XAPP694 (v1.1.1) November 19, 2007

# Reading User Data from Configuration PROMs

## Summary

This application note describes how to retrieve user-defined data from Xilinx configuration PROMs (XC18V00 and Platform Flash devices) after the same PROM has configured the FPGA. The method to add user-defined data to the configuration PROM file is also discussed. The reference design described in this application note can be used in any of the following Xilinx FPGA architectures: Spartan™-II, Spartan-IIE, Spartan-3, Virtex™, Virtex-E, Virtex-II, and Virtex-II Pro.

## Introduction

After an FPGA is configured, it is often necessary to retrieve user-defined data that is used by the FPGA during operation. The user-defined data needs to be retrieved from an external storage device, requiring an external control circuit to interface to the storage device. This requirement implies additional time for logic design and board level engineering, not to mention extra board real estate requirements and a higher FPGA pin count.

Xilinx configuration PROMs are generally used to store an FPGA design, which is downloaded to the FPGA upon system power-up. In most cases, configuration storage is the PROM's only function, and its capacity is usually not fully used by the FPGA design, leaving the unused portion of the PROM as wasted space.

The design in this application note describes how user-defined data can be stored and retrieved from Xilinx configuration PROMs using existing connections and only one user I/O, reducing the FPGA pin count, component count, board space, and overall system cost. The user-defined data can be an Ethernet MAC ID, bitstream revision code, coefficients, processor code, ASCII data to be displayed, encryption codes, and so on. A Perl script that automatically modifies existing configuration PROM files with user-defined data is available for download. The script supports Intel Object (`.mcs`) and HEX (`.hex`) file formats, with optional bit-swapping.

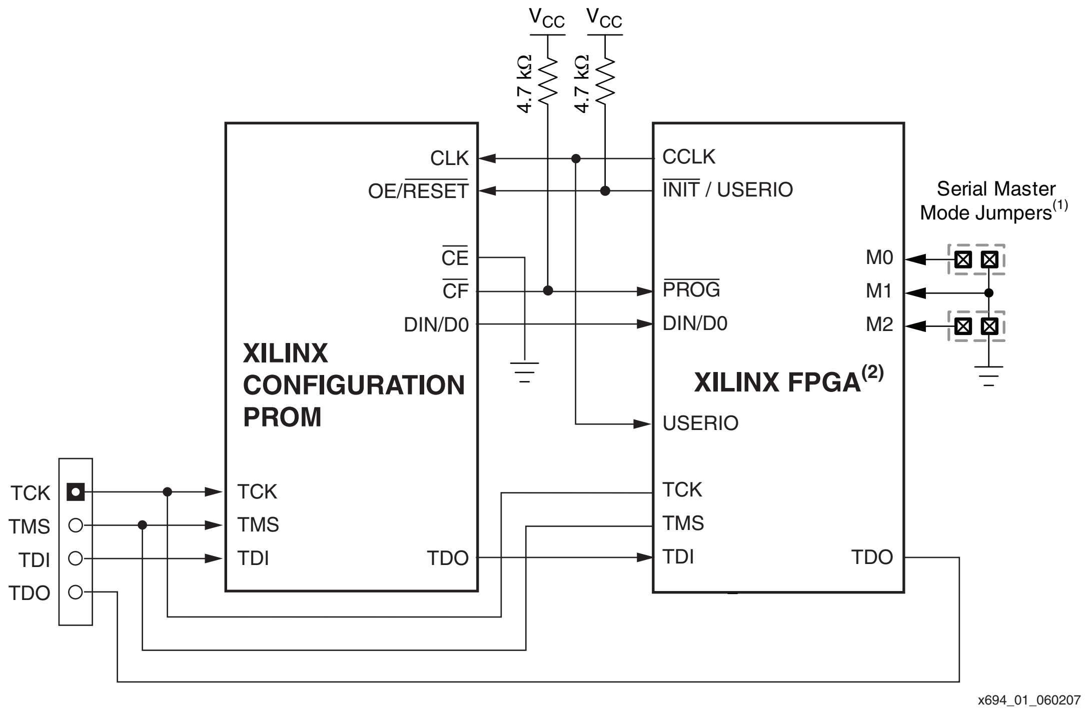
## Considerations for PROM and FPGA Connections

Figure 1 clearly shows the minimum connections necessary to create a suitable interface between the PROM and the FPGA. This interface allows the FPGA to retrieve data from the PROM before and after configuration and is the simplest implementation for reading user-defined data from a configuration PROM. However, other implementations do exist and are discussed later in this application note. For an in-depth description of the logic implementation for Figure 1, see "Macro Implementation," page 5.

The connections in Figure 1 are required for the following reasons:

- In any master configuration mode, the configuration clock CCLK generated by the FPGA stops toggling after the FPGA is successfully configured, preventing the PROM's address counter from advancing beyond the FPGA design stored in the PROM.
- When the $\overline{\text{CE}}$ pin on the PROM goes High, the PROM's address counter is held reset at zero. If the $\overline{\text{CE}}$ pin of the PROM is held High, the address counter of the PROM cannot be advanced. In this state, the PROM DATA outputs are High-Z (explained in Table 1).
- When the $\overline{\text{INIT}}$ pin goes Low, indicating either that a CRC error occurred during configuration or that $\overline{\text{INIT}}$ has become user I/O initialized to a Low state after configuration, the address counter of the PROM is held reset to zero. If the OE/$\overline{\text{RESET}}$ pin of the PROM

is held Low, the address counter of the PROM cannot be advanced. In this state, the PROM DATA outputs are High-Z.



**Notes:**

1. The jumpers on the M0 and M2 pins are enhancements for prototyping and debugging, to allow changing the FPGA configuration mode from the default master-serial mode, M[2:0]=[000], to JTAG mode, M[2:0]=[101].
2. This schematic applies to Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro FPGAs. Changes to post-configuration pin functionality in newer FPGA families can require additions or modifications to the schematics. See the appropriate FPGA data sheet for details.

*Figure 1:* **PROM and FPGA Connections with Additional Control Signal**

*Table 1:* **Truth Table for PROM Control Inputs**

| Control Inputs | | Internal Address(1) | Outputs |
|---|---|---|---|
| OE/$\overline{\text{RESET}}$ | $\overline{\text{CE}}$ | | DATA |
| High | Low | If address ≤ TC: increment | Active |
| | | If address > TC: do not change | High-Z |
| Low | Low | Held reset | High-Z |
| High | High | Held reset | High-Z |
| Low | High | Held reset | High-Z |

**Notes:**

1. TC = Terminal Count of address counter. Address = address count.

The above considerations have several solutions:

- The $\overline{\text{CE}}$ pin can be connected to ground, enabling the PROM. An alternative is to connect the $\overline{\text{CE}}$ pin to a user I/O pin on the FPGA with an external pull-down resistor. This resistor must be sufficiently strong to hold the signal at a valid logic Low during the pre-configured state of the FPGA (when the FPGA I/O is High-Z with its possible internal pull-up resistor). This external resistor allows the user to enable or disable the PROM and reduce power consumption.
- The OE/$\overline{\text{RESET}}$ pin should be connected to $\overline{\text{INIT}}$, allowing the FPGA to re-initiate a configuration if a CRC error occurs during configuration. The $\overline{\text{INIT}}$ pin becomes user I/O after configuration and so can be configured to output a logic High to keep the PROM outputs enabled.

  ***Note:*** Consult the FPGA data sheet for the function of the INIT pin after configuration.
- With the OE/$\overline{\text{RESET}}$ pin held High and the $\overline{\text{CE}}$ pin held Low, the PROM's address counter is able to advance beyond the FPGA configuration data.
- In master configuration modes, the clock signal to the PROM must be generated by one of the user I/O and connected in parallel with the CCLK pin since the clock needs to be controlled by the FPGA.

The configuration clock CCLK stops toggling after the STARTUP phase of the FPGA has completed provided the **`-persist`** option in BitGen is set to **`off`** (default). After CCLK stops toggling, the pin is 3-stated with a weak pull-up resistor connected to $V_{CCO}$. However, depending on configuration options set during bitstream generation, the number of CCLK cycles needed to complete the STARTUP phase is uncertain. This uncertainty makes it difficult to know exactly at what address the PROM's address counter has stopped, which means that user-defined data stored directly after the FPGA configuration can not be retrieved reliably. Figure 2 shows this concept graphically.

This problem can be overcome by employing a mechanism similar to the configuration logic inside the FPGA. After power-up, the FPGA starts receiving data from the PROM. The FPGA, however, does not load any data received from the PROM until it has received a synchronization word. For Xilinx FPGAs based on the Virtex architecture, the synchronization word is `AA995566h`. This same idea can be used in the FPGA after configuration to look for user-defined data.

> ***Note:*** For more information regarding FPGA configuration, see XAPP501, *Configuration Quickstart Guidelines*, and XAPP138, *Virtex FPGA Series Configuration and Readback.* For information regarding configuration of Virtex-II and Virtex-II Pro devices, see the "Configuration" chapters of the UG002, *Virtex-II Platform FPGA User Guide*, and UG012, *Virtex-II Pro and Virtex-II Pro X FPGA User Guide*.

After the FPGA is configured, it sends out clock pulses on the user I/O connected to the CLK pin of the PROM. With every rising edge of the clock, the PROM's address counter is incremented by one, and the data at that address is presented on the data pins of the PROM.

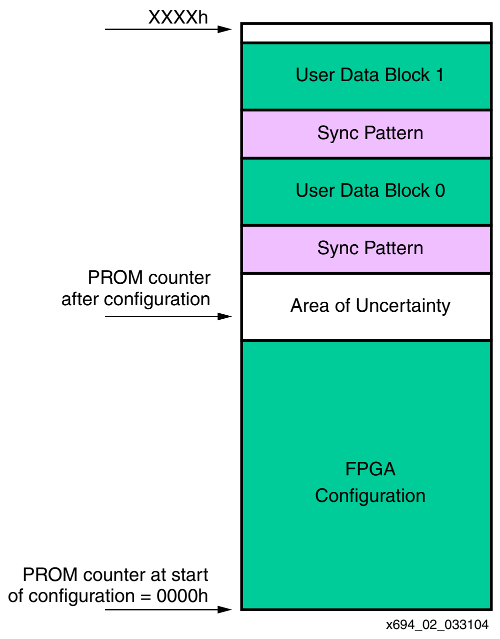


*Figure 2:* **PROM Memory Map Showing Address Counter Uncertainty**

A Perl script exists that adds synchronization patterns and user-defined data to configuration PROM files. For a detailed description on how to use the script, see "Adding User Data to the PROM," page 8. For now, an example of how to use the script is given. All files used in the example, including the script, can be found in the `\Perl_Script` directory of the reference design archive (see "Design Resources," page 12).

The PROM file containing the FPGA configuration is named `prom_file.mcs` and the file containing the synchronization patterns and user-defined data is named `user_data.txt`. The command line that invokes the script to add the contents of `user_data.txt` to `prom_file.mcs` is:

```
xilperl pc.pl -f F -uf user_data.txt -pf prom_file.mcs -ps 2
```

The script produces a PROM file named `new_prom_file.mcs`. All files can be examined in a text editor. The `user_data.txt` file has four separate blocks of user-defined data, each defined by a separate synchronization pattern. The synchronization pattern in this example is `8F9FAFBF`h and the user-defined data that follows is ASCII encoded data. When converted, the ASCII code represents:

```
XAPP 694 DATA BLOCK x
```

where x represents the current data block.

Other choices of synchronization patterns and user-defined data can be implemented. For a more detailed description, see "Adding User Data to the PROM," page 8, and "Other Implementations," page 10.

## Macro Implementation

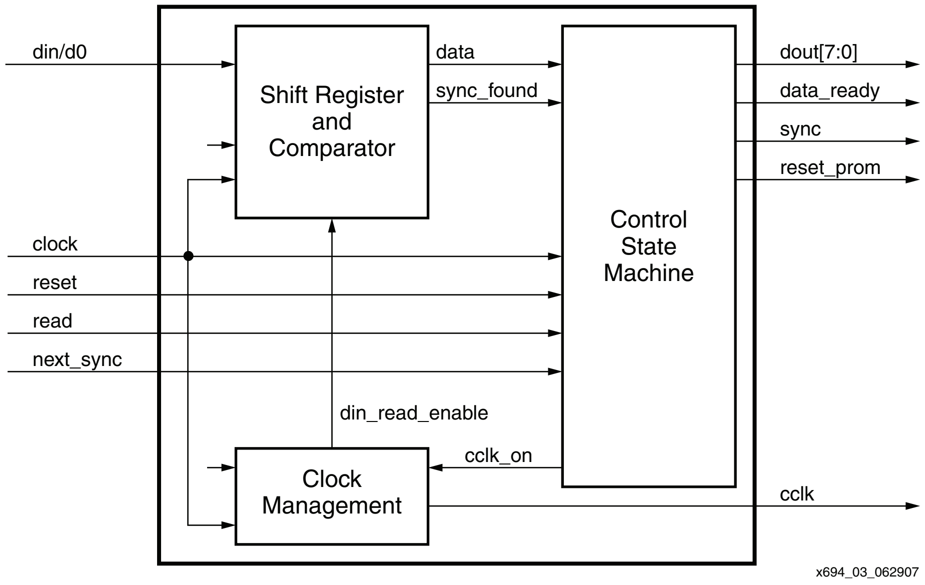


*Figure 3:* **PROM Reader Macro**

Figure 3 shows the macro used to read user-defined data from the PROM after the FPGA is configured. A description of the input and output signals and their function can be found in Table 2.

*Table 2:* **PROM Reader Macro Signal Descriptions and Functions**

| Signal | I/O Direction | Description |
|---|---|---|
| clock | Input | All signals are registered on the rising of this signal. |
| reset | Input | This signal resets all logic to the initial state. This signal is asynchronous and active Low. |
| din/d0 | Input | Data from the PROM is presented on this signal. |
| read | Input | This signal is used to instruct the macro to retrieve the next 8 bits of data from the PROM. This signal is active Low. |
| next_sync | Input | This signal is used to instruct the macro to search and find the next synchronization pattern in the PROM. This signal is active Low. |
| dout[7:0] | Output | This is the data bus on which actual user-defined data is presented. Valid user-defined data is indicated by the data_ready signal going Low for one clock cycle. |
| data_ready | Output | This signal indicates that valid data is present on the dout[7:0] pins. This signal is active Low. |
| sync | Output | This signal indicates detection of a synchronization pattern in the data retrieved from the PROM. The signal is active Low. |
| reset_prom | Output | This signal outputs a Low whenever the macro is reset. This signal can be connected to either the $\overline{\text{CE}}$ or OE/$\overline{\text{RESET}}$ pin of the PROM to reset the PROM's address counter. This signal is connected to the PROM OE/$\overline{\text{RESET}}$ pin in the reference design. If this signal is connected to the PROM $\overline{\text{CE}}$ pin, the signal must be inverted. |
| cclk | Output | This signal mimics the CCLK signal from the FPGA during configuration. |

A description of each block of the macro and the functionality that it provides follows.

## Clock Management

This block generates clock enables at appropriate clock edges so that data from the PROM is captured correctly by the FPGA. It also generates a signal that mimics the CCLK signal provided during configuration.

With every rising edge of CCLK, the PROM presents new data on its data pins. On the falling edge of CCLK, a clock enable signal allows the data to be captured by the Shift Register and Comparator block on the rising edge of the system clock, ensuring an adequate setup-and-hold time exists for the input registers of the FPGA.

The CCLK signal is generated by enabling a register that provides a signal equivalent to the system clock divided down to 10 Mhz. The enable signal to the register is provided by a rotating logic High through an SRL16. The length of the SRL16 is determined by the system clock frequency, which the user enters via a parameter. Using this method to divide the system clock is advantageous, since it does not require the use of DLLs or DCMs and still keeps the circuit synchronous.

Figure 4 shows the timing relationship between the data and CCLK.

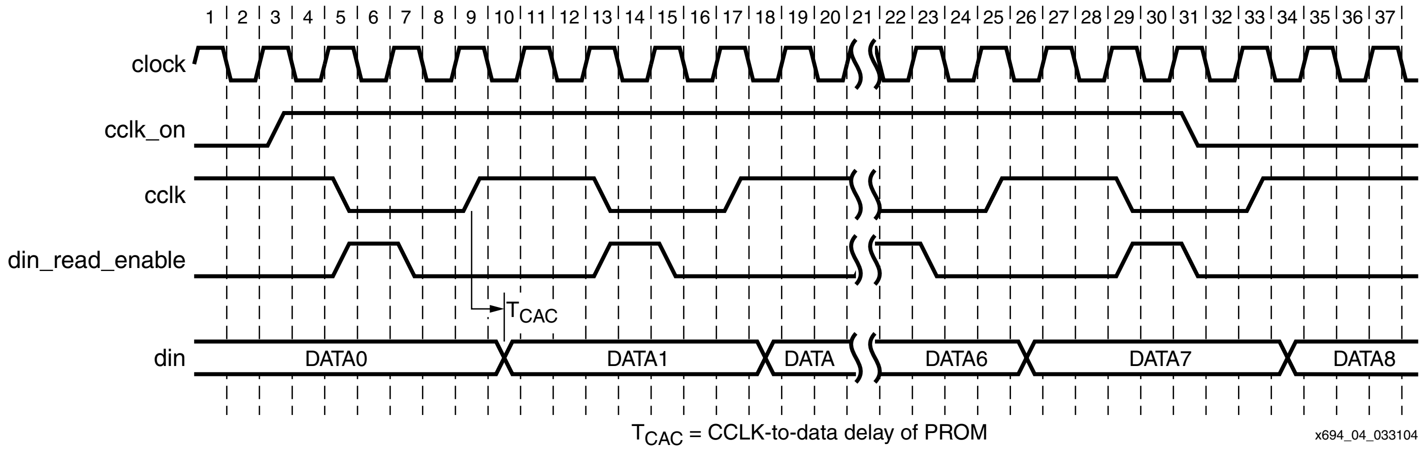


*Figure 4:* **Clock Management Block Timing**

## Shift Register and Comparator

This block captures data from the PROM one bit at a time. Whenever a data bit is clocked into the FPGA, the data is compared to a synchronization pattern to determine the start position of user-defined data. The length of the synchronization pattern, as well as the pattern itself, can be user-defined. The pattern can also be changed during the operation of the FPGA, allowing the user to define and search for multiple sets of user-defined data, a capability that makes this circuit very flexible.

Upon finding the synchronization pattern, a sync pulse is generated to indicate that synchronization is achieved. The sync pulse is valid for as long as the synchronization pattern remains valid.

Figure 5 shows the timing relationship of the sync signal and the data.

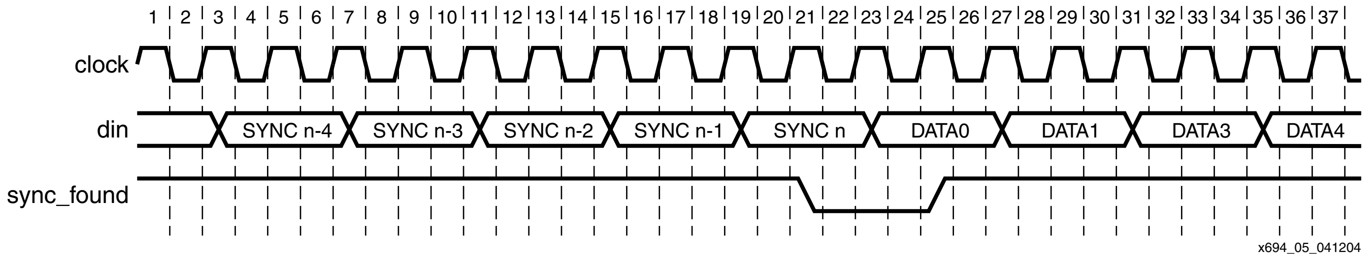


*Figure 5:* **Shift Register and Comparator Block Timing**

The Shift Register and Comparator block continues to check for a synchronization pattern, even after synchronization is achieved. If synchronization is achieved and the user-defined data also happens to match the synchronization pattern, a sync pulse is still generated. Therefore, only the data_ready signal qualifies actual user-defined data. The data_ready pulse is described in the Control State Machine section.

## Control State Machine

This block provides control for all other modules in the macro. It determines how many data bits to retrieve from the PROM after synchronization, when to search for the next synchronization pattern, when to retrieve the next byte of user-defined data, and when to reset the PROM's address counter. Figure 6 shows the different states implemented in the state machine.

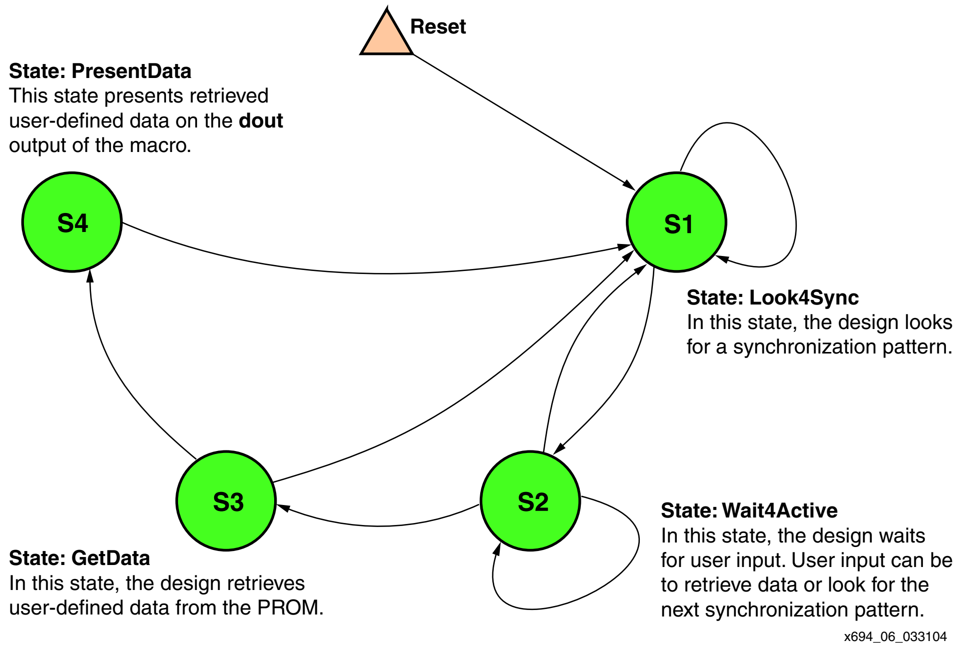

Reset
State: PresentData
This state presents retrieved user-defined data on the dout output of the macro.
S4
S1
State: Look4Sync
In this state, the design looks for a synchronization pattern.
S3
S2
State: GetData
In this state, the design retrieves user-defined data from the PROM.
State: Wait4Active
In this state, the design waits for user input. User input can be to retrieve data or look for the next synchronization pattern.
x694_06_033104

*Figure 6:* **Control State Machine Diagram**

Figure 7 shows the timing relationship of the data_ready pulse, together with data on din/dout and the sync pulse.

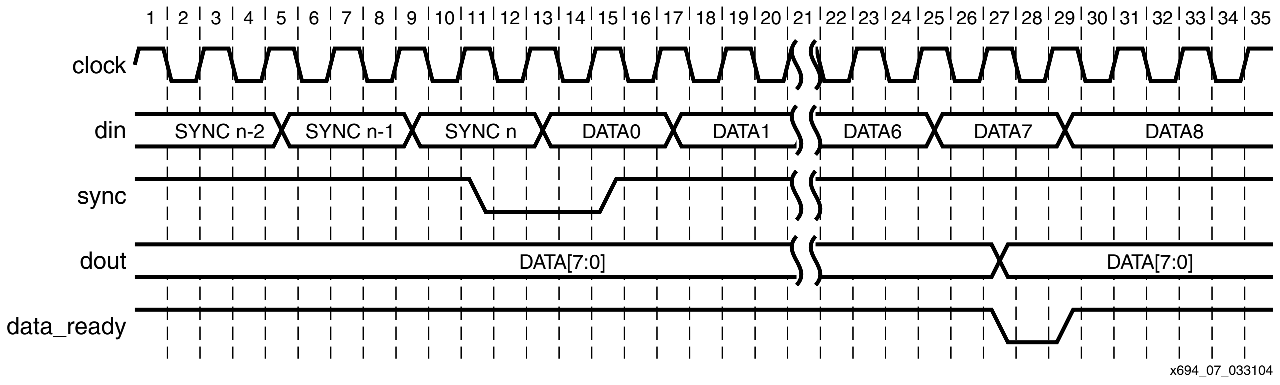


*Figure 7:* **Control State Machine Block Timing**

## Adding User Data to the PROM

In order to read user-defined data from the configuration PROM, a method must exist that allows the designer to easily add user-defined data to the PROM file. A Perl script that adds user-defined data to existing PROM files for Intel Object and Hex file formats can be found in the `\Perl_Script` directory of the reference design for this application note. The Perl script does not support Motorola EXORmacs and TEKTRONIX TEK formats. In order to provide an explanation of what the script does and how it works, some details concerning PROM file formats are discussed in the next three sections:

- Bit-swapping
- Records, byte counts and checksums
- PROM size

### Bit-Swapping

Every byte of data retrieved from the configuration PROM is presented in a bit-swapped fashion: that is, the MSB of every byte is read first, and the LSB is read last. It is important to bear this in mind, as it has an effect on how data is presented and used inside the FPGA. An example of bit-swapping is shown in Figure 8, which illustrates that the way data is presented in the PROM file differs from the way it is presented inside the FPGA. (The values chosen in the example are arbitrary.)

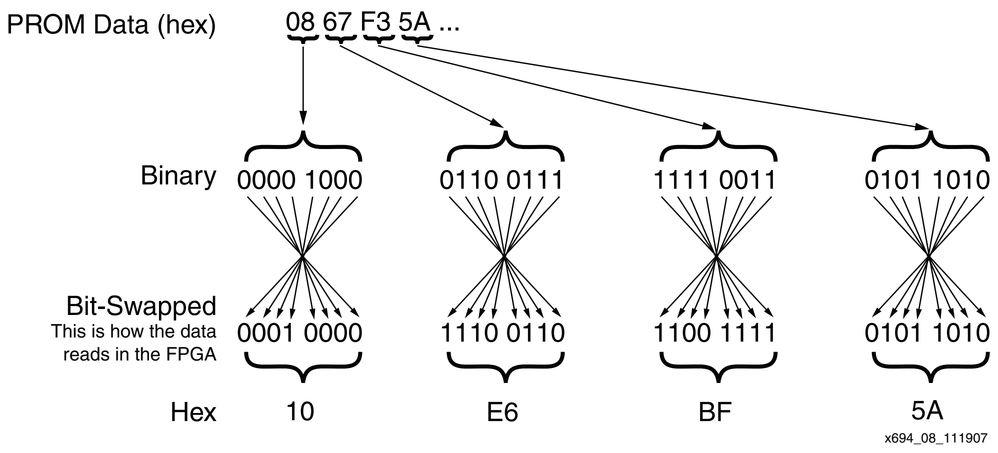


*Figure 8:* **Bit-Swapping of PROM Data**

Although other PROM file formats exists, the Perl script discussed in this application note supports only the Intel Object and Hex file formats, so only these formats are discussed here.

Bit-swapping is a feature of the Intel Object (`.mcs`) and Hex PROM file formats. The Xilinx iMPACT configuration tool supports both formats. The Hex file format can have bit-swapping enabled or disabled. The bit-swapping is important to note when adding user-defined data to the configuration PROM file, since it influences the way that data is read by the FPGA, particularly the synchronization pattern. The bit-swapping can be dealt with in two ways:

1. The user-defined data can be added to the configuration PROM file without any bit-swapping. The data can then be bit-swapped inside the FPGA, if necessary, after it is read.
2. The user-defined data can be added to the configuration PROM file with previously bit-swapped bytes. There is no need to bit-swap the data inside the FPGA after it is read.

Either of the above mentioned methods is suitable, and although the first option mentions swapping the bits inside the FPGA, this option does not add logic resources to realize the interface. The reference design in this application note implements the first option.

As already mentioned, the Hex file format can have bit-swapping enabled or disabled. When using the Perl script, the user can specify whether the user-defined data is to be added in a bit-swapped fashion or not by using the **-swap** option. If the **-swap** option is **on**, then all bytes of the user-defined data are bit-swapped before they are added to the existing Hex file. If the **-swap** option is **off**, then the user-defined data is simply appended to the existing Hex file.

## Records, Byte Counts and Checksums

When adding user-defined data to an existing configuration PROM file, there are a number of data fields that need to be added in order for the file to be successfully downloaded into a PROM. Since the Perl script only supports the Intel Object and Hex file formats, only these are discussed here. The Hex file formats do not require any additional data fields to be added. However, the Intel Object file format requires a Start Character, Byte Count, Address, Record Type, and Checksum fields, in addition to the actual data. The Perl script automatically calculates these fields and inserts them into the existing PROM file. In order for the script to calculate these values correctly, the user-defined data must be presented 16 bytes per line.

Table 3 lists the input data types used by the iMPACT configuration software, with Table 4 through Table 6 providing details on each type.

*Table 3:* **Input Data Types**

| Data Type | Description |
|---|---|
| 00 | Data Record |
| 01 | End of File Record (signals the end of the file) |
| 04 | Extended Linear Address Record (provides the offset to determine the absolute destination address) |

The checksum is the two's complement of the binary summation of the preceding bytes in the record (including the byte count, address, and any data bytes), in hexadecimal notation.

*Table 4:* **Input Data Type 00**

| : | BC | AAAA | 00 | hhhh....h | CC |
|---|---|---|---|---|---|
| Start Character | Byte Count | Hex Address | Record Type | hh = 1 Data Byte | Checksum |
| | 2 Characters | 4 Characters | 2 Characters | 2 up to 32 Characters | 2 Characters |

*Table 5:* **Input Data Type 01**

| : | 00 | 0000 | 01 | FF |
|---|---|---|---|---|
| Start Character | Byte Count | Hex Address | Record Type | Checksum |
| | 2 Characters | 4 Characters | 2 Characters | 2 Characters |

The extended linear address record (Type 04) defines bits 16 to 31 of the 32-bit linear base address. This address is added to subsequent data record addresses to provide the absolute address.

*Table 6:* **Input Data Type 04**

| : | 02 | 0000 | 04 | hhhh....h | CC |
|---|---|---|---|---|---|
| Start Character | Byte Count | Hex Address | Record Type | 2 Byte Offset | Checksum |
| | 2 Characters | 4 Characters | 2 Characters | 2 up to 32 Characters | 2 Characters |

### PROM Size

Care must be taken not to add too much user-defined data to the PROM, since doing so causes the configuration tools to reject the PROM file. To select a PROM that can store both the FPGA configuration and the user-defined data, simply add the number of bits used for the FPGA configuration to the number of user-defined data bits and synchronization patterns. The number of bits used for the FPGA configuration can be found by consulting the appropriate FPGAs data sheet.

***Note:*** A detailed description of how to use the Perl script can be found in the `readme.txt` file in the `\Perl_Script` directory of the reference design.

## Reference Design Implementation

The reference design described in this application note can be implemented on the Xilinx Virtex-II High Speed Demo Board**.** A block diagram of the design is shown in Figure 9.

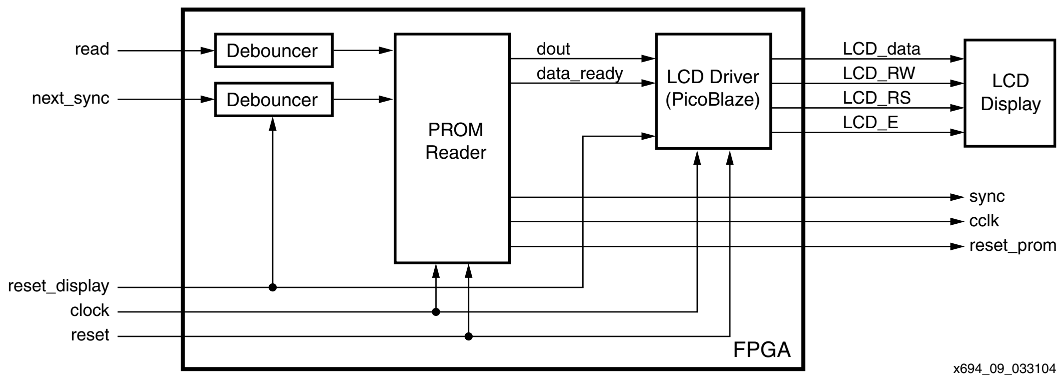


*Figure 9:* **Reference Design for Virtex-II High Speed Demo Board**

The user-defined data retrieved from the PROM is displayed on an LCD display. The LCD driver is implemented using PicoBlaze™.

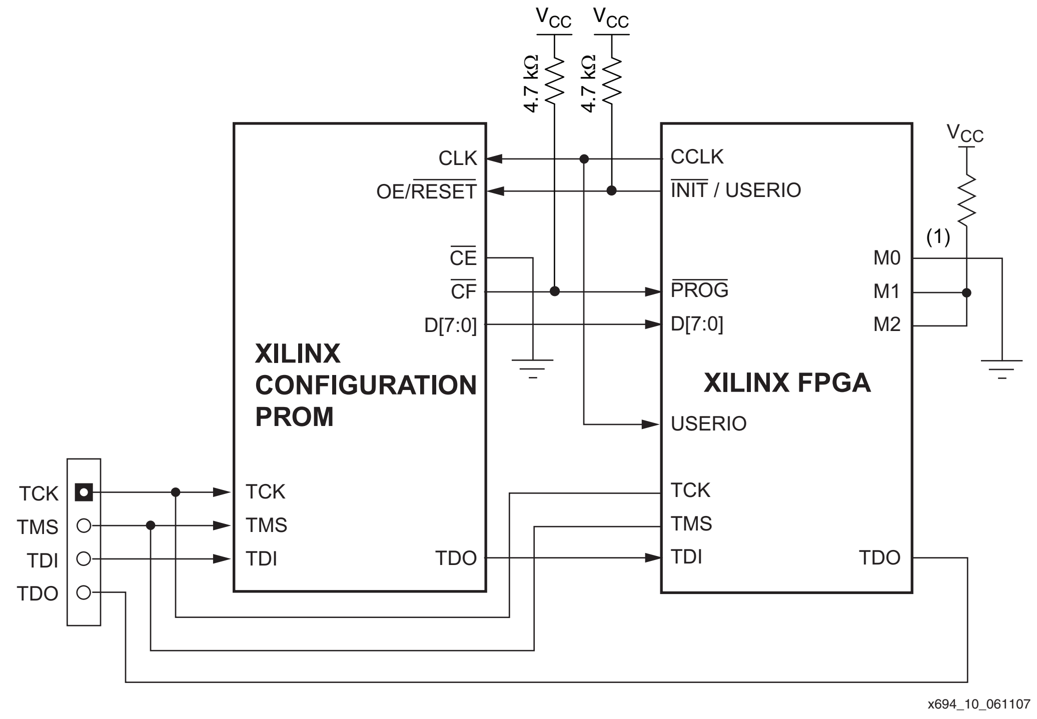
## Other Implementations

Until now, this application note has dealt with reading user-defined data from a configuration PROM in a serial fashion. What happens when the PROM is connected in SelectMAP™ mode? What if different synchronization patterns need to be used, and how does the designer keep track of which data block is currently being read?

These questions are dealt with in the following sections.

### Parallel Implementation

The reference design described in this application note can be used for several implementations. A macro that allows the FPGA to read user-defined data from the configuration PROM in SelectMAP mode—that is, 8 bits at a time—is also available. A connection diagram for this implementation can be seen in Figure 10. The functionality of the parallel macro is the same as that of the serial macro, except that additional manipulation needs to be done in the Shift Register and Comparator block.



**Notes:**

1. Mode pin connections can differ for different architectures.
2. This schematic applies to Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, and Virtex-II Pro FPGAs. Changes to post-configuration pin functionality in newer FPGA families can require additions or modifications to the schematics. See the appropriate FPGA data sheet for details.

*Figure 10:* **PROM and FPGA Connections in SelectMAP Mode**

The connections shown in Figure 10 are the same as in Figure 1, page 2, except that now an 8-bit connection exists between the PROM and the FPGA.

## Synchronization Patterns

The reference design automatically scales the Shift Register and Comparator blocks depending on a length parameter. The parameter is entered in the HDL code, and the compiler uses this parameter to scale the block. The length parameter is the exponent of 2 that determines the number of bits used in the synchronization pattern:

Number of bits in synchronization pattern = $2^{LENGTH}$

Therefore, if length = 5 then

Number of bits in synchronization pattern = $2^5$ = 32 bits

The designer can choose different lengths for the synchronization pattern based upon the above requirements. Although the pattern can be specified as a certain length, any pattern shorter than the length specified can be used during operation, since the bits not used are simply tied to ground.

## Multiple Data Blocks

Each block of user-defined data can be specified by using either the same synchronization pattern or a different synchronization pattern for each block (Figure 11).

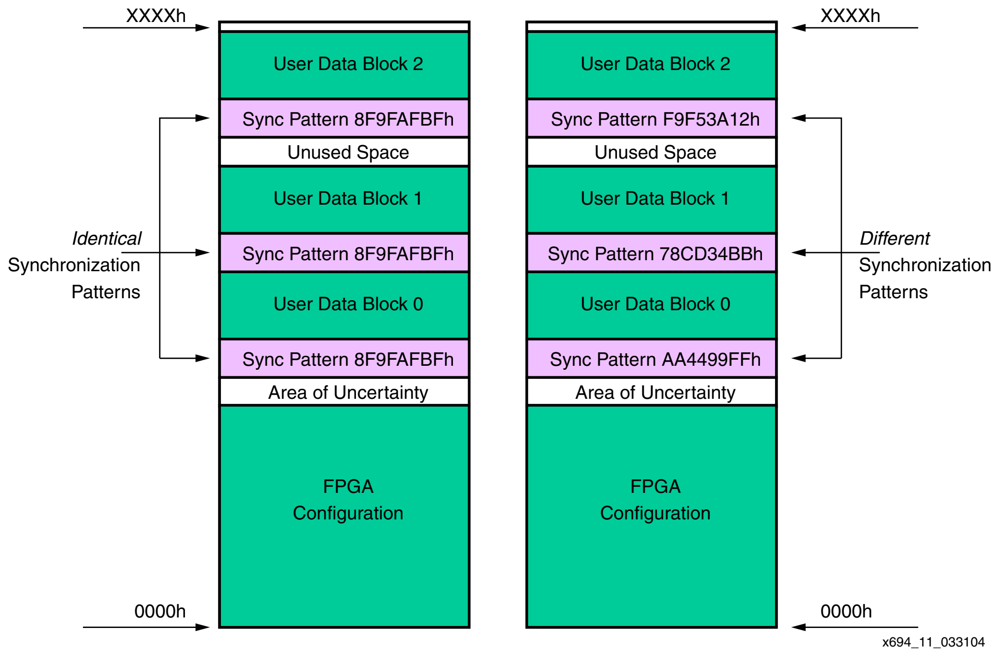


*Figure 11:* **Using Identical and Different Synchronization Patterns**

Both methods have advantages and disadvantages.

When using a synchronization pattern that remains constant, the user does not have to change the pattern during operation. However, it is not always clear which data block is the current one. A counter would need to be implemented in order to keep track of the data blocks.

By using different synchronization patterns for each data block, the blocks can be uniquely identified and addressed. There is no need to keep track of which data block is currently being used, since this is indicated by the pattern. The user, however, needs to change the pattern depending on the data block that is required (requiring that the synchronization patterns need to be stored). The patterns can be hard-coded in the FPGA design or retrieved from pre-initialized block RAMs. The patterns can also be stored as user-defined data in the PROM. The first data block in the PROM contains all the synchronization words for all the other data blocks stored in the PROM, avoiding having to hard-code the patterns in the FPGA design or initialize block RAMs.

## Conclusion

The design described in this application shows an efficient and cost-effective approach to retrieving user-defined data from Xilinx configuration PROMs using existing connections and only one user I/O.

## Design Resources

The reference design described in this application note can be downloaded from:

https://secure.xilinx.com/webreg/clickthrough.do?cid=55722

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 05/26/04 | 1.0 | Initial Xilinx release. |
| 07/16/07 | 1.1 | • Figure 1, page 2 updated.<br>• Figure 3, page 5 updated.<br>• Note, and pull-ups added to Figure 10, page 11.<br>• Other minor fixes and updates.<br>• In addition to text updates, associated Perl script (pc.pl) updated to v1.03. |
| 11/19/07 | 1.1.1 | • Updated URLs.<br>• Updated document template. |

## Notice of Disclaimer