Securing data in FPGAs

FPGAs have non-volatile memory which could be used for storing user settings and ID tag data during system shutdown. **Gordon Hands** looks at the different approaches for achieving this

The use of field programmable gate arrays (FPGAs) continues to increase within many kinds of systems, from consumer devices to industrial control equipment. A common need in many of these applications is the ability to store data such as calibration information, user settings and identification tags while the equipment is powered off. If the equipment has an embedded microprocessor, then the processor's code must also be stored.

For systems where cost, part count and board footprint are not concerns, a standard flash memory device can typically be used for these data storage needs. However, in many cases these constraints lead designers to search for a way to deploy dual-purpose components within their systems. FPGAs always have non-volatile storage associated with them, whether they are monolithic non-volatile devices or SRAM devices using a boot memory. In either case a natural question is whether some of this non-volatile memory can be used to store data during system shutdowns.

Key Considerations

Before choosing which approach to take, it is useful to consider the requirements of the application. The first factor to consider is whether read/write capability is required, or whether read only operation is acceptable. The next factor to consider is the amount of data to be stored. Does the application require just a few bits of storage, or are thousands to millions of bits needed? Another item to consider is the number of write cycles required by the application. Finally, the speed of read and write operations along with their nature, sequential or random, must be reviewed.

Initialised Block RAM

Most FPGAs allow the block RAMs within their fabrics to be initialised from their boot device. This provides an excellent method of providing a relatively large amount of ROM for use storing look up tables or program code. These ROMs support high-speed random reads and can be reprogrammed along with the rest of the FPGA configuration. If all that the design requires is a small amount of ROM, this approach should not be overlooked.

SPI Reuse

All modern SRAM FPGAs allow configuration from SPI memory. Because extra bits within the SPI memory are cheap and consume no added board space, using additional space in the SPI memory for the storage of data often makes sense. It is usually easy to take control of the SPI interface after an FPGA has used it for configuration (although this can vary, so the designer should confirm this prior to committing to this approach) and building an SPI

Although SPI devices allow random reads for many applications, higher performance can be achieved by caching the current sector using the FPGA's block memories

uring system shutdown

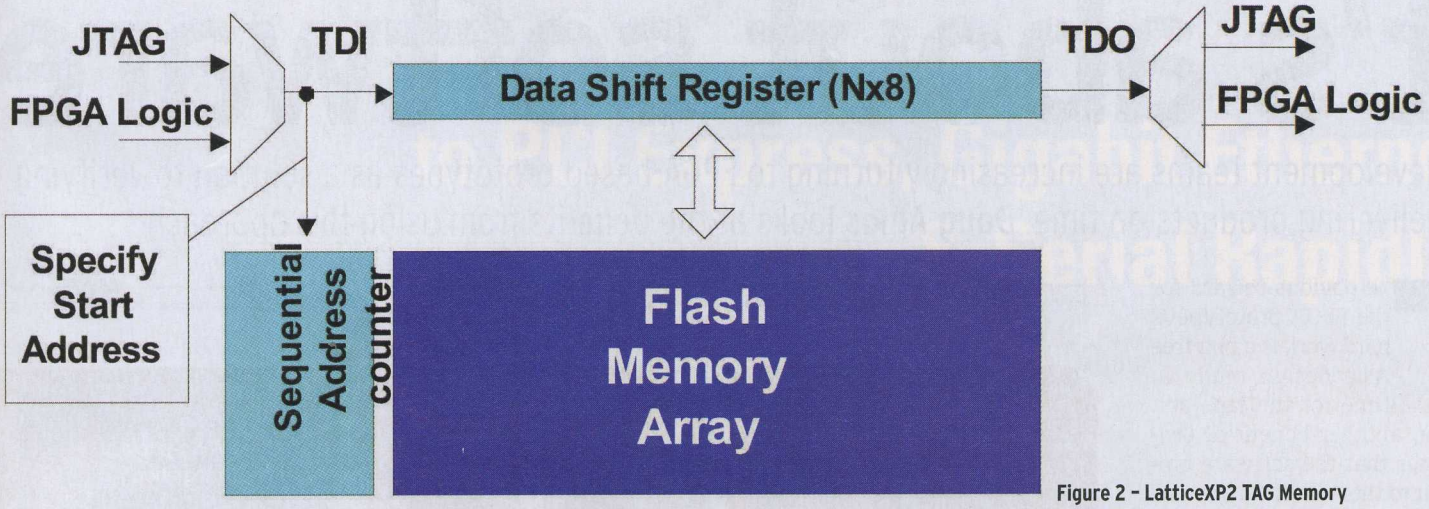

Figure 2 – LatticeXP2 TAG Memory

interface engine requires minimal logic. When using the SPI memory, care should be taken to avoid writing in the FPGA configuration space. This can be locked out through the design of the interface logic and/or through the use of sector locks that are found on many SPI flash memory devices.

Although SPI devices allow random reads for many applications, higher performance can be achieved by caching the current sector using the FPGA's block memories. As writing requires first that the sector be erased, this caching approach allows significantly more flexibility. With SPI memory, the erase/program cycles allowed normally exceed 100K.

This specification typically covers the number of power-up and power-down cycles that would occur during normal operation. However, if the number of memory writes is expected to exceed this amount, then memory caching can again be used to prevent the system from reaching the SPI endurance limit.

Non-volatile FPGA TAG Memory

Many non-volatile FPGAs incorporate small TAG memories within the device. For instance, the LatticeXP2 family of devices provides between 0.6 and 3.4kbit of memory for general-purpose use. This is the amount of memory often required to store equipment settings and calibration data. As shown in figure 2, the LatticeXP2 TAG memory can be accessed through the JTAG interface or on-chip logic. Details differ among FPGA vendors, so the designer should confirm that the desired approach is supported.

The TAG memory space is separate from the FPGA configuration, so there is no chance of accidentally overwriting the FPGA configuration during these operations. The on-chip TAG memory is ideal for integrating the small stand-alone serial EEPROMs that are found in many systems.

Flash Shadowed Block RAMs

These FPGA devices typically offer the capability to use the flash initialisation bits that shadow the block RAMs in the device. At device configuration time, flash memory bits initialise each of the block RAMs within the device. These memories can be written to and read just like any other SRAM. When it is necessary to store the configuration data within the FPGA, toggling the appropriate signal results in the current RAM values being written back into flash. This approach allows for the high speed reading and writing of RAM to be combined with the non-volatile capability of flash. Typically, thousands of erase and write cycles are supported for this operation. As with the TAG memory, the Flash associated with the block RAMs is logically separated, ensuring that the FPGA logic configuration is not accidentally overwritten.

	Initialized Block RAM	SPI Re-use	SPI Reuse with Block RAM Cache	TAG	Flash Shadowed Block RAM
Read/write	Read only	Read/Write	Read/Write	Read/Write	Read/Write
Burst Speed	300MB/s+	6MB/s	300MB/s+	10MB/s	300MB/s+
Sequential / Random	Random read Random write	Random read Sector write	Random read Random write	Sequential read Sector write	Random read Random write
Write Cycles	N/A	100K+	Infinite (within cache)	1K+	SRAM Infinite Flash 1K+

Choosing an Approach

As mentioned previously, ultimately it is the application requirements that drive the selection of the most appropriate technology. Each of the approaches detailed certainly has a place within system design. In summary, if the designer is looking for a moderate amount of ROM for code store or for use as a look-up table, then the initialised block RAM approach is likely the best. On the other hand, if there is a small amount of data, such as calibration information, current setting information or system identifier information, then TAG memory provides a convenient solution. For larger amounts of memory that need to be re-written, the SPI memory reuse or flash shadowed block RAM approaches are useful. Of course, no matter which approach is taken, board area and cost will be reduced.

So, review your requirements and choose your approach. You have nothing to lose except board area and cost.

Gordon Hands is director of strategic marketing for Lattice Semiconductor