Syscore Board v. 1.0 Flash Interface

David Rohr

21.3.2008

This document refers to Syscore Board v. 1.0 with a Virtex4 FPGA, an Actel ProAsic3 CPLD and 16 MB of onboard flash RAM. It contains a description of the software and VHDL design that can be used to store and retrieve Virtex bitfiles in the flash RAM. As a second functionality the Actel design will be able to check the Virtex status and reprogram the Virtex if needed. Two different bitfiles will be stored in the flash RAM chips. The user will either be able to select the file to program by jumper or there will be a full bitfile for initial programming and then only a partial bitfile for continous refresh in the second flash chip.

Contents

1 Tools

1.1 PC Components

The component that handles the communication for the pc side is a simple c++ program for windows written in Visual Studio 6. Most OS dependant function have been collected in os.cpp, so it should be easily portable to other operating system by accomodating this file.

1.2 Virtex4 Components

Tools used to create the Virtex4 components are Xilinx ISE and Xilinx EDK version 8.2i service pack 2. The Virtex4 was programmed using standard tools.

1.3 Actel Components

Actel Libero IDE 8.1 was used for creating and compiling the Actel deesign. Though flashing the design to the actel was way more complicated than expected. Flashing the Actel chip with FlashPro 6.1 or FlashPro 6.2 software fails with an error, saying the flash programmer "FlashPro Lite" was not compatible with the ProAsic3 chip family on the Syscore board. A research on the Actel web site came to exactly the same conclusion. The "FlashPro3" Programmer should be used for this chip family. Contrary to the "FlashPro Lite" the "FlashPro3" would also have an USB interface instead of the parallel interface, that could be used easier on recent computers.

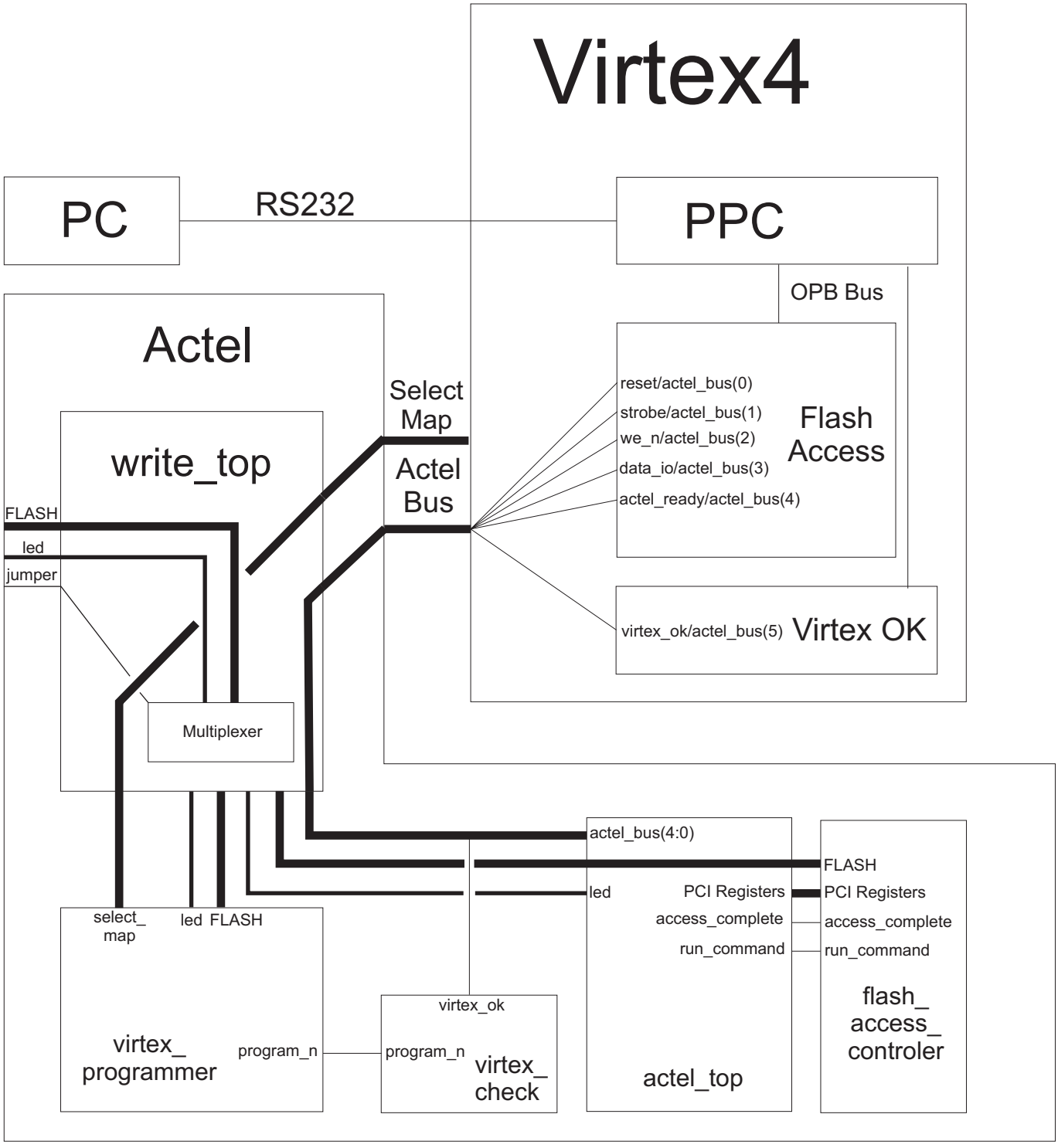
However it is a fact that the FlashPro Lite has been used to program the Syscore board before on another computer. A comparison of the software versions revealed that FlashPro 6.0 was installed there. A first attempt with FlashPro 6.0 failed again. But no longer complaining about incompatibility it stated that the ident code was unknown. It then turned out that FlashPro 6.0 could program the ProAsic3 chip using FlashPro Lite programmer when opening the project file that was used before instead of creating a new one.

For this reason a copy of FlashPro 6.0 and the FlashPro project file are included in the directory "save/actel_flash_pro". No installer needs to be run. The FlashPro 6.0 can just be copied to hard disc and by using the supplied project file programming the chip using the FlashPro Lite is possible.

Another problem with the FlashPro Lite is it's parallel interface. The programmer will only work with an ECP capable parallel port. Further it needs the 5V voltage supplied by the parallel port. So attempts with an USB to parallel converter failed. Therefore the programming was done with a second laptop, old enough to still have a parallel port.

2 Design

2.1 Component Diagram



2.2 Component Description

The Virtex design is stored in Linux9 folder, the Actel design in test_write_neu folder. A set of working bitfiles and a c++ binary is stored in the working design folder.

2.2.1 flash_access

This component is required to read/write data from/to flash RAM using a PC. The following addresses are available relative to OPB bus base address.

flash_access OPB bus addresses

IDCODE_ADDR	000	RO	Can read OPB IDCode constant X"41465031" (AFP1)
CHECK_IDLE_ADDR	001	RO	Returns 1 if both, Actel and Virtex state machines are idle
RESET_ADDR	010	WO	Write any data to reset Actel and Virtex statemachines
GEN_WRITE_ADDR	011	WO	Write 8 bits (15:8) to 8 bit address (7:0)
DATA_READ_ADDR	100	RW	Write any data to copy Actel read out register to Virtex read out register. Read to read out Virtex read out register. Bit 31 will then contain actel_ready signal
DATA_WRITE_ADDR	101	WO	Write 8 bit to Actel flash_access_controller PCI data register
ADDRESS_REG_ADDR	110	WO	Write 24 bit to Actel flash_access_controller PCI address register
COMMAND_REG_ADDR	111	WO	Write 4 bit to Actel flash_access_controller PCI command register. Writing to this register will start the PCI command on the Actel side

(Given are address bits 27 to 29, bits 30/31 are tied to zero because of DWORD addressing, remaining bits are ignored)

Flash access from the PPC side is completely transparent and only handled by access to these addresses. Though committing PCI write commands will write to the flash chip and not to direct flash RAM addresses. It is necessary to send commands to the flash memory controller to actually store data in the flash RAM as described later. The PC program though will handle this of course.

In the following the communication between flash_access and the Actel CPLD will be described in detail. All write commands (to PCI data, PCI address or PCI command register addresses) will internally be translated into gen_write_commands. Each gen_write_command will write 8 bit to the corresponding register in the Actel chip addressed by the 8 bit gen_write_address. So 16 bits need to be transmitted. Those bits are transferred bit by bit through the one bit serial data_io/actel_bus(3) pin. The 24 bit address is split into three 8 bit registers and the access to the OPB address_register address will transparently be split into three gen_write_commands. A gen_write cycle will hold the first bit on the io pin, then hold strobe high for 8 clock cycles (because of different clock speeds), then hold strobe low for 8 cycles, then shift the bits put on the io pin and repeat the procedure until the entire 16 bits have been transmitted. When not in gen_write state the io pin is tristated.

The read process is similar though there is no read address. The Actel provides an 8 bit read register that is read bit by bit every time we_n falls to low. The Actel has an internal counter so that after 8 read cycles the read register is refreshed from the flash_interface read register. So when a new connection to the actel is established a reset command should be executed to reset this counter. Again: The read process itself is started when writing to the DATA_READ_ADDR what will copy the read out register from Actel to Virtex. To actually read the data an OPB read command must then be run on DATA_READ_ADDR.

The flash_access entity consists of two state machines. One for handling OPB commands and another for the Actel communication. OPB read commands can always be initiated and will read the

appropriate data. (Data read from DATA_READ_ADDR are internally stored in a register that can always be read). Write commands however will only succeed when the second state machine is in idle state (This can be checked with CHECK_IDLE_ADDR). Then the VHDL signal WRITE_TO_ACTEL is set 1 for twp cycles for the Actel communication state machine to copy the OPB Data bus to internal registers.

2.2.2 virtex_ok

virtex_ok is a simple component that will output a slow clock that can be controlled by the Actel chip. When the clock stops running the Actel should reprogram the Virtex. The component will accept OPB Writes to it's base address and then enable or disable the clock depending on the OPB Data.

2.2.3 write_top

This is the top component in the Actel design. It's main purpose is to multiplex data between virtex_programmer and actel_top component. The operation mode is determined by the jumper settings. With both jumpers open the actel chip will be hold in reset mode. With the reset jumper (CON2) unconnected but the con1 jumper closed the actel_top component will be enabled so data can be read from and written to flash. With the reset jumper (CON2) closed the virtex_programmer component is active and CON1 jumper is passed to it. The write_top component also acts as tristate buffer for some signals and sets some default values for flash_reset, usb_reset etc.

2.2.4 virtex_check

This will just check if the slow control clock coming from the Virtex runs and then sets program_n correspondingly. With program_n low the virtex_programmer entity should start reprogramming the Virtex.

2.2.5 virtex_programmer

The component will evaluate the virtex_ok signal from the virtex_check component. To program the Virtex at first prog_b is pulled low to do a full chip reset on the Virtex (S_Prog_b_0). It will then wait for the init_b pin to raise high (S_Prog_b_1). Now the Virtex is ready for programming. Before programming the binfile's size stored in flash is read during 3 states (S_Size_1 to S_Size_3). The jumper con1 decides which flash chip is used. Afterwards CCLK is clocked once (S_Sync_1, S_Sync_2) , then cs_b set low. Then the binfile is clocked out with the CCLK to the Virtex (S_Fetch_Data, S_CCLK_1, S_CCLK_2). The process should be transparent to the user as the binfile already contains the synchronize and the start sequence for the Virtex. For this to operate correctly the startup clock must be set to CCLK and the done pin must be driven high during startup. The corresponding options need to be enabled in Bitgen. After the programming has finished the Actel will wait some time (S_Reset) and then return to it's observation mode (S_Idle) and check the Virtex clock again. If an error occurs during programming (init_b does not become high, a busy signal is received or the done signal does not go high) the Actel will enter its error state indicated by LED1 blinking. This is important for debug purposes right now. Alternatively this could be changed into reprogramming the Virtex again after an error occurred when installed in an experiment later. There is an alternative to entering observation mode after programming. If the par_refresh_partial_binfile parameter is set 1 the actel will permanently program the Virtex again out of the second flash chip. (The initial programming will use the first flash chip and CON1 jumper is ignored). This is used in conjunction with a partial binfile stored in the second flash for example to keep the Virtex programming alive even if the Virtex is exposed to radiation which otherwise would destroy the programming and leave the Virtex inoperable. The bit order of the flash data passed to the select_map pins is reversed because the Virtex, when in 8 bit select map mode, requires reversed bit order.

2.2.6 actel_top

actel_top will handle the communication between the Virtex and flash_access_controller. The incoming signals strobe and we_n are converted into we_n_shift and strobe_previous that indicate a rising/falling edge of the signal. When we_n_shift is pulled the output register is at first copied from the flash_access_controller output register (controller_reg_data_read) and then shifted 7 times. While we_n is low the corresponding bit is laid on the data_io pin. When we_n is high the pin is tristated. When writing to the Actel always 16 bits are received. Those are split into data and address bits and made available through gen_write_addr_i and gen_write_data_i wires. After address and data have been completely received they must be evaluated within one clock cycle. The following addresses are used right now:

Actel Register Addresses

REG_DATA_A	8'h01	
REG_CMD_A	8'h02	
REG_ADDR_0_A	8'h03	write only, bits 7:0 of the address register
REG_ADDR_1_A	8'h04	write only, bits 15:8 of the address register
REG_ADDR_2_A	8'h05	write only, bits 23:16 of the address register

When writing data to the command register run_command is set 1 the next clock cycle so flash_access_controller will run the PCI command. The component will then wait for controller_access_complete from the flash_access_controller and then reset the command register to 4'b0001. Then the actel_ready signal goes ready again and a new command can be committed.

The bit order is reversed in this component to account for OPB bus bit order.

2.2.7 flash_access_controller

This component has PCI registers (data, address, command) as input. When the state machine is idle and run_command is high the PCI registers' data is executed as PCI command. All read commands will write the data read from flash to flash_read_data_reg and pass this to actel_top. Write commands again do not result in direct writes to the flash RAM but just write to the flash memory controller. Commands that can be send to the controller are listed below when describing the c program.

3 C programs and RS232 Interface

3.1 Interface

A PC and a PPC C-program will communicate through an RS232 Interface. The interface runs up to a baud rate of 57600 baud. Higher baud rates might result in errors. Optimization here naturally is possible. The PPC program uses the XUART commands while the PC program uses standard Windows API (CreateFile / ReadFile / WriteFile).

3.2 PPC Program

On the PowerPC a C-program runs that communicates with a computer through the RS232 interface and with flash_access and virtex_ok through the OPB bus. OPB bus base addresses are store in integer volatile base_addr pointers. The integer pointers ties bits 30/31 of the OPB address to 0. Volatile is required so that the compiler will not cache data written to the OPB bus as it would for memory. In any state a reset command must be receivable. For this reason 100 consecutive reset commands ('a') will always reset the PPC interface even when reading data that should be written to flash. So the flow control has to make sure no 100 consecutive reset commands can occure in a data stream. For this reason control bytes (255) must be send in a smaller interval. Those control bytes then also are used for flow control. The PPC Program permanently waits for incoming RS232

data and has a state machine that handles incoming data. Most common commands should be burst read and burst write that read/write a specified amount of data from/to a flash address to/from the RS232 interface. Other commands are rather for debugging purpose. Available commands are listed below in PC program description.

3.3 PC Program

This is a (currently windows) c++ program that can access the flash memory. There are some os dependant functions in os.cpp. To adept the program to Linux those should be adapted. Then there is some more Windows specific code in rs232.cpp (like QueryPerformanceCounter) to measure time. Those need to be changed too or just be thrown away. When started the program initializes the com port and issues a reset command. Then it awaits commands from the user. The following commands are available.:

3.3.1 Commit Chip Erase [e]

As one can only write a 0 to Flash RAM but no 1 the Flash must be erased and every byte set to 0xFF before it can be written. This command can erase a single or both FlashRAM chips. To commit a chip erase the folowing sequence is written to the flash ship (in fact to it's state machine)

Address	Data
0xAAA	0xAA
0x555	0x55
0xAAA	0x80
0xAAA	0xAA
0x555	0x55
0xAAA	0x10

To select between addressing the two flash chips the highest address bit (24 bit address) is set 0 or 1. Those commands are commited by c-function CommitChipErase. The program then calls function CheckChipErase that waits until the chip erase has completed. There are sequences being able to delete small sections of the flash chip. Those could be implemented later if needed. But as we only want to store a single binfile in each flash RAM this is not necessary yet.

3.3.2 Ident Device [n]

Makes the PPC ident itself and print the IDCode address from the OPB bus.

3.3.3 Reset Interface [r]

Will reset the interface by sending reset commands (up to 100) until "Resetting" (ASCII string) is received. If no such answer is received after 100 tries the PPC program is not working correctly and the program will terminate. Having reset the PPC successfully the PC will issue a reset command to the OPB interface and this will then reset the flash_access component in the Virtex and the Actel chip. Finally an ident device command is send to the PPC.

3.3.4 Ping [p]

Ping the PPC and wait for pong to check if connection is alive.

3.3.5 Low Level Write [o]

This will call the WriteActel function to write to OPB addresses as described in the flash_access component description.(might be confusing as there are also Actel addresses but this will write to OPB addresses). In combination with the GEN_WRITE_ADDR Actel addresses can be written. So for example to write a PCI read command to the Actel PCI command register there are two possibilities:

- Write 0x04 to 0x07 (COMMAND_REG_ADDR)
- Write 0x0402 to 0x03 (GEN_WRITE_ADDR / 0x0402 is splitted into address 02 (REG_CMD_A) and data 04 (PCI_COMMAND_READ)

3.3.6 Low Level Read [k]

Similar to Low Level Write this will read from an OPB address.

3.3.7 Write Single Byte [w]

This will write a single byte to a flash address. Inputa are a 24 bit address and an 8 bit data value. The WriteRAM C-function is called. It will call the WriteFlash function to issue a write sequence to the flash:

Address	**Data**
0xAAA	0xAA
0x555	0x55
0xAAA	0xA0
Address	Data

(Address and data must of course be replaced by the appropriate values.)
The WriteFlash function itself will use the WriteActel function to fill the Actel PCI registers to issue a write command. *(As Low Level Read/Write).* This way it will write the data to 0x05 (DATA_WRITE_ADDR), the address to 0x06 (ADDRESS_REG_ADDR) and then 0x02 to 0x07 (COMMAND_REG_ADDR):

3.3.8 View Single Byte [v]

Will use the ReadFlash C-function to read one byte from a particular flash address and display this. There is no need to talk with the flash state machine for reading so there are no different C-functions ReadFlash / ReadRAM but only ReadFlash as compared to WriteFlash/WriteRAM. The function will then:

- Write the address to OPB address 0x06 (ADDRESS_REG_ADDR)
- Write 0x04 to OPB address 0x07 (PCI_COMMAND_READ: so the data is read from flash to internal register in flash_access_controller)
- Write 0x00 to OPB address 0x04 (DATA_READ_ADDR: so data is read from the Actel to Virtex)
- Read from OPB address 0x04 (DATA_READ_ADDR: data is read to PPC and send through RS232)

3.3.9 Upload File [u]

This command will upload a file to the flash RAM. It will not erase the chip before. The program will ask for the filename and the flash address to write the file to. It then uses the WriteRAMBurst function to perform a burst write to the flash. The WriteRAMBurst function sends the WriteRAMBurst

command, then 32 address bits followed by a 32 bit size. Then the data bits are sent. There must be control bytes set to 255 in a defined interval (see rs232.h) so that a reset sequence cannot be sent unintentionally. For flow control reason the PPC will answer every control byte with a ".". Pressing a key during transfer will abort the BurstWrite. A simple XOR checksum is calculated during transfer and compared at the end to check for errors. This could be advanced to a CRC later on.

3.3.10 Download File [d]

Will ask for Filename, Flash Address and Size. Then downloads [Size] bytes from [Flash Address] and save those to [Filename]. For this it uses the ReadRAMBurst command that resembles the WriteRAMBurst.

3.3.11 Copy from PPC Address [a]

Because File upload through RS232 console is quite slow (about 4 kb/s at 57600 baud), a second option is hereby supplied to write data to the flash RAM. With this command, data can copied from a PPC address to a flash address. So for example you could use XMD to write a file to the onboard DDR RAM and then use this command to copy from the DDR RAM to flash. (This results in a higher speed of about 10 kb/s at the moment). The command needs 3 Inputs, PPC address, flash address and amount of bytes to copy.

3.3.12 Extract Binfile from Bitfile [x]

Bitgen can output bin and bitfiles, where bitfiles are just binfiles supplied with a header. For programming binfiles are needed so this option can remove the bitfile header and extract a binfile out of a bitfile. If given a binfile as input it will just be copied. Other input files should be recognized as invalid. This command will write the binfile to hard disc and append ".bin" to the filename.

3.3.13 Upload Bin/Bitfile [b]

This will upload a given bit/binfile to either chip 0 or 1. It will perform the following tasks:

- Extract a binfile out of the bitfile if neccessary
- Provide the binfile with a 512 bit header, that contains the filesize and some extra info. *(This filesize in the header is then used by the virtex_programmer component of the Actel design to obtain the number of bytes to send to the virtex.)*
- Commit a chip erase on the chip selected
- Use WriteRAMBurst to upload the binfile with header to the selected chip

(Look at rs232.h to see which other values are stored in the header!)

3.3.14 Show Bitfile Information from Flash [i]

Will show the information contained in the binfile headers stored in both flash rams or an "invalid binfile" error if no header is found.

3.3.15 Download Binfile [l]

Will display the binfile header info and then allow to download one of the binfiles from one flash chip and store it in a file.

3.3.16 Test Sequence [t]

The PPC C-programm contains a TestSequence function that is used for debugging purpose. This command will call the function and print it's output.

3.3.17 Change Debug Mode [m]

Changes the amount of debugging info displayed in a range of 0 to 5.

3.3.18 Print Help [h]

Prints a list of available commands.

3.3.19 Quit [q]

Exits C++ program on PC side.

3.3.20 Set Virtex_ok_disable state to 1 [z]

This will stop the virtex_ok entity in the Virtex from sending the virtex_ok_clock to the Actel. The Actel (when in observation mode) should then start reprogramming the Virtex.