib = LoadLibrary("inpout32.dll");
 if (hLib == NULL)
 {
 fprintf(stderr,"LoadLibrary Failed.\n");
 return -1;
 }

 /* get the address of the function */
 inp32fp = (inpfuncPtr) GetProcAddress(hLib, "Inp32");
 if (inp32fp == NULL)
 { fprintf(stderr,"GetProcAddress for Inp32 Failed.\n");
 return -1;
 }
 oup32fp = (oupfuncPtr) GetProcAddress(hLib, "Out32");
 if (oup32fp == NULL)
 {
 fprintf(stderr,"GetProcAddress for Oup32 Failed.\n");
 return -1;
 }

 ///main program frm here...

 Out32(ECR, 0x01); /*Intialisation to SPP Byte Mode*/
 Out32(BASE+2, 0x24);// Enable Bi directional port and reset =H
 Out32(BASE+2, 0x20);// Enable Bi directional port and reset =l
  Out32(BASE+2, 0x24);// Enable Bi directional port and reset =H

printf("Reading d file");
scanf("%s" , filename);



Out32(BASE+2,0x00);// check for inverted logic
Out32(BASE+2,0x04);//  pulling c2 high for write
Out32(BASE,0x84);//op code for writing in buffer 1
Out32(BASE, 0x00);//dcb
Out32(BASE, 0x00);//dcb
Out32(BASE, 0x01);//address reg

Filepointer=fopen(filename,"rb");
if (Filepointer==NULL)
{
printf("Can not open d file\n\n");
return 1;
}

    fseek(Filepointer,0,SEEK_SET);
```

```
    for (i=k;i<k+256; i++)

    {

     fread(&r,1*i,1,Filepointer);
     fseek(Filepointer,1,SEEK_CUR);

Out32(BASE, &r);

}
fclose(Filepointer);
Out32(BASE+2, 0x08);
Out32(BASE+2,0x00);
Out32(BASE,0xD7);
Out32((BASE+2),0x08);


do
{
statusbits = Inp32(BASE+1);
busystatus=(Ox80 & statusbits);
if (busystatus== 0x80)
{
//give timedelay
}
Out32(BASE+2,0x00);
Out32(BASE, 0x83);// op code for writing in main memory frm buffer 1
//insert loop for address bits...
Out32(BASE, 0x00);//4dc bits+ 12 add bitd+8 dc bits
Out32(BASE, 0x01);
Out32(BASE, 0x00);//dc bits
Out32((BASE+2), 0x08);//cs high


// writing into buffer 2
Out32(BASE+2,0x00);// check for inverted logic
Out32(BASE+2,0x04);//  pulling c2 high for write
Out32(BASE,0x87);//op code for writing in buffer 1
Out32(BASE, 0x00);//dcb
Out32(BASE, 0x00);//dcb
Out32(BASE, 0x01);//address reg

Filepointer=fopen(filename,"rb");
if (Filepointer==NULL)
{
printf("Can not open d file\n\n");
return 1;
}


fseek(Filepointer,0,SEEK_SET);

    for (i=j;i<;j+256 i++)

    {

     fread(&r,1*i,1,Filepointer);
     fseek(Filepointer,1,SEEK_CUR);
       Out32(BASE, &r);

}
fclose(Filepointer);
Out32(BASE+2, 0x08);
Out32(BASE+2,0x00);
Out32(BASE,0xD7);
```

```
Out32((BASE+2),0x08);


do
{
statusbits = Inp32(BASE+1);
busystatus=(Ox80 & statusbits);
if (busystatus== 0x80)
{
//give timedelay
}
Out32(BASE+2,0x00);
Out32(BASE, 0x83);// op code for writing in main memory frm buffer 1
//insert loop for address bits...
Out32(BASE, 0x00);//4dc bits+ 12 add bitd+8 dc bits
Out32(BASE, 0x01);
Out32(BASE, 0x00);//dc bits

Out32((BASE+2), 0x08);//cs high


}
}
```

VHDL Code for Xilinx 9572 CPLD for transferring data from Parallel port to Atmel Dataflash

```
----------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
---------------------------------------
entity cpld_flash_interface is
generic (addr_length: integer:= 24; byte: integer:= 8);
port(
        -- parallel port data, reset and status pins
        ppd: inout std_logic_vector(byte-1 downto 0);     -- data byte from
parallel port
        ppr: in std_logic;                      -- reset o/p from parallel port
        ppc: in std_logic;         -- chip select instruction from parallel port
        pprw: in std_logic;             -- read/write insruction from parallel
port(low for read, high for write)

        --external clock pin for cpld
        cclk: in std_logic;

        -- flash data, address and control pins
        fin: out std_logic;                   -- serial i/p from cpld  to flash
        fout: in std_logic;                   -- serial o/p from flash to cpld
        fclk: out std_logic;                         --  clk to flash from cpld
        frst: buffer std_logic;                      -- reset to flash  from cpld
        fcs: out std_logic                       -- chip select to flash from
cpld

         );
end cpld_flash_interface;
----------------------------------------


architecture design of cpld_flash_interface is
constant HI: std_logic:= '1';
constant LO: std_logic:= '0';

begin

fcs<= ppc ;             --parallel port send instruction for chip select
frst <= ppr;                    --remove flash reset for writing
fclk <=cclk;                    --external clock input from vlsi kit (3.57
MHz)


process(cclk)

if(pprw =LO )then       -- pprw is low meaning writing operation should be done.

variable inp: std_logic_vector(byte-1 downto 0);


--    if (pprw=LO) then       --(pprw is for read and write)
           inp:= ppd;
           if(cclk'event and cclk='1') then --
                 fin<= inp(byte-1);
                 inp(byte-1 downto 1):= inp(byte-2 downto 0);
                                 -- shift register
           end if;
--    end if;
--end process;
else
```

```
variable shift_data: std_logic_vector(7 downto 0);                          --
reading operation in case pprw is high

      for i in 0 to 7 loop
            if(cclk'event and cclk='1') then
                  shift_data(i):= fout;
                  --shift_data(2 downto 0):= shift_data(3 downto 1);
--                shift_data(0):= fout;
            end if;
      end loop;
      ppd<= shift_data;
end process;


end design;
```