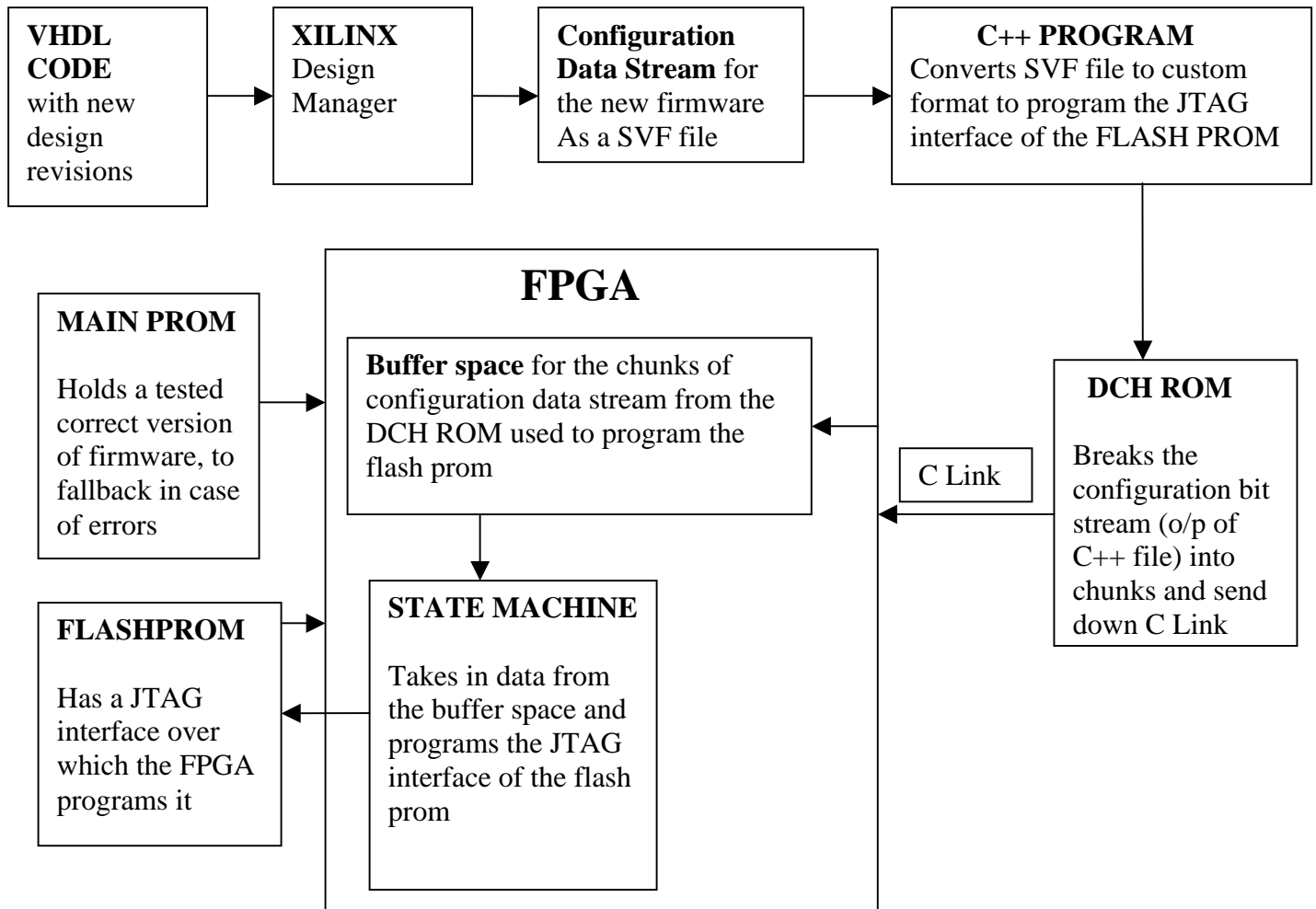


FIRMWARE DOWNLOAD AND ON-BOARD FLASH PROM PROGRAMMING

Overview:

The proposed system is to make possible, the reprogramming of the configuration PROM on the FEA “On-board”, so that it is not required to manually replace the configuration PROM every time there is a design revision and there is the comfort of standby prom to test the design revisions. The system works as follows:



System Level Description:

- The VHDL code for the new firmware is first compiled on a Xilinx Design Manager which gives a SVF (Serial Vector Format) File, which is an Assembly Language Equivalent of the actual Binary File used to program the PROM. The SVF is a standardized format to interact with a JTAG interface and contains a list of commands to work the TAP controller (state machine) inside the JTAG interface controller.
- The exact way Xilinx Programs a FLASH prom using parallel jtag cables is proprietary. Hence the procedure adopted here is to read the SVF file generated by Xilinx and convert the file to a convenient protocol, which can then be used to control the Jtag interface controller. For this purpose a C++ file has been written to read in the SVF file and convert it to a binary file of the desired protocol
- When it is desired to program the Flash Prom, the DCH ROM sends down a command down the C-Link, which triggers the prom-program state machine in the FPGA. The DCH ROM then send a chunk of the configuration bit stream such that the chunk has an integer number of SVF commands, to the FPGA. The reason to break the configuration stream is that, the FPGA does not have enough memory to store the whole data stream.
- The “prom-program” state machine in the FPGA on receipt of this data stream, generates the appropriate signals to send down the 4 control pins on the JTAG and it does so at a reduced frequency of 7.5Mhz to program the PROM
- The DCH ROM all this while keeps polling a status register through the D-Link, to check if the state machine has finished the programming, and if it has finished it sends down the next chunk of configuration data stream.
- The “prom-program” state machine maps every command to a set of control signals on the 4 pins of the JTAG to cause one of the 16 allowed state transitions on the JTAG.
- The status register also maintains a list of errors, which track the errors of transmission down the C-link, and errors on write and read back of the same memory location
- Between two chunks of data stream, the clock to the JTAG is held low. There is no loss of functionality in doing so, as the JTAG is only sensitive to the rising edge of the clock and nothing else.

Technical Details

Following are the list of commands in the protocol.

SDR: Equivalent to the SDR command of SVF format without any check on the out coming bits on the TDO line

Format: (0100 xxxx) (xxxxxxxx...) (xxxxxxxx xxxxxxxx)

- In the first part the first 4 bits signify the type of command, the next 4 bits specify The end stable state for that command.
- The 2nd part specifies the size of payload following it. It can be max of 32 bits
- The 3rd part is the actual payload of size given by the 2nd part

SDRMASK: Equivalent to the SDR command of SVF format with check on the out coming bits on the TDO line with a given mask

Format: (0110 xxxx) (xxxxxxxx...) (xxxxxxxx....) (xxxxxxxx ...) (xxxxxxxx)

- In the first part the first 4 bits signify the type of command, the next 4 bits specify The end stable state for that command.
- The 2nd part specifies the size of payload following it. It can be max of 32 bits
- The 3rd part is the actual payload of size given by the 2nd part
- The 4th part is the check bits of tdo line of size given by 2nd part
- The 5th part is the mask bits to signify which bits to check on tdo of size given by 2nd part

SDRMASK1: Equivalent to the SDR command of SVF format with check on the out coming bits on the TDO line assuming a mask of all '1's. This command is used to optimize the fact that 99% of the time the mask is all '1's

Format: (0010 xxxx) (xxxxxxxx...) (xxxxxxxx....) (xxxxxxxx ...)

- In the first part the first 4 bits signify the type of command, the next 4 bits specify The end stable state for that command.
- The 2nd part specifies the size of payload following it. It can be max of 32 bits
- The 3rd part is the actual payload of size given by the 2nd part
- The 4th part is the check bits of tdo line of size given by 2nd part

Similarly the corresponding commands for the SIR command in SVF are

SIR: Equivalent to the SDR command of SVF format without any check on the out coming bits on the TDO line

Format: (0011 xxxx) (xxxxxxxx...) (xxxxxxxx xxxxxxxx)

SIRMASK: Equivalent to the SDR command of SVF format with check on the out coming bits on the TDO line with a given mask

Format: (0101 xxxx) (xxxxxxxx...) (xxxxxxxx....) (xxxxxxxx ...) (xxxxxxxx)

SIRMASK1: Equivalent to the SDR command of SVF format with check on the out coming bits on the TDO line assuming a mask of all '1's. This command is used to optimize the fact that 99% of the time the mask is all '1's

Format: (0001 xxxx) (xxxxxxxx...) (xxxxxxxx....) (xxxxxxxx ...)

STATE: Equivalent to the STATE cmd in the SVF but has only one stable state as payload. Any other complex STATE cmd can be broken into the above simple STATE cmd.

Format: (1111) (one of 4 valid stable states rep. In 4 bits)

RUNTEST: Equivalent to the RUNTEST cmd but specifies only in terms of TCK

Format: (1011) (one of 4 valid stable states rep. In 4 bits) (payload of max of 32 bits)

OPTIMIZATIONS:

- To optimize the fact that for the payload size, less than 32 bits are enough, the minimum number of bits required as calculated by the C++ program are passed as parameters by the data stream to the “prom-program” state machine in FPGA
- ENDDR, ENDIR Cmd in SVF format are integrated along with the end states of the SDR/SIR Cmd
- TIR, TDR, HIR, HDR Cmds of the SVF format are not supported since we do not program proms, which are connected in chains. It is always a single PROM
- The payload is sent as payload size-1 to use mostly 8 bits. The most common payload 256 can be represented in 8 bits if sent down as 255

OTHER TIDBITS OF INFORMATION ABOUT THE SYSTEM

- Error signal flagged to the status register
 - Bit 0 says type of error
 - Bit 1 says whether if there is an error or not
 - '1' - if received tdo not same as expected tdo
 - '0' - if received fifo data does not match any known svf command due to transmission mistake or some other mistake which permanently makes the state machine lose track of fifo. This does not mean that the state machine lost track at the instant error happened. It's only true that somehow it locked an error now.
- There will also be abnormal execution when the fifo gets empty before the complete payload is over. This cannot be trivially observed.
- After an error the state machine automatically resets itself so there is no need for any external reset to program it again.
- The jtag machine cannot go to the reset state between two chunks, which will cause the programming to fail. Hence it is not advised for an external master to take control of the pin between chunks during programming.
- The Xilinx design manager generates the svf file assuming a 1Mhz frequency hence the runtest duration has to be multiplied for 7.5Mhz at which the system is running. This is handled by the C++ program, which converts the svf file.
- The last line in the svf file send a 1 bit sdr which might be to verify prom, which is not a necessary command and has been omitted since the c++ file assumes a minimum 8-bit payload.
- After the verify stage a couple of svf commands are executed which are critical to the jtag, completing the programming operation and transferring pin control and hence is essential even though it might look redundant.
- When the c++ file produces an output file from the svf file most of the times a carriage return is inserted midway in the file, which causes a failure. A cursory glance through the file is good enough to locate this.
- Since there is a pull up on the jtag output pins, the jtag pins could not be trivially put in high impedance state, which causes a spurious clock edge. This should be kept in mind when modifying the code.

- The maximum rated speed for the programming system is limited by the speed of the jtag pin speed which is currently 10Mhz.
- There is a lot of right to left data transmission and byte swapping which should be properly accounted for when debugging which is all clearly documented in the c++ file.
- The present system does not exactly mimic the way Xilinx programs the chip over the jtag cable. Hence a clock by clock analysis with the Xilinx programming will fail
- A number of assumptions regarding the payload size, svf file format have been made in the C++ file (documented in the c++ file), hence it is not a completely generic system but has been designed to be easily portable.