XVPI: A Portable Hardware / Software Interface for Virtex

Prasanna Sundararajan and Steven A. Guccione

Xilinx Inc.
2100 Logic Drive, San Jose, CA 95124 (USA)

ABSTRACT

XVPI, the *Xilinx Virtex Portable Interface*, is a hardware / software interface and specification to assist in the design and implementation of Xilinx Virtex$^{(tm)}$ based systems. *XVPI* specifies a software accessible register to be defined in the hardware. This register contains all of the control and data signals necessary to drive the Virtex device. The software supplied with *XVPI* uses this register to read and write control and data signals to perform various device level functions. These functions combine to produce an Application Program Interface (API) which provides access to the Virtex device from software. The *XVPI* API supports all of the device level operations including partial configuration download, partial configuration readback, clock control and reset. Once the system is operational, designers may replace the software routines in the *XVPI* API with hardware assisted routines. This increases the system performance incrementally, without affecting the functionality. Though specified for the Virtex based system, this technique to perform the device level functions from software can be applied to any FPGA device. Additionally, *XVPI* is also supplied with an interface supporting Xilinx's *JBits* toolkit. Once *XVPI* is implemented, *JBits* and its associated applications, including the *BoardScope* debug tool, are fully operational on that Virtex based system.

Keywords: Reconfigurable systems, FPGA, Hardware / software co-design.

1. INTRODUCTION

In the past, supplying a configuration data to an FPGAs device has been relatively simple. The configuration data, or *bitstream* was typically stored in some form of serial, non-volatile memory. This data was loaded into the FPGA at start-up with little or no additional hardware or software support. While this approach is still available, it is a very low performance alternative and ignores many of the powerful features of newer devices such as the Xilinx Virtex$^{(tm)}$ series.

While older families of Xilinx FPGA devices offered bit-serial configuration interfaces operating at approximately one MHz, The Xilinx Virtex series of FPGA devices uses an eight bit parallel port capable of speeds of as much as 80 MHz. This interface, known as the *SelectMap*$^{(tm)}$ port, provides over five hundred times the bandwidth of the older families of devices. And in addition to simple configuration download, the *SelectMap* port also provides support for partial reconfiguration, readback, partial readback and several control functions including reset[5].[1]

While the *SelectMap* interface provides many useful capabilities for FPGA design, debug and run-time functionality, it is substantially more complex than older interfaces. This new level of functionality requires more complex hardware interfacing and additional software to make use of these features. This has placed an additional burden on the hardware designer, adding new and somewhat unfamiliar interfacing tasks to FPGA designs. Additionally, because many of these features are used in modes such as debug, they are often ignored by hardware designers.

With this in mind *XVPI*, the *Xilinx Virtex Portable Interface*, provides a complete set of hardware design specifications and software which should permit designers to immediately take advantage of not only the basic functionality of the new Virtex *SelectMap* interface, but also of existing software such as *JBits* and its associated design and debug tools which make use of this interface.

Further author information -
P.S. email: Prasanna.Sundararajan@xilinx.com
S.A.G. email: Steven.Guccione@xilinx.com

2. THE *XVPI* HARDWARE INTERFACE

Because this approach is a combined hardware / software solution, it is assumed that some microprocessor is interfaced to the *SelectMap* port of the Virtex device. The microprocessor will initially provide all of the control and data for the Virtex device, and maintains the responsibility for reading and writing data to the *SelectMap* port over some bus.

The Virtex family of devices reserves a set of IO pins which provide the interface to the *SelectMap* port. In *XVPI*, this interface is attached to some register called the *Virtex Interface Register* or *VIR* which is accessible by the microprocessor. The only functionality required by the system is a readable and writable *VIR* register which accesses the *SelectMap* port. Table 1 describes the *VIR* used by *XVPI*. Note that all signals used by the *SelectMap* interface are connected to this register. This provides the simplest possible hardware interface to quickly provide operational hardware.

Bits	Name	Direction	Description
0-7	DATA	In/Out	Bi-directional Databus. Data is written during configuration and read during readback
8	CS	Out	Chip Select input enables SelectMAP data bus. To write or read data onto or from the data bus, CS signal must be asserted low.
9	PROG	Out	Asynchronous reset to configuration logic
10	INIT	In/Out	Indicates Initialization progress and configuration error, if any.
11	M0	Out	Mode Bit 0
12	M1	Out	Mode Bit 1
13	M2	Out	Mode Bit 2
14	CCLK	Out	Configuration Clock that synchronizes the loading and reading of the data during configuration and readback.
15	RW	Out	SelectMap Read/Write Signal. When asserted low, the RW signal indicates data being written to the data bus. When asserted high, RW indicates data is read from the data bus.
16	BUSY	In	SelectMap Port Busy Signal. When CS is asserted, BUSY output indicates when the Virtex can accept another byte. If BUSY is low, Virtex reads the data bus on the next rising CCLK edge that both CS and RW are asserted Low. If BUSY is high, the current byte is Ignored and must be reloaded during next CCLK rising edge when BUSY is low. When CS not asserted BUSY is tri-stated and asserted high.
17	DONE	In	Configuration Complete Signal
18	CAPTURE	Out	State Capture enable for readback
19	CAPCLK	Out	State Capture Clock for readback
20	SWCLK	Out	Software Control to perform clock stepping. This bit should be tied up with any one of the Global Clock buffers on the Virtex chip.
21-31			System defined Input/Output Bits

Table 1. Signals in a Virtex Interface Register.

While Table 1 defines a suggested implementation of the VIR register, it is quite simple to use other variations. The software which communicates with the VIR is configurable and may be modified to reassign bits in this register. Note that in the table, the *In* and *Out* direction is defined with respect to the VIR. That is, *In* means Virtex chip writes to the VIR and *Out* means the VIR writes to the Virtex Chip.

Once this register is defined and provided access through the device driver, data can be read from and written to the Virtex device with no modification to the supplied software API. A diagram illustrating *XVPI*'s hardware interface is shown in Figure 1. More information on the API is provided in the next section.

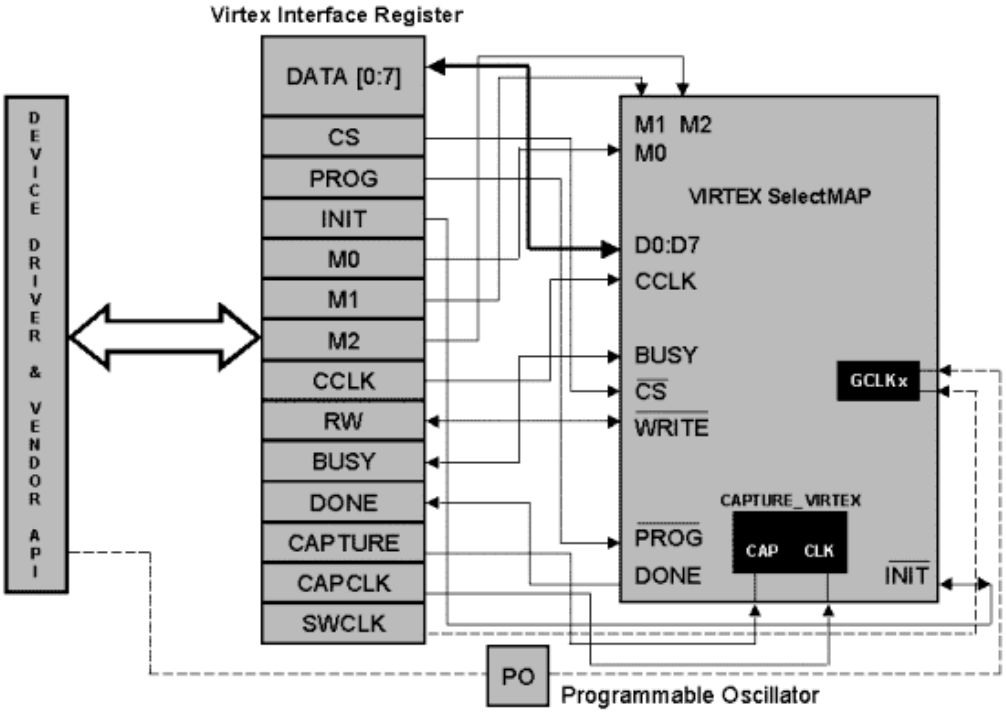

Figure 1. The *XVPI* hardware interface.

3. THE *XVPI* SOFTWARE INTERFACE

Because the goal of *XVPI* is to help designers quickly get functioning Virtex hardware, a complete software API which drives the VIR is supplied with *XVPI*. While this provides a relatively slow interface, it should require little or no software modification by the designer and requires only a single register as an interface.

The *XVPI* Application Program Interface (API) is written completely in the *C* language. It provides a small, simple, layered API which interacts directly with the VIR. This layered design is implemented so that functions and entire layers can be replaced incrementally by faster, hardware assisted versions. This will increase system performance without changing the functionality of the system or requiring changes to higher levels of the software.

Table 2 describes the layers and their functions in the API. *Layer 0* is the most basic layer and consists of the `XVPI_Open()` and `XVPI_Close()` calls. In simple embedded systems, these functions may do nothing but return a "success" value. In systems with some operating system, the `XVPI_open()` call is used to initialize hardware, software and device drivers and provide access to the Virtex device, usually through the VIR register. Similarly the `XVPI_close()` call is used to perform any housekeeping necessary when de-establishing a connection.

Layer	Routines in Layer
Layer 0	XVPI_Open(), XVPI_Close()
Layer 1	XVPI_Read(), XVPI_Write()
Layer 2	XVPI_ReadBlock(), XVPI_WriteBlock(), XVPI_Abort(), XVPI_Reset()

Table 2. API Layers.

Layer 1 provides the direct access to the VIR register. Two functions make up this layer of the API: `XVPI_Read()` and `XVPI_Write()`. In simple embedded systems, these function will read and write the memory or IO location mapped to the VIR register. In systems with some operating system support, these function will typically make calls

to some device driver to access the VIR register. These functions should access all bits in the register atomically, that is, in a single cycle. This assures that invalid intermediate signals are not sent to or read from the Virtex device via the VIR register.

The top layer of the *XVPI* interface, *Layer 2*, contains control routines and routines to access blocks of data from the *SelectMap* port of the Virtex device. The data access routines, `XVPI_ReadBlock()` and `XVPI_WriteBlock()` are used to read and write arrays of bytes to the Virtex *SelectMap* port. In the default inmplementation, this is accomplished by software that makes repeated calls to the *Layer 1* `XVPI_Read()` and `XVPI_Write()` call, both reading and writing data, as well as manipulating the necessary control signals.

Of course, software toggling of all of the control signals, including the clock, to read and write data in this fashion is relatively slow. But for the applications which do not require high performance for download and readback of configuration data, this solution may be perfectly acceptable.

The remaining *Layer 2* API components are `XVPI_Abort()` and `XVPI_Reset()`. The `XVPI_Abort()` call is used to supply a Virtex "abort sequence". This is a synchronization primitive involving the manipulation of the *Chip Select (CS)* and *Read / Write (RW)* signals. It is used primarily when restarting the software and resynchronizing with the Virtex device. It may also be used in situation where reads and writes alternate, to ensure that the bus maintains the correct state. Lastly, the `XVPI_Reset()` call is used simply to provide a hardware reset to the Virtex device.

4. HARDWARE ACCELERATION

While *XVPI* permits Virtex hardware to be designed with a minimum of hardware and software design effort, the resulting interface is necessarily somewhat slow. Every transition of a signal in the *SelectMap* bus requires a write to the VIR register. In simple embedded systems, this may rquire as little as a single processor cycle. In more complex systems with operating system overheads, this may consume as much as thousands of processor cycles. Depending on the application this may be an unacceptable overhead.

XVPI, however, was designed not only to help produce working hardware quickly, but to provide a simple path to accelerate performance once a working system has been established. Some simple modifications present themselves immediately. Using software to clock data into and out of a register is somewhat unnecessary. The VIR register can be modified providing these clock signals in hardware, eliminating code from the *Layer 1* `XVPI_Read()` and `XVPI_Write()` routines, improving performance.

Similar bus hardware interface modifications can be used to eliminate the need to manipulate control signals such as *Chip Select (CS)* and *Read / Write (RW)*, further reducing the code in *Layer 1.* Finally, if hardware assisted block access such as DMA is available, *Layer2* may be replaced by these calls, eliminating the dependence on *Layer 1* entirely.

What is notable here is that a system can be brought up quickly, then enhanced incrementally. Once a stable system has been established, offsetting hardware and software changes can be made to gradually increase performance, without changing the software API.

5. INTERFACING TO *JBITS*

JBits is a set of Java classes that provide an Application Program Interface (API) into the Xilinx FPGA family configuration bitstream[2].[4] This interface operates on either bitstreams generated by Xilinx design tools, or on bitstreams read back from actual hardware. This provides the capability of designing, modifying and dynamically reconfiguring circuits in Xilinx FPGA devices.[3] While *JBits* can modify Virtex configuration bitstreams off-line, it may also be used to modify configuration data in operating hardware.

JBits communicates with Virtex hardware through a standard interface called *XHWIF*. This interface supplies a high-level Java API to perform functions such as configuration bitstream download, configuration bitstream readback, reset, clock control and other similar functions. Once the *XVPI* interface was established, it was relatively simple to construct an *XHWIF* interface which communicated directly with *XVPI*. Figure 2 shows a diagram of how *XHWIF* uses *XVPI* to provide support for *JBits* and *JBits* applications.

While the *JBits* design capabilities may not be of direct interest to many Virtex users, the *JBits* tool suite comes with utilities that should be useful to most designers. The first is the *BoardScope* is a graphical and interactive debug tool.[6] *BoardScope* enables users to look examine internal state of the Virtex device in various formats. The second

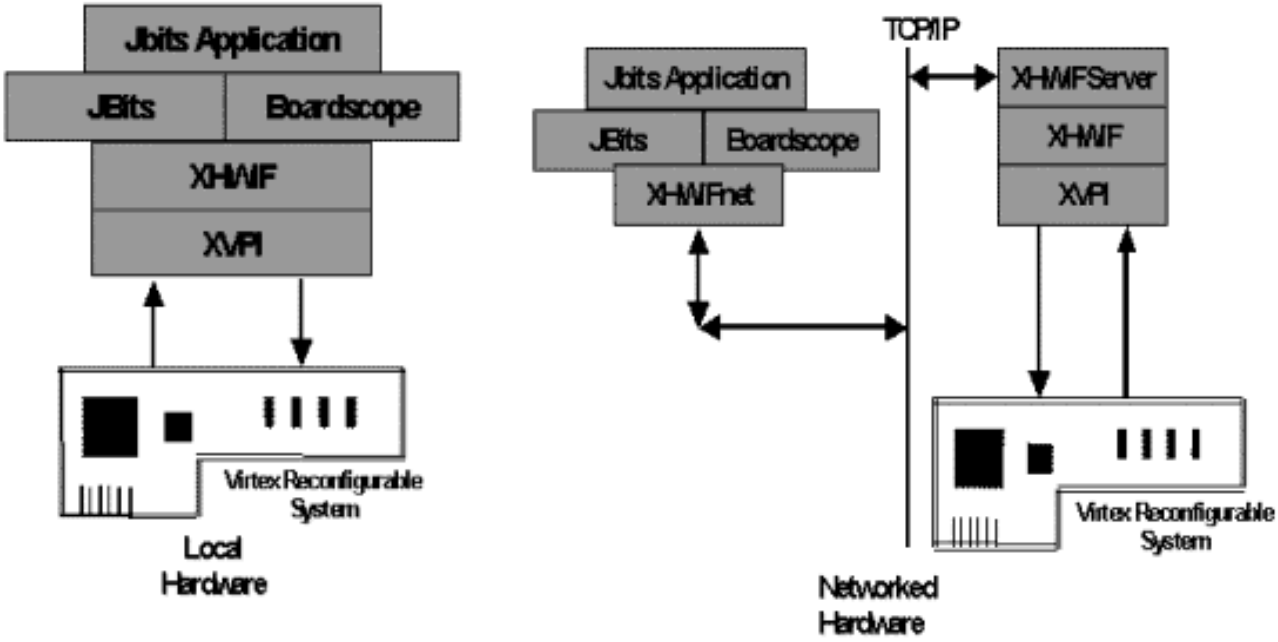

Figure 2. *XVPI*'s system level interface to *JBits*.

capability provided by *XHWIF* is remote network access. Any *XHWIF* application, including the *BoardScope* debug tool, can be run remotely across a network. This permits limited hardware resources to be easily shared among a number of users. Figure 2 shows a diagram of this remote network interface via the *XHWIFServer* remote server software.

6. CONCLUSIONS

XVPI provides a set of hardware design specifications combined with software and software API specifications to produce a flexible system to bring up and access Virtex-based hardware. Provided with a simple hardware interface consisting of a single software accessible register, *XVPI* can be used to bring up a complete Virtex-based hardware system with little or no additional effort.

This initial configuration is completely software-driven. All hardware control and handshaking is handled by supplied *XVPI* software routines. While this makes functionality available quickly, performance may be too slow for some applications. In these cases, hardware and software control may be traded off incrementally, increasing performance. This can be accomplished in stages, without changing the software interfaces or modifying upper layers of the software API. In addition, the *JBits* tool suite, with accompanying *BoardScope* interactive graphical debug tool along with remote network access is made available through a supplied interface.

In early commercial Virtex board designs, much of the functionality of the *SelectMap* fast configuration port was not implemented. This is often the case with new and somewhat unusual hardware capabilities. To help to remedy this situation, *XVPI* was implemented. Supplied with complete source code, *XVPI* enables designers to bring up new Virtex hardware, along with a substantial software tool set, all by specifying a simple single register interface. And once this hardware is functional, *XVPI* permits incremental performance increases to be made without disturbing existing and operational parts of the system.

REFERENCES

1. Carl Carmichael. Virtex FPGA series configuration and readback. Xilinx Application Note XAPP138, version 1.1, Xilinx, Inc., July 1999.
2. Steven A. Guccione and Delon Levi. XBI: A java-based interface to FPGA hardware. In John Schewel, editor, *Configurable Computing: Technology and Applications, Proc. SPIE 3526*, pages 97–102, Bellingham, WA, November 1998. SPIE – The International Society for Optical Engineering.
3. Steven A. Guccione and Delon Levi. Design advantages of run-time reconfiguration. In John Schewel, editor, *Reconfigurable Technology: FPGAs for Computing and Applications, Proc. SPIE 3844*, pages 87–92, Bellingham, WA, September 1999. SPIE – The International Society for Optical Engineering.
4. Steven A. Guccione, Delon Levi, and Prasanna Sundararajan. JBits: A java-based interface for reconfigurable computing. In Richard Katz, editor, *Second Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD)*, September 1999.
5. Steve Kelem. Virtex configuration architecture advanced users' guide. Xilinx Application Note XAPP151, version 1.1, Xilinx, Inc., July 1999.
6. Delon Levi and Steven A. Guccione. BoardScope: A debug tool for reconfigurable systems. In John Schewel, editor, *Configurable Computing Technology and its use in High Performance Computing, DSP and Systems Engineering, Proc. SPIE Photonics East*, pages 239–246, Bellingham, WA, November 1998. SPIE – The International Society for Optical Engineering.
7. Xilinx, Inc. *Xilinx Data Book*, 2000.