DIGITAL DESIGN OF SERIAL FLASH MEMORY CONTROLLER WITH SPI INTERFACE FOR EMBEDDED SYSTEMS

by

CİHAN TUZCU

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of Master Science

Sabancı University
May 2004

DIGITAL DESIGN OF SERIAL FLASH MEMORY CONTROLLER WITH SPI INTERFACE FOR EMBEDDED SYSTEMS

APPROVED BY:

Assist. Prof. Dr. Ayhan BOZKURT ………………………….
(Thesis Advisor)

Assist. Prof. Dr. İlker HAMZAOĞLU ………………………….
(Jury Member)

Assist. Prof. Dr. Erkay SAVAŞ ………………………….
(Jury Member)

DATE OF APPROVAL: …………………………………

© Cihan TUZCU 2004

All Rights Reserved

ABSTRACT

This thesis presents digital design and implementation of a controller module for serial flash memories.

Firstly, the platform including the serial flash memory controller, flash memories and SPI (Serial Peripheral Interface) protocol have been investigated to solve the current problems related with controlling of serial flash memories. Then, in the implementation part of the thesis, the Serial Flash Memory Controller module has been designed by using VHDL (VHSIC Hardware Description Language-VHDL) and synthesized in CMOS 0.35 μm technology. Functional and gate-level simulations have been done with Cadence simulator. Lastly the final gate level netlist has been placed and routed with Cadence Silicon Ensemble.

A great deal of attention has been given to design a generic controller that needs simple software and minimum processor access cycle. It is programmed from the processor for different operations of serial flash memories. The structure of the frame, control data and timings are controlled by hardware according to the programmed operation. In addition to this, our Serial Flash Memory Controller module can be used with different flash memories. This is very important property for reusability of the module.

The Serial Flash Memory Controller module is capable to work up to 20 MHz serial communication speed and it can be integrated to processor platforms that have AMBA (Advanced Microcontroller Bus Architecture) APB (Advanced Peripheral Bus) interface.

ÖZET

Bu tez seri flaş belleklerin kontrolünü sağlayan devrenin sayısal olarak tasarımı, ve uygulanması aşamalarından oluşmuştur.

İlk olarak seri flaş bellek kontrolör bloğunun da içerisinde bulunduğu işlemci platformu, seri flaş bellekler ve SPI (Serial Peripheral Interface) protokolu, seri flaş belleklerin kontrolündeki mevcut problemler için araştırılmıştır. Tezin uygulama bölümünde seri flaş bellek kontrolör bloğu VHDL (VHSIC Hardware Description Language-VHDL) kullanılarak sayısal olarak tasarlanmış, 0.35 μm sayısal CMOS teknolojisi kullanılarak sentezlenmiş, fonksiyonel ve kapı seviyesinde test edilmiştir. Tezin son aşamasında, sentezlenmiş blok yerleştirme ve yol atama işlemlerinden geçirilmiştir.

Seri flaş bellek kontrolör bloğunun jenerik olarak tasarlamasının yanında bloğun basit bir yazilima ve minimum işlemci kontrolüne ihtiyaç duymasına büyük önem verilmiştir. Seri flaş belleğe transfer edilecek bilginin içeriği ve SPI (Serial Peripheral Interface) protokolüne uygun olarak gönderilmesi, seri flaş bellek kontrolör bloğu tarafından, işlemcinin programlandığı operasyona göre kontrol edilir. Seri flaş bellek kontrolör bloğu farklı seri flaş belleklerle kullanılabilir. Blok, işlemci tarafından seri flaş belleklerin farklı operasyonları için programlanabilir.

Seri flaş bellek kontrolör bloğunun maksimum 20 MHz seri transfer hızına kadar çıkabilmektedir. Blok AMBA (Advanced Microcontroller Bus Architecture) APB (Advanced Peripheral Bus) arayüzü bulunan işlemci platformlarına entegre edilebilir.

To my parents.

ACKNOWLEDGEMENTS

This research begun while I was working in ST Microelectronics Istanbul Design Center. Many, many people have helped me not to get lost during the development of this thesis.

First, I would like to thank my Assist. Prof. Dr. Ayhan BOZKURT who supervised and helped me so much. It was a great pleasure to me to conduct this thesis under his supervision. I also acknowledge Assoc. Prof. Dr. Yaşar GÜRBÜZ who as my second supervisor provided constructive comments during my thesis time.

I am very grateful to Alcatel Microelectronics and ST Microelectronics for providing financial support to study at Sabancı University as part of a university industry collaboration agreement.

My colleagues from ST Microelectronics supported me in my research work. I want to thank them for all their help, support, interest and valuable hints.

Finally, I am grateful to my parents for their continuous encouragement, abundant love and generous support they have given me throughout my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ADC	Analog to Digital Converter
AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
ASB	Advanced System Bus
ASIC	Application Specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
DAC	Digital to Analog Converter
EPROM	Erasable Programmable Read Only Memory
EEPROM	Electrically Erasable Programmable Read Only Memory
HDL	Hardware Description Language
LCD	Liquid Crystal Display
MOS	Metal Oxide Semiconductor
RAM	Random Access Memory
RTC	Real Time Clocks
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

DIGITAL DESIGN OF SERIAL FLASH MEMORY CONTROLLER WITH SPI INTERFACE FOR EMBEDDED SYSTEMS

by

CİHAN TUZCU

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of Master Science

Sabancı University
May 2004

DIGITAL DESIGN OF SERIAL FLASH MEMORY CONTROLLER WITH SPI INTERFACE FOR EMBEDDED SYSTEMS

APPROVED BY:

Assist. Prof. Dr. Ayhan BOZKURT (Thesis Advisor)	………………………….
Assist. Prof. Dr. İlker HAMZAOĞLU (Jury Member)	………………………….
Assist. Prof. Dr. Erkay SAVAŞ (Jury Member)	………………………….

DATE OF APPROVAL: …………………………………

© Cihan TUZCU 2004

All Rights Reserved

ABSTRACT

This thesis presents digital design and implementation of a controller module for serial flash memories.

Firstly, the platform including the serial flash memory controller, flash memories and SPI (Serial Peripheral Interface) protocol have been investigated to solve the current problems related with controlling of serial flash memories. Then, in the implementation part of the thesis, the Serial Flash Memory Controller module has been designed by using VHDL (VHSIC Hardware Description Language-VHDL) and synthesized in CMOS 0.35 μm technology. Functional and gate-level simulations have been done with Cadence simulator. Lastly the final gate level netlist has been placed and routed with Cadence Silicon Ensemble.

A great deal of attention has been given to design a generic controller that needs simple software and minimum processor access cycle. It is programmed from the processor for different operations of serial flash memories. The structure of the frame, control data and timings are controlled by hardware according to the programmed operation. In addition to this, our Serial Flash Memory Controller module can be used with different flash memories. This is very important property for reusability of the module.

The Serial Flash Memory Controller module is capable to work up to 20 MHz serial communication speed and it can be integrated to processor platforms that have AMBA (Advanced Microcontroller Bus Architecture) APB (Advanced Peripheral Bus) interface.

ÖZET

Bu tez seri flaş belleklerin kontrolünü sağlayan devrenin sayısal olarak tasarımı, ve uygulanması aşamalarından oluşmuştur.

İlk olarak seri flaş bellek kontrolör bloğunun da içerisinde bulunduğu işlemci platformu, seri flaş bellekler ve SPI (Serial Peripheral Interface) protokolu, seri flaş belleklerin kontrolündeki mevcut problemler için araştırılmıştır. Tezin uygulama bölümünde seri flaş bellek kontrolör bloğu VHDL (VHSIC Hardware Description Language-VHDL) kullanılarak sayısal olarak tasarlanmış, 0.35 μm sayısal CMOS teknolojisi kullanılarak sentezlenmiş, fonksiyonel ve kapı seviyesinde test edilmiştir. Tezin son aşamasında, sentezlenmiş blok yerleştirme ve yol atama işlemlerinden geçirilmiştir.

Seri flaş bellek kontrolör bloğunun jenerik olarak tasarlamasının yanında bloğun basit bir yazilima ve minimum işlemci kontrolüne ihtiyaç duymasına büyük önem verilmiştir. Seri flaş belleğe transfer edilecek bilginin içeriği ve SPI (Serial Peripheral Interface) protokolüne uygun olarak gönderilmesi, seri flaş bellek kontrolör bloğu tarafından, işlemcinin programlandığı operasyona göre kontrol edilir. Seri flaş bellek kontrolör bloğu farklı seri flaş belleklerle kullanılabilir. Blok, işlemci tarafından seri flaş belleklerin farklı operasyonları için programlanabilir.

Seri flaş bellek kontrolör bloğunun maksimum 20 MHz seri transfer hızına kadar çıkabilmektedir. Blok AMBA (Advanced Microcontroller Bus Architecture) APB (Advanced Peripheral Bus) arayüzü bulunan işlemci platformlarına entegre edilebilir.

To my parents.

ACKNOWLEDGEMENTS

This research begun while I was working in ST Microelectronics Istanbul Design Center. Many, many people have helped me not to get lost during the development of this thesis.

First, I would like to thank my Assist. Prof. Dr. Ayhan BOZKURT who supervised and helped me so much. It was a great pleasure to me to conduct this thesis under his supervision. I also acknowledge Assoc. Prof. Dr. Yaşar GÜRBÜZ who as my second supervisor provided constructive comments during my thesis time.

I am very grateful to Alcatel Microelectronics and ST Microelectronics for providing financial support to study at Sabancı University as part of a university industry collaboration agreement.

My colleagues from ST Microelectronics supported me in my research work. I want to thank them for all their help, support, interest and valuable hints.

Finally, I am grateful to my parents for their continuous encouragement, abundant love and generous support they have given me throughout my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ADC	Analog to Digital Converter
AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
ASB	Advanced System Bus
ASIC	Application Specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
DAC	Digital to Analog Converter
EPROM	Erasable Programmable Read Only Memory
EEPROM	Electrically Erasable Programmable Read Only Memory
HDL	Hardware Description Language
LCD	Liquid Crystal Display
MOS	Metal Oxide Semiconductor
RAM	Random Access Memory
RTC	Real Time Clocks
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

1. INTRODUCTION

1.1. Motivation

An increasing number of embedded applications use in-circuit reprogrammable memory chips. Embedded system designers have begun to use flash memory to hold a system's code and its data, replacing solutions that include a combination of EPROM, EEPROM, and/or flash. The flash memory included on chip allows data to be reloaded whenever required, ie after assembly of the hardware or even after shipment at customer’s site in case of a bug or new features. Therefore data can be loaded or modified with the microcontroller installed in the application system.

The evolution of computer and peripherals market segment is driven by the progress of semiconductor industry and in particular by the development of high-speed processors that demand higher capacity, higher data processing time and faster memories. Flash memories are ideal for the applications requiring fast code programming and fast code downloading.

Parallel Access and Serial Access Parallel buses were primarily used to interface flash memories with microcontrollers and microprocessors through an address bus, a data bus and a control bus. By default, the term "Flash memory" refers to a parallel interface memory. The data bus can be organized as x8 bits, x16 bits or x32 bits. In some cases, address and data buses can be multiplexed.

The serial bus is used to connect a Flash memory to a microcontroller or an ASIC equipped with a serial bus. Serial buses are input/output interfaces supporting a mixed address/data protocol. The serial bus connectivity reduces the number of interface signals required. For example, the SPI bus, the most popular serial bus for serial Flash, memories, requires only 4 signals (data in, data out, clock and chip select) compared to 21 signals necessary to interface a 10-bit address parallel memory. As a result, the number of pins of the memory package (memory and bus master) is reduced, as is the number of PCB tracks. Consequently, a serial memory can fit into a smaller and less expensive package. However, serial Flash memories are available in lower densities than Flash memories. The communication throughput between serial Flash memory and master processor is lower than the communication throughput between Flash memory and master processor. Consequently, the time to download code into the serial memory and execute it from the memory is longer. As a result, serial Flash memories are usually used for small code storage associated with a cache RAM. This is called a code shadowing architecture. The executable code is first programmed in the memory and it is write protected. After power-up, it is downloaded from memory to RAM from where it is executed by the master processor.

The main purpose of this thesis is to design a controller for serial flash memories to be used in embedded applications. Moreover, it has been tried to design the whole IP architecture as generic as possible. So that it can be used with different serial flash memories.

1.2. Thesis Organization

The goal of this thesis is to design a controller for fast and reliable communication between the processor and the serial flash memories in embedded applications.

In order to understand where the serial flash memory controller module is used and what the complete system looks like, we need to have a closer look to the structure of a processor platform. Chapter 2 gives an introductory knowledge about on-chip communications standard for designing high-performance embedded microcontrollers.

Chapter 3 gives an overview of flash memories. Firstly the structures of different types of flash memories are explained and then parallel and serial interfaces for flash memories are compared.

Serial Flash memories have serial peripheral interface to transfer data. Chapter 4 covers the serial peripheral interface protocol. Firstly the structure and the functionality of SPI is introduced. Then the configuration parameters for serial peripheral interface are explained. Lastly where SPI is used in practice is covered in this chapter.

Chapter 5 contains detailed explanation of serial flash memory controller module. After introducing the properties of controller module, detailed functional descriptions for each of subblocks are given. Then upload and download timings for different flash memories are calculated. Also the comparison made between our serial flash memory controller module and several different SPI master modules. The configuration of serial flash memory controller module is also given in this part.

Chapter 6 covers the test scenarios, which we used to verify our design, for different flash memories.

Synthesis results in terms of area and power consumption estimations for serial flash memory controller are covered in chapter 7.

The results for place and route of serial flash memory controller is given in chapter 8.

Chapter 9 includes the details for FPGA implementation of serial flash memory controller. FPGA board used for this application and the test environment are explained in this chapter.

Finally, some conclusions are drawn for the overall assessment of the study and some possible future research topics are pointed in chapter 10.

2. PROCESSOR PLATFORM

The hardware, including the Serial Flash Controller, is a clean and stable processor platform that can be reused in different product lines. The kernel of the platform is the AMBA AHB and APB bus system.

The platform can be based on any processor core, which has a direct AHB interface or an appropriate wrapper for the AHB bus (ARM7 and ARM9 family).

The overview of the hardware platform is illustrated in Figure 2-1.

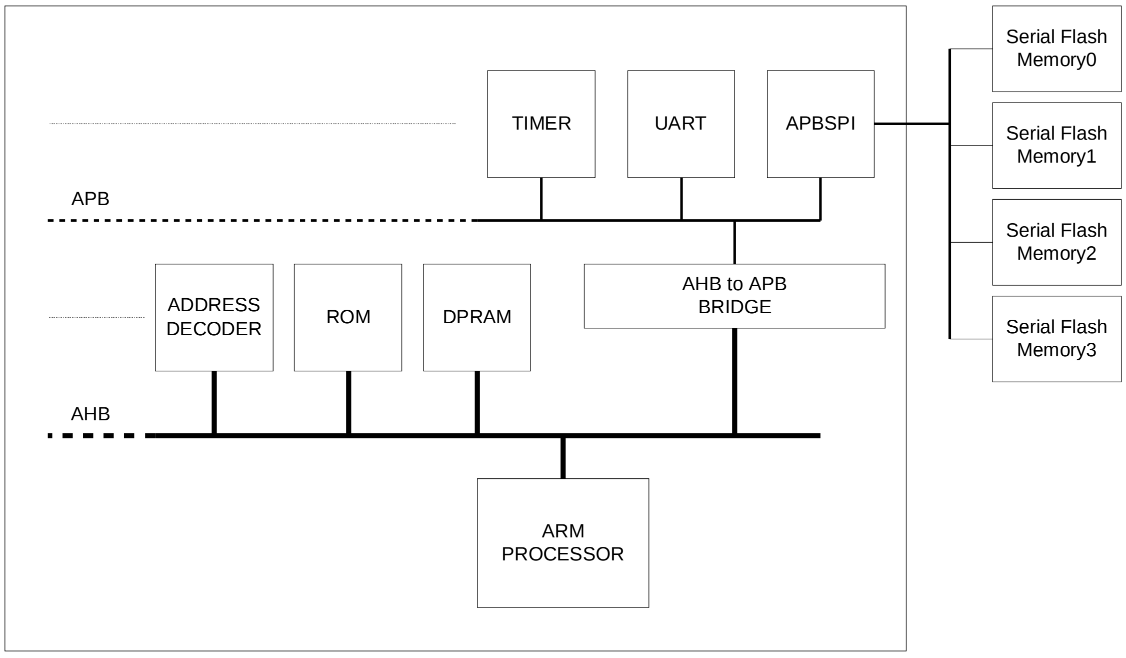


Figure 2-1 Overview of processor platform

The AMBA AHB is a multimaster multislave high-speed bus that is controlled via an Arbiter and an Address Decoder. The AMBA APB bus is a single master multislave low speed bus. Both busses, AHB and APB, are accessible from outside.

Processor platform includes a bridge from the AHB to the APB bus that is slave on the AHB and a master on the APB side.

2.1. Introduction to the AMBA Buses

The Advanced Microcontroller Bus Architecture (AMBA) specification defines an on-chip communications standard for designing high-performance embedded microcontrollers [2].

Two distinct buses are defined within the AMBA specification:

• the Advanced High-performance Bus (AHB)

• the Advanced Peripheral Bus (APB)

The AMBA AHB is for high-performance, high clock frequency system modules. The AHB acts as the high-performance system backbone bus. AHB supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral macrocell functions. AHB is also specified to ensure ease of use in an efficient design flow using synthesis and automated test techniques.

The AMBA APB is for low-power peripherals. AMBA APB is optimised for minimal power consumption and reduced interface complexity to support peripheral functions. APB can be used in conjunction with either version of the system bus.

2.2. AMBA AHB

AHB is a new generation of AMBA bus, which is intended to address the requirements of high-performance synthesizable designs. AMBA AHB is a new level of bus, which sits above the APB, and implements the features required for high-performance, high clock frequency systems including:

• burst transfers

• split transactions

• single cycle bus master handover

• single clock edge operation

• non-tristate implementation

• wider data bus configurations (64/128 bits).

Table 2-1 contains an overview of the AMBA AHB signals.

Name	Description
HCLK	Bus clock: This clock times all bus transfers. All signal timings are related to the rising edge of HCLK.
HRESETn	Reset: The bus reset signal is active LOW and is used to reset the system and the bus. This is the only active LOW signal.
HADDR[31:0]	Address bus: The 32-bit system address bus.
HTRANS[1:0]	Transfer type: Indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY.
HWRITE	Transfer direction: When HIGH this signal indicates a write transfer and when LOW a read transfer.
HSIZE[2:0]	Transfer size: Indicates the size of the transfer, which is typically byte (8-bit), halfword (16-bit) or word (32-bit). The protocol allows for larger transfer sizes up to a maximum of 1024 bits.
HBURST[2:0]	Burst type: Indicates if the transfer forms part of a burst. Four, eight and sixteen beat bursts are supported and the burst may be either incrementing or wrapping.
HPROT[3:0]	Protection control: The protection control signals provide additional information about a bus access and are primarily intended for use by any module that wishes to implement some level of protection. The signals indicate if the transfer is an opcode fetch or data access, as well as if the transfer is a privileged mode access or user mode access. For bus masters with a memory management unit these signals also indicate whether the current access is cacheable or bufferable.
HWDATA[31:0]	Write data bus: The write data bus is used to transfer data from the master to the bus slaves during write operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
HSELx	Slave select: Each AHB slave has its own slave select signal and this signal indicates that the current transfer is intended for the selected slave. This signal is simply a combinatorial decode of the address bus.
HRDATA[31:0]	Read data bus: The read data bus is used to transfer data from bus

	slaves to the bus master during read operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
HREADY	Transfer done: When HIGH the HREADY signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer.
HRESP[1:0]	Transfer response: The transfer response provides additional information on the status of a transfer. Four different responses are provided, OKAY, ERROR, RETRY and SPLIT.

Table 2-1 AMBA AHB signal definitions

2.2.1. Bus Interconnection

The AMBA AHB bus protocol is designed to be used with a central multiplexer interconnection scheme. Using this scheme all bus masters drive out the address and control signals indicating the transfer they wish to perform and the arbiter determines which master has its address and control signals routed to all of the slaves. A central decoder is also required to control the read data and response signal multiplexer, which selects the appropriate signals from the slave that is involved in the transfer.

Figure 2-2 illustrates the structure required to implement an AMBA AHB design with three masters and four slaves.

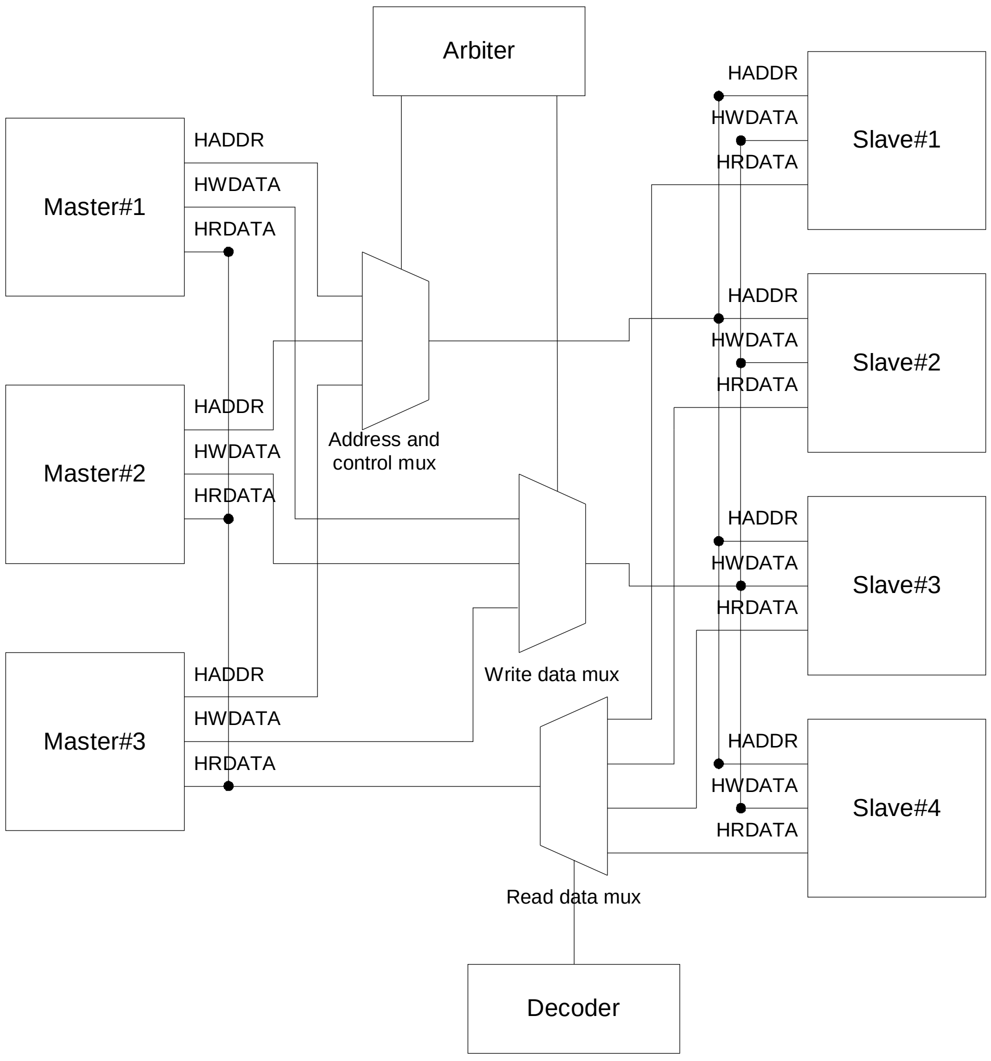


Figure 2-2 Multiplexer Interconnection

2.2.2. Overview of AMBA AHB operation

Before an AMBA AHB transfer can commence, the bus master must be granted access to the bus. The master starts this process by asserting a request signal to the arbiter. Then the arbiter indicates when the master will be granted use of the bus.

A granted bus master starts an AMBA AHB transfer by driving the address and control signals. These signals provide information on the address, direction and width of the transfer, as well as an indication if the transfer forms part of a burst. Two different forms of burst transfers are allowed:

• incrementing bursts, which do not wrap at address boundaries

• wrapping bursts, which wrap at particular address boundaries.

A write data bus is used to move data from the master to a slave, while a read data bus is used to move data from a slave to the master.

Every transfer consists of:

• an address and control cycle

• one or more cycles for the data.

The address cannot be extended and therefore all slaves must sample the address during this time. The data, however, can be extended using the HREADY signal. When LOW this signal causes wait states to be inserted into the transfer and allows extra time for the slave to provide or sample data.

During a transfer the slave shows the status using the response signals, HRESP[1:0]:

• OKAY

The OKAY response is used to indicate that the transfer is progressing normally and when HREADY goes HIGH, the transfer has completed successfully.

• ERROR

The ERROR response indicates that a transfer error has occurred and the transfer has been unsuccessful.

• RETRY and SPLIT

Both the RETRY and SPLIT transfer responses indicate that the transfer cannot complete immediately, but the bus master should continue to attempt the transfer.

In normal operation a master is allowed to complete all the transfers in a particular burst before the arbiter grants another master access to the bus. However, in order to avoid excessive arbitration latencies it is possible for the arbiter to break up a burst and in such cases the master must re-arbitrate for the bus in order to complete the remaining transfers in the burst.

2.3. AMBA APB

The APB is part of the AMBA hierarchy of buses and is optimised for minimal power consumption and reduced interface complexity. The AMBA APB appears as a local secondary bus that is encapsulated as a single AHB slave device. APB provides a low-power extension to the system bus, which builds on AHB signals directly. The APB bridge appears as a slave module, which handles the bus handshake and control signal retiming on behalf of the local peripheral bus. The AMBA APB should be used to interface to any peripherals, which are low bandwidth and do not require the high performance of a pipelined bus interface. All signal transitions are only related to the rising edge of the clock.

An AMBA APB implementation typically contains a single APB bridge, which is required to convert AHB transfers into a suitable format for the slave devices on the APB. The bridge provides latching of all address, data and control signals, as well as providing a second level of decoding to generate slave select signals for the APB peripherals.

The APB bus is characterized by:

- a simple bus unpipelined architecture
- easy to implement with all the peripherals acting as slaves
- low gate count
- low power
- reduced loading of the main system bus by isolating peripherals behind the bridge
- peripheral bus signals only active during low bandwidth peripheral transfers.

A simple APB interface is recommended for:

- simple register-mapped slave devices
- very low power interfaces where clocks cannot be globally routed
- grouping narrow-bus peripherals to avoid loading the system bus.

AMBA APB signal names with a description is given in Table 2-2.

Name	Description
PCLK	Bus clock: The rising edge of PCLK is used to time all transfers on the APB.
PRESETn	APB reset: The APB bus reset signal is active LOW and this signal will normally be connected directly to the system bus reset signal.
PADDR[31:0]	APB address bus: This is the APB address bus, which may be up to 32-bits wide and is driven by the peripheral bus bridge unit.
PSELx	APB select: A signal from the secondary decoder, within the peripheral bus bridge unit, to each peripheral bus slave x. This signal indicates that the slave device is selected and a data transfer is required. There is a PSELx signal for each bus slave.
PENABLE	APB strobe: This strobe signal is used to time all accesses on the peripheral bus. The enable signal is used to indicate the second cycle of an APB transfer. The rising edge of PENABLE occurs in the middle of the APB transfer.
PWRITE	APB transfer direction: When HIGH this signal indicates an APB write access and when LOW a read access.
PRDATA	APB read data bus: The read data bus is driven by the selected slave during read cycles (when PWRITE is LOW). The read data bus can be up to 32-bits wide.
PWDATA	APB write data bus: The write data bus is driven by the peripheral bus bridge unit during write cycles (when PWRITE is HIGH). The write data bus can be up to 32-bits wide.

Table 2-2 AMBA APB signal definitions

2.3.1. AMBA APB States

Figure 2-3 shows AMBA APB state diagram.

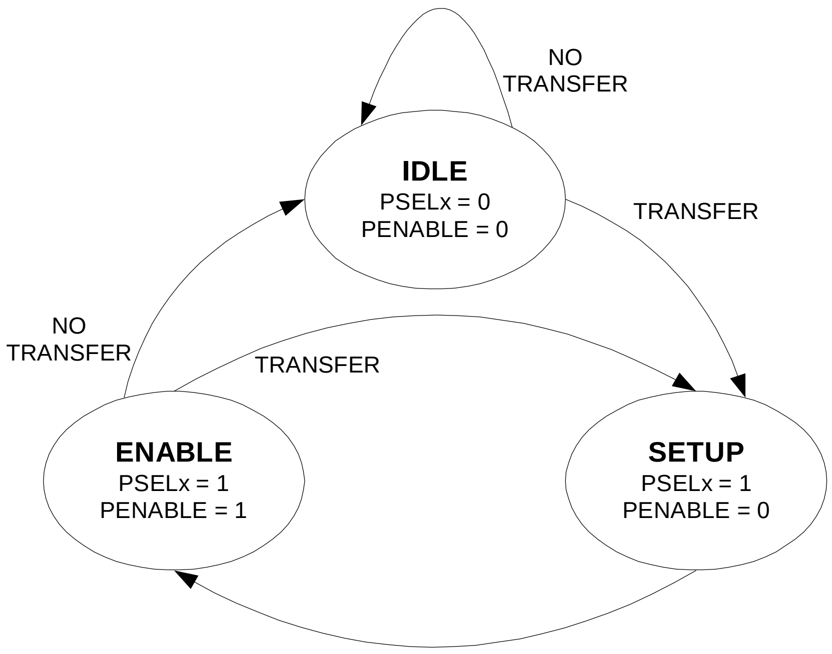


Figure 2-3 State Diagram

IDLE state is the default state for the peripheral bus.

When a transfer is required the bus moves into the SETUP state, where the appropriate select signal, PSELx, is asserted. The bus only remains in the SETUP state for one clock cycle and will always move to the ENABLE state on the next rising edge of the clock.

In the ENABLE state the enable signal, PENABLE is asserted. The address, write and select signals all remain stable during the transition from the SETUP to ENABLE state. The ENABLE state also only lasts for a single clock cycle and after this state the bus will return to the IDLE state if no further transfers are required. Alternatively, if another transfer is to follow then the bus will move directly to the SETUP state. It is acceptable for the address, write and select signals to glitch during a transition from the ENABLE to SETUP states.

2.3.1.1.Write Transfer

The waveform of write transfer is shown in Figure 2-4.

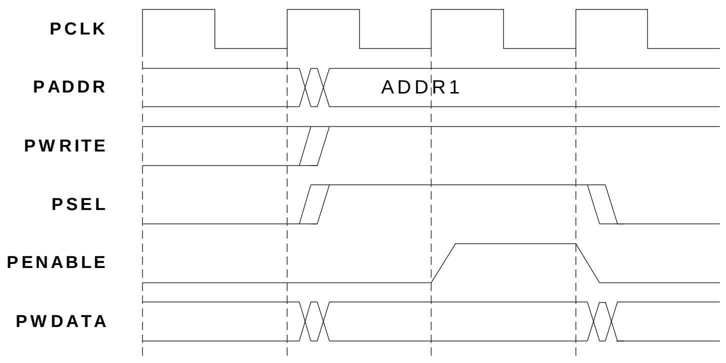


Figure 2-4 Write transfer

The write transfer starts with the address, write data, write signal and select signal all changing after the rising edge of the clock. The first clock cycle of the transfer is called the SETUP cycle. After the following clock edge the enable signal PENABLE is asserted, and this indicates that the ENABLE cycle is taking place. The address, data and control signals all remain valid throughout the ENABLE cycle. The transfer completes at the end of this cycle.

The enable signal, PENABLE, will be deasserted at the end of the transfer. The select signal will also go LOW, unless the transfer is to be immediately followed by another transfer to the same peripheral.

In order to reduce power consumption the address signal and the write signal will not change after a transfer until the next access occurs.

2.3.1.2.Read Transfer

The waveform of read transfer is shown in Figure 2-5.

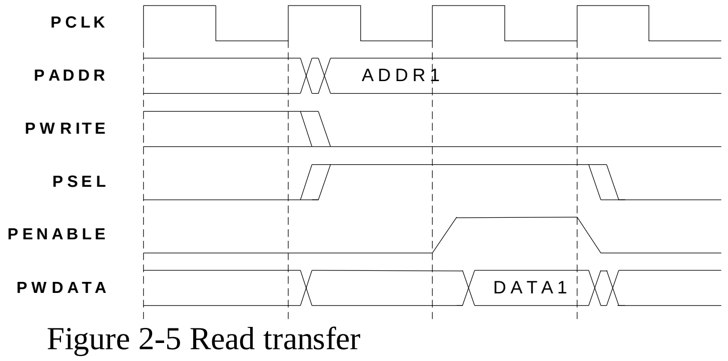


Figure 2-5 Read transfer

The timing of the address, write, select and strobe signals are all the same as for the write transfer. In the case of a read, the slave must provide the data during the ENABLE cycle. The data is sampled on the rising edge of clock at the end of the ENABLE cycle.

2.4. Typical AMBA Based Microcontroller

An AMBA-based microcontroller typically consists of a high-performance system backbone bus (AMBA AHB or AMBA ASB), able to sustain the external memory bandwidth, on which the CPU, on-chip memory and other Direct Memory Access (DMA) devices reside. This bus provides a high-bandwidth interface between the elements that are involved in the majority of transfers. Also there is a bridge to the lower bandwidth APB, located on the high-performance bus. Most of the peripheral devices in the system are located on the APB. Figure 2-6 shows the structure of typical AMBA bus system.

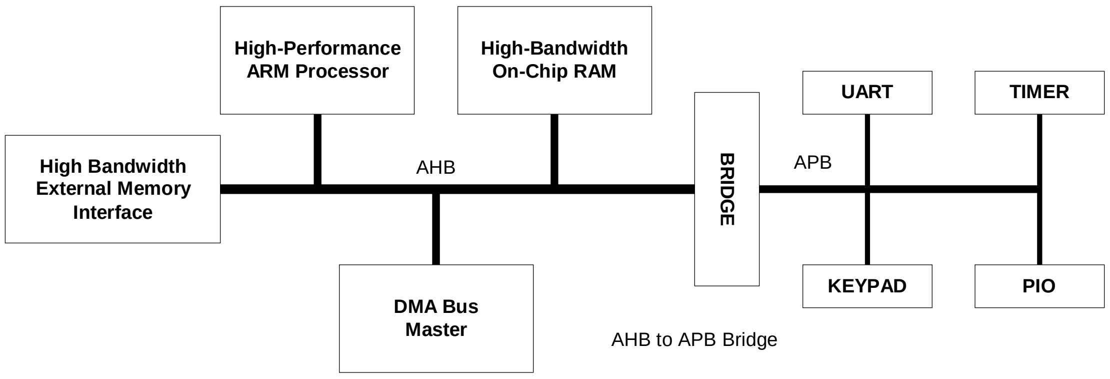

High-Performance ARM Processor
High-Bandwidth On-Chip RAM
UART
TIMER
High Bandwidth External Memory Interface
AHB
BRIDGE
APB
KEYPAD
PIO
DMA Bus Master
AHB to APB Bridge

Figure 2-6 Typical AMBA Bus System

3. FLASH MEMORIES

3.1. Flash Memory

Flash memory is a type of electronic memory increasingly used in a wide range of communications, consumer, computer and peripherals, and automotive applications, but which relatively few semiconductor companies can produce in volume at the low cost equipment manufacturers require.

Flash belongs to the class of semiconductor memories called non-volatile memories, of which it is the most dynamic driving force. Semiconductor memories can be divided into two different types: those that can only retain data stored in them while they are connected to a battery or some other source of electrical power (volatile), and those that retain their data even if their power supply is removed (non-volatile).

Flash memories can be electrically erased and it is not necessary to erase the whole memory array in order to store new data in part of it.

Flash memory, EPROM and EEPROM devices all use the same basic floating gate mechanism to store data, but they use different techniques for reading and writing data. In each case, the basic memory cell consists of a single MOS transistor (MOSFET) with two gates:

- control gate connected to the read/write control circuitry
- floating gate located between the control gate and the channel of the MOSFET (the part of the MOSFET through which electrons flow between the so-called Source and Drain terminals).

In a standard MOSFET, a single Gate terminal controls the electrical resistance of the channel: electrical voltage applied to the gate controls how much current can flow between the Source and Drain. The MOSFETs used in non-volatile memories include a second gate that is completely surrounded by an insulating layer of silicon dioxide, i.e., it is electrically isolated from the rest of the circuitry. Because the floating gate is physically very close to the MOSFET channel, even a small electric charge has an easily detectable effect on the electrical behavior of the transistor. By applying appropriate signals to the control gate and measuring the change in transistor behavior, it is possible to determine whether there is an electrical charge on the floating gate. Because the floating gate is electrically isolated from the rest of the transistor, special techniques are required to move electrons to and from the floating gate.

One method is to fill the MOSFET channel with high-energy electrons by making a relatively high current pass between the drain and the source of the MOSFET. Some of these "hot" electrons have sufficient energy to cross the potential barrier between the channels and reach the floating gate. When the high current in the channel is removed, these electrons remain trapped in the floating gate. This is the method used to program the memory cells in EPROM and Flash memories. This technique, known as Channel Hot Electron (CHE) injection, can be used to load an electrical charge onto the floating gate, but does not provide a way to discharge it. EPROM technology achieves this by flooding the entire memory array with ultra-violet light; the high-energy light rays penetrate the chip structure and impart enough energy to the trapped electrons to allow them to escape from the floating gate.

The second method of moving a charge to a floating gate is the quantum mechanical effect known as tunneling. In this method electrons are removed from the floating gate by applying a voltage that is large enough to cause electrons to 'tunnel' across the insulating oxide layer to the source between the MOSFET control gate and the source or the drain. The number of electrons that can tunnel across an insulating layer in a given time depends on the thickness of the layer and the value of the applied voltage. To meet realistic voltage levels and erase-time constraints, the insulating layer must be very thin, typically 7nm (70 Angstroms).

EEPROM memories use tunneling to charge and discharge the floating gate according to the polarity of the applied tunneling voltage. A Flash memory can therefore be considered to be a memory device that is programmed like an EPROM and erased like an EEPROM, although there is much more to Flash technology than simply grafting the EEPROM erase mechanism onto EPROM technology.

The most important difference between EPROM and the other two processes lies in the thickness of the oxide layer that separates the floating gate from the source. In an EPROM, this is typically 20-25nm, but this is far too thick to allow tunneling to take place at an acceptable rate with a practical voltage level. For Flash memory, tunnel oxide thickness of around 10nm is required, and the quality of this oxide layer has a dramatic effect on the performance and reliability of the device. This is one of the reasons that relatively few semiconductor manufacturers have mastered Flash technology and even fewer have been able to reliably combine Flash technology and mainstream CMOS processes to build products such as microcontrollers with embedded Flash memory.

3.1.1. NAND and NOR Flash Memories

Although all flash memories use the same basic storage cell, there are a number of ways in which the cells can be interconnected within the overall memory array. The two most prominent architectures are known as NOR and NAND; these terms, derived from traditional combinatorial logic, indicate the topology of the array and the manner in which individual cells are accessed for reading and writing. Initially, there was a basic distinction between these two fundamentally different architectures, with NOR devices exhibiting inherently faster read times and NAND devices offering higher storage densities (because the NAND cell is about 40% smaller than the NOR cell). NOR Flash memories are considered to be the best choice for densities up to 256 Mbits, while NAND types are preferred for 512-Mbits and up. This is the best compromise between large data storage capacities and cell size - and consequently, final die size.

3.1.1.1. NOR Flash Memory

NOR-type Flash memories are based on technologies that evolved largely from the first non-volatile memory technologies. They are typically organized as a number of blocks between 16 Kbytes and 128 Kbytes, each of which can be individually erased or programmed. The architecture can be either uniform if all of the blocks are the same size or asymmetrical when the blocks vary in size. The array can be organized as a single piece of memory or split into dual or multiple banks, and in some cases, one block (called boot block) located at the top or the bottom of the address space, is dedicated to the storage of the boot code. NOR Flash memories usually have a random access for reading at byte/word level and sometimes a page access mode, allowing the reader to view an entire page of 2 to 4 words in one go. When very rapid read operations are required, the Flash memory is equipped with a burst read mode, which allows data to be transferred on every clock cycle.

3.1.1.2.NAND Flash Memory

Flash Memories can also be organized in NAND arrays by connecting cells in series. They feature parallel interface, higher storage densities (up to 1 Gbit), faster erase time, and slower random access time compared with the NOR type. For these reasons, they are used for storing large amounts of data, such as music files and digital images, handled by digital consumer applications. NAND Flash memories are organized into small blocks of 8 Kbytes to 16 Kbytes. Each block is divided into pages, usually 512-Bytes long, which can be read and programmed as a whole. This organization perfectly fits the data format widely used by mass storage systems such as floppy disks and hard disks. NAND flash memories are not suitable for direct code execution because of their slow access times, even if some of the last generation microcontrollers equipped with integrated cache memory can use NAND Flash to manage code and data storage. In addition, standard NAND Flash memories have a multiplexed data/address bus that reduces the device pin count and enables density upgrades within a single footprint.

3.1.2. Parallel and Serial Interface

Parallel Access and Serial Access Parallel buses were primarily used to interface flash memories with microcontrollers and microprocessors through an address bus, a data bus and a control bus. By default, the term "Flash memory" refers to a parallel interface memory. The data bus can be organized as x8 bits, x16 bits or x32 bits. In some cases, address and data buses can be multiplexed. They are available in densities of up to 128 Mbits. Because of their rapid read times, Flash memories are traditionally used for basic code or code-plus-parameter storage where greater flexibility compared to EPROM is more important than the additional unit cost. More recently, they have pervaded many new applications where their key functions are to store both code and data. This was achieved by dual operations supported by dual or multiple bank architecture, which enable programming/erasing operations in one bank while reading from another bank.

The serial bus is used to connect a Flash memory to a microcontroller or an ASIC equipped with a serial bus. Serial buses are input/output interfaces supporting a mixed address/data protocol. The serial bus connectivity reduces the number of interface signals required. For example, the SPI bus, the most popular serial bus for serial Flash memories, requires only 4 signals (data in, data out, clock and chip select) compared to 21 signals necessary to interface a 10-bit address parallel memory. As a result, the number of pins of the memory package (memory and bus master) is reduced, as is the number of PCB tracks. Consequently, a serial memory can fit into a smaller and less expensive package. However, serial Flash memories are available in lower densities than Flash memories. The communication throughput between serial Flash memory and master processor is lower than for traditional Flash memories. Consequently, the time to download code into the serial memory and execute it from the memory is longer. As a result, serial Flash memories are usually used for small code storage associated with a cache RAM. This is called a code shadowing architecture. The executable code is first programmed in the memory and it is write protected. After power-up, it is downloaded from memory to RAM from where it is executed by the master processor.

4. SPI PROTOCOL

The Serial Peripheral Interface (SPI) is a synchronous, serial data link that is standard across microprocessors, microcontrollers and peripherals. It enables communication between microprocessors, peripherals and inter-processor communication, and is widely used to connect peripherals to each other and to microprocessors.

4.1. SPI Protocol Signal Definition

The interface uses a 3-wire bus plus a chip/slave select line for each device connected to the bus. The three bus lines are as follows:

- SCLK - the clock signal used for synchronizing data transfers. It is generated by the bus "Master"
- MISO - Master In Slave Out. Line used for sending data from a slave to the master.
- MOSI - Master Out Slave In. Line used for sending data from the master to a slave.

4.2. SPI Functionality

Each device connected to the bus can be selected by the bus master using a dedicated SS (Slave Select) line. It is possible to have more than one master hanging off the bus, but only one master can be active at any given time. The implication of this configuration is that the bus master has to have as many lines as there are devices to drive each of the SS lines.

When the master initiates a data transfer, the master writes a bit to the MOSI line and reads a bit from the MISO at the same time on every cycle of the SCLK signal. The data is transferred through a simple shift register transfer scheme where the data is clocked into and out of devices on a first-in, first-out basis. This means that every data transfer results in an exchange of bits between the master and the slave (each device is simultaneously a transmitter and a receiver), making it a full duplex serial interface. When a device is not selected, it must tri-state (release) the output (MISO) line. Through buffering, it would be possible to drive more than one receive-only device, but not more than one transmit-only or receive and transmit device since there would be a contention issue on the MISO line.

The block diagram of this process is shown in Figure 4-1.

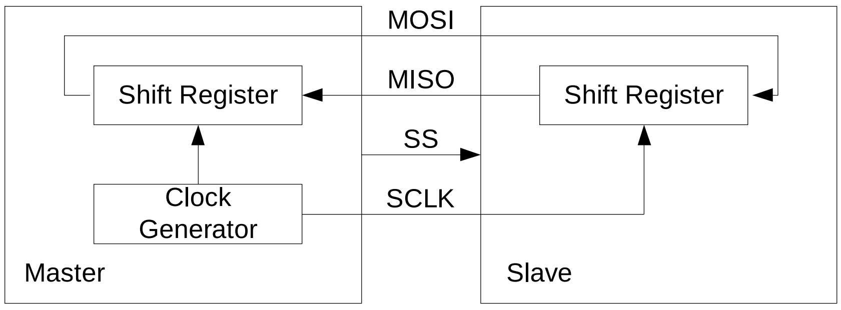


Figure 4-1 SPI Process

Usually, in synchronous serial protocols, data is clocked out on one edge and clocked in on the other edge to reduce clock skew errors.

4.3. SPI Configuration

Because there is no official specification, what exactly SPI is and what not, it is necessary to consult the data sheets of the components one wants to use. Important are the permitted clock frequencies and the type of valid transitions.

There are no general rules for transitions where data should be latched. Although not specified, in practice four modes are used. These four modes are the combinations of clock polarity and clock phase. In Table 4-1, the four modes are listed.

SPI Mode	Clock Polarity	Clock Phase
0	0	0
1	0	1
2	1	0
3	1	1

Table 4-1 SPI Modes

If the phase of the clock is zero, data is latched at the rising edge of the clock with polarity of the clock equals zero, and at the falling edge of the clock with polarity of the clock equals one. If the phase of the clock is one, the polarities are reversed. Polarity of the clock equals zero means falling edge and polarity of the clock equals one means rising edge.

The waveforms are shown in Figure 4-2.

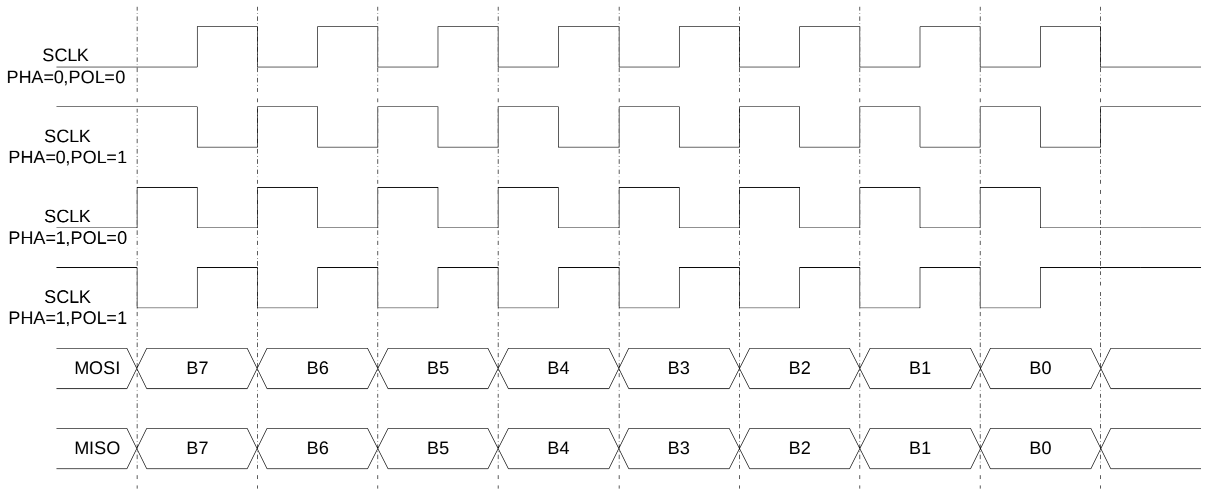


Figure 4-2 SPI clocking waveforms

4.4. Peripheral Types

Peripheral types that can be connected to the host processor through the SPI interface can be subdivided into the following categories:

- Converters (ADC and DAC)

- Memories (EEPROM and FLASH)
- Real Time Clocks (RTC)
- Sensors (temperature, pressure)
- Others (signalmixer, potentiometer, LCD controller, UART, USB controller, amplifier)

In the three categories, converters, memories and RTCs, there is a great variety of components. Devices belonging to the last two groups are more rarely.

There are lots of converters with different resolutions, clock frequencies and number of channels to choose from. (8, 10, 12 up to 24Bit with clock frequencies from 30ksps up to 600ksps).

Memory devices are mostly EEPROM variants. There are also a few SPI flash memories. Capacities range from a couple of bits up to 64KBit. Clock frequencies up to 3MHz. Serial EEPROMS SPI are available for different supply voltages (2.7V to 5V) allowing their use in low-voltage applications. The data retention time duration from 10 years to 100 years. The permitted number of write accesses is 1 million cycles for most components. By cascading memory devices any number of bits/word can be obtained.

RTCs are ideally suited for serial communication because only small amounts of data have to be transferred. There is also a great variety of RTCs with supply voltages from 2.0V. In addition to the standard functions of a "normal" clock, some RTCs offer an alarm function, non-volatile RAM etc.

The group of the sensors is yet weakly represented. Only a temperature and a pressure sensor could be found.

USB controllers with SPI make it easier to use these protocols on a micro controller and interfacing a LCD via SPI saves the troublesome parallel wiring.

5. SERIAL FLASH MEMORY CONTROLLER

Serial Flash Memory Controller module provides four-wire synchronous, serial communication with peripheral devices. It has the following features:

- Programmable through the AMBA APB bus by a host CPU.
- Master operation.
- Common data clock to receive and transmit data.
- Receive FIFO.
- SPI operation completed and FIFO overrun interrupts.
- Four external peripheral selects.

The Serial Flash Memory Controller module is split into four blocks:

- APB Interface
- SPI Interface
- SPIClockLogic
- RxFIFO

Block diagram of SPI block is depicted in Figure 5-1.

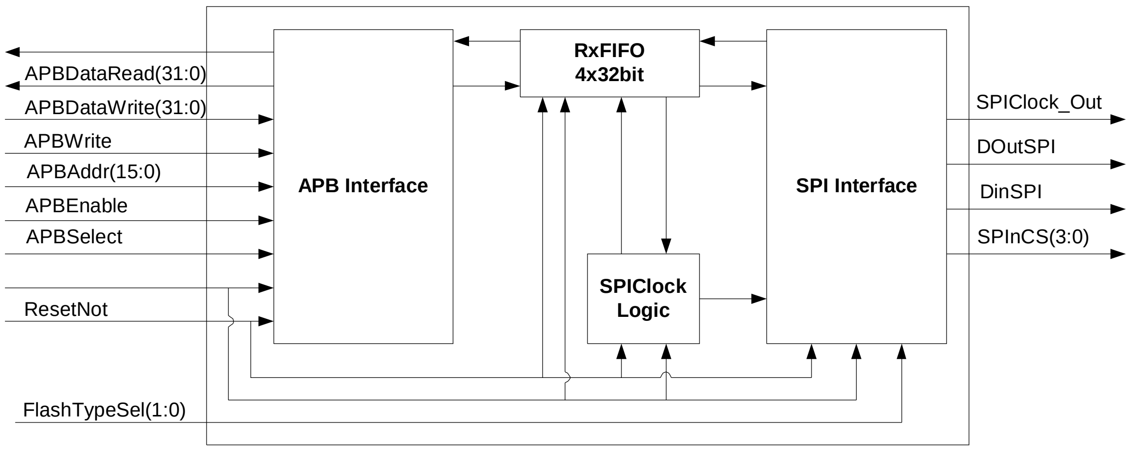


Figure 5-1Block Diagram of Serial Flash Controller Module

5.1. General Functional Description

5.1.1. APBIf

This block ensures the APB bus interfacing. It provides communication with the controller.

5.1.1.1. Interfaces

Name	Size	Direction	Description
APBClock	1	I	Clock
APBResetNot	1	I	Reset
APB Interface			
APBSelect	1	I	Chip select for APBSPI module
APBEnable	1	I	Enables APBSPI operation
APBAddr	16	I	Address input to APBSPI module
APBWrite	1	I	Write/Read enable input
APBDataWrite	32	I	Data input to APBSPI module
APBDataRead	32	O	Data output from APBSPI module
Intr	1	O	Interrupt output
SPIIf Interface			
WriteDataReg	32	O	Data output to SPIIf
ReadLengthReg	10	O	Length of the data that is read from the flash
AddrReg	32	O	Flash memory address of the data
SPInCSReg	2	O	Chip select output to SPIIf
StartRead_sig	1	O	Enable signal to send read opcode
GetPage_sig	1	O	Enable signal to send GetPage opcode
WriteData_sig	1	O	Enable signal to send WriteData opcode
WritePage_sig	1	O	Enable signal to send WritePage opcode
GetFlashStatus_sig	1	O	Enable signal to send GetFlashStatus opcode

WriteEnable_sig	1	O	Enable signal to send WriteEnable opcode
SectorErase_sig	1	O	Enable signal to send SectorErase opcode
BulkErase_sig	1	O	Enable signal to send BulkErase opcode
StartRead_captured	1	I	Indicates that the Read operation is started
GetPage_captured	1	I	Indicates that the GetPage operation is started
WriteData_captured	1	I	Indicates that the WriteData operation is started
WritePage_captured	1	I	Indicates that the WritePage operation is started
GetFlashStatus_captured	1	I	Indicates that the GetFlashStatus operation is started
WriteEnable_captured	1	I	Indicates that the WriteEnable operation is started
SectorErase_captured	1	I	Indicates that the SectorErase operation is started
BulkErase_captured	1	I	Indicates that the BulkErase operation is started
Busy	1	I	Indicates that SPIIf is busy
RxFIFO Interface			
Data_in	32	I	Data input from RxFIFO
FIFORE	1	O	Read Enable output to RxFIFO
EmptyFlag	1	I	FifoEmpty input from RxFIFO
FullFlag	1	I	FifoFull input from RxFIFO

Table 5-1 List of I/O Interfaces for APBIf Block

5.1.1.2.Detailed Functional Description

APBIf is synchronized with the rising edge of the APBClock input. The registers inside APBIf block can be read or written via APB bus by looking at the values of the APBSelect, APBWrite and APBAddr. If the block is selected for write operation by the controller, the corresponding register will be written. If it is selected for read operation by the controller, the corresponding register would be read. APBAddr input will be the address of the register.

The registers inside APBIf block are shown in Table 5-2.

Register Name	Address	Reset Condition (MSB to LSB)	# bits to write/ read	Description
CTRLREG	0x1800e000	00000000	W: 6	Bit: 0: selects StartRead state 1: selects GetPage state 2: selects Write Data state 3: selects WritePage state 4: selects GetFashStatus state 5: selects WriteEnable state 6: selects BulkErase state 7: selects SectorErase state
STATUSREG	0x1800e010	100	R: 3	Bit: 0: indicates RxFifo is Full. 1: indicates RxFifo is not empty 2: indicates SPI is not busy (it is Done).
ADDRESSREG	0x1800e020	0x00000000	W: 32	Flash Memory, page and byte addresses stored in this register.
WRITEDATAREG	0x1800e030	0x00000000	W: 32	Word that will be sent to the flash memory is stored in this register.
READLENGTHREG	0x1800e040	0000000000	W: 10	The number of words that will be read is written in this register.
RXFIFO	0x1800e050	0x00000000	R: 32	Words that are read in ReadDataSate are stored in the FIFO.
SPInCSREG	0x1800e060	00	W: 2	These two bits are used for SPInCS(3:0) bit selection. 00 selects bit0 of SPInCS to be active. 01 selects bit1 of SPInCS to be active 10 selects bit2 of SPInCS to be active 11 selects bit3 of SPInCS to be active

Table 5-2 APBIf registers

CTRLREG, WRITEDATAREG, ADDRREG and READLENGTH registers can only be written. Whereas STATUSREG can only be read.

Writing to CTRLREG initiates one of the ReadData, WriteEnable, GetPage, WriteData, WritePage, GetFlashStatus, WriteEnable, BulkErase or SectorErase states. After initiation of one of these states, CTRLREG will be reset automatically. Whenever one of these states starts, SPI Interface becomes busy. After SPI finishes the current state, Done signal becomes ‘1’ and it will be stored in the STATUSREG. Writing a new data to CTRLREG during the busy period will cause the previous state to be completed unproperly. So, it is suggested to observe the Done bit of the Status Register when there is an interrupt from serial flash memory controller module (Any change in the content of status register causes serial flash memory controller module to generate interrupt). If it is '1', then the new state can be initiated by writing to CTRLREG. If the new state starts, Done bit goes to '0'. One-cycle, active low interrupt is produced, when Done bit goes '1'. Writing to CTRLREG is implemented as in Figure 5-2.

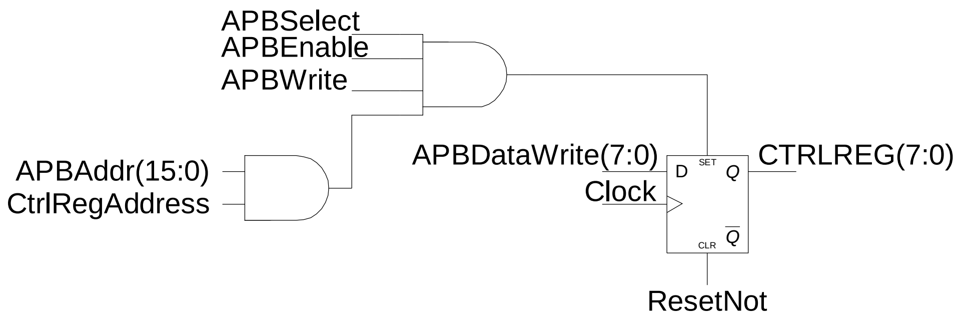


Figure 5-2 Write to CTRLREG

When CTRLREG(0) goes 1 from 0, StartRead signal is generated for SPIIf to start read operation. SPIIf sends StartRead_captured signal with the StartRead signal coming from APB Interface. After receiving the StartRead_captured signal, APBIf deasserts StartRead. GetPage, WriteData, WritePage, GetFlashStatus, WriteEnable, SectorErase, and BulkErase signals are asserted by checking CtrlReg(1), CtrlReg(2), CtrlReg(3), CtrlReg(4), and CtrlReg(5), respectively, and deasserted by checking GetPage_captured, WriteData_captured, WritePage_captured, GetFlashStatus_captured, WriteEnable_captured, BulkErase_captured, and SectorErase_captured signals, respectively. This operation is depicted in Figure 5-3.

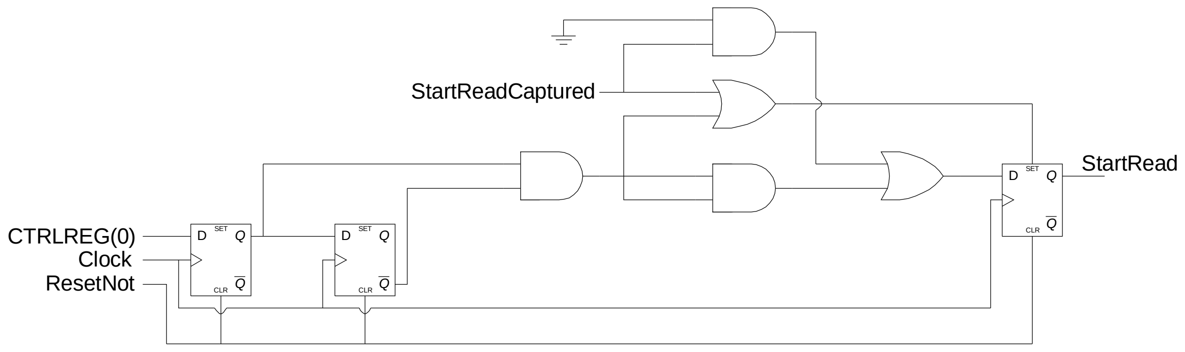


Figure 5-3 Generating StartRead signal

The waveforms of the control signals generated by APBIf can be seen in Figure 5-4.

STATUSREG register holds Full Flag (FF) and notEmptyFlag (nEF) coming from RxFIFO, as well. When RxFIFO is full, FF bit of STATUSREG goes '1' and a one-cycle, active low interrupt is produced at the Interrupt signal. When RxFIFO is not empty, nEF bit of STATUSREG goes '1' and a one-cycle, active low interrupt is produced at the Interrupt signal.

When reading the RxFIFO, the bytes of RxFIFO are placed reversely into the APBDataRead signal. That is the first byte of 32-bit RxFIFO data is assigned to the last byte of 32-bit APBDataRead signal and the last byte of 32-bit RxFIFO data is assigned to the last byte of 32-bit APBDataRead signal. As the same manner, the second byte placed to third byte and the third byte placed to second byte.

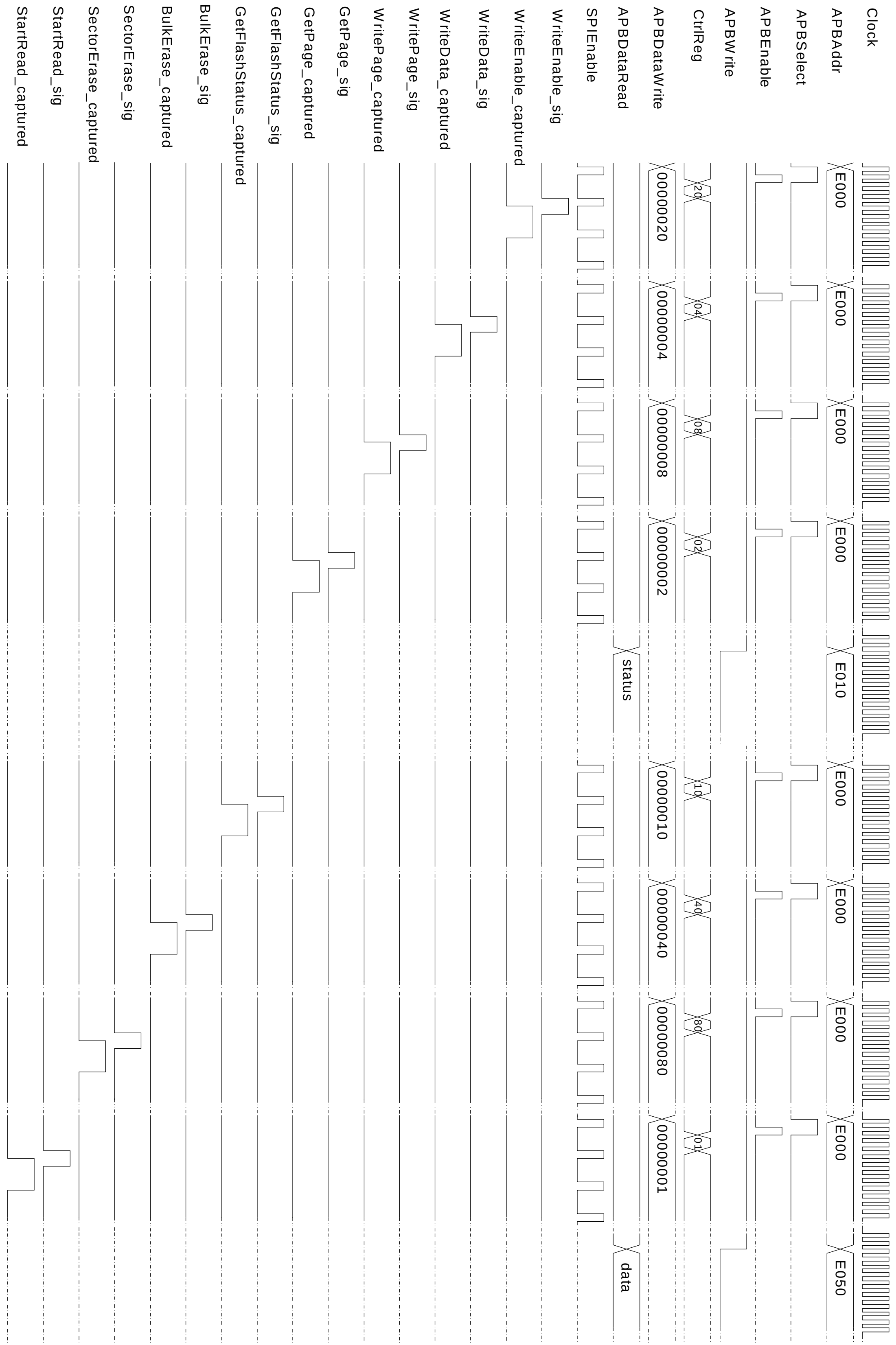


Figure 5-4 Control of SPI operation

5.1.2. SPIIf

5.1.2.1. Interfaces

Name	Size	Direction	Description
APBClock	1	I	Clock
APBResetNot	1	I	Reset
APBIf Interface			
WriteDataReg	32	I	Data input from APBIf
ReadLengthReg	10	I	Length of the data that is read from the flash
AddrReg	32	I	Flash memory address of the data
SPInCSReg	2	I	Chip select input from APBIf
StartRead_sig	1	I	Enable signal to send read opcode
GetPage_sig	1	I	Enable signal to send GetPage opcode
WriteData_sig	1	I	Enable signal to send WriteData opcode
WritePage_sig	1	I	Enable signal to send WritePage opcode
GetFlashStatus_sig	1	I	Enable signal to send GetFlashStatus opcode
WriteEnable_sig	1	I	Enable signal to send WriteEnable opcode
SectorErase_sig	1	I	Enable signal to send SectorErase opcode
BulkErase_sig	1	I	Enable signal to send BulkErase opcode
StartRead_captured	1	O	Indicates that the Read operation is started
GetPage_captured	1	O	Indicates that the GetPage operation is started
WriteData_captured	1	O	Indicates that the WriteData operation is started
WritePage_captured	1	O	Indicates that the WritePage operation is started
GetFlashStatus_captured	1	O	Indicates that the GetFlashStatus operation is started
WriteEnable_captured	1	O	Indicates that the WriteEnable operation is started

SectorErase_captured	1	O	Indicates that the SectorErase operation is started
BulkErase_captured	1	O	Indicates that the BulkErase operation is started
Busy	1	O	Indicates that SPIIf is busy
RxFIFO Interface			
Data_out	32	O	Data input from RxFIFO
FIFOWE	1	O	Write Enable output to RxFIFO
EmptyFlag	1	I	FifoEmpty input from RxFIFO
FullFlag	1	I	FifoFull input from RxFIFO
SPIClockLogic Interface			
SPIEnable	1	I	Indicates the rising edge of the SPI clock
SPIEnableFall	1	I	Indicates the falling edge of the SPI clock
SPIClock	1	I	SPI Clock output to SPIIf block

Table 5-3 List of I/O Interfaces for SPIIf Block

5.1.2.2.Detailed Functional Description

The SPIIf is synchronized on the rising edge of the Clock signal when the SPIEnable signal becomes high. It also uses the SPIEnableFall signal. The SPIClock_Out clock is generated by the SPI Interface block and is a copy of SPIClock. It only toggles when the SPI peripheral is activated by the SPInCS signal.

SPIIf sends opcode, address, data and control data to the peripheral devices. According to the state initiated by APBIf, SPIIf selects the opcode for three different types of flash memories (ATMEL, NEXFLASH, and ST). FlashTypeSel signal selects the flash memory type ("00": ATMEL, "01": ST, "10": NEXFLASH).

There are Setup, Opcode, Address, DontCare, ReadData and Hold internal states working successively in each of the states shown in Figure 5-5. Setup internal state is used to assert the SPInCS signal for 5 SPIClock cycle before the operation begins. In the Opcode State, opcode is sent serially to the flash memory. SPIClock_Out signal starts to toggle with the first bit of the opcode. In the Address State, SPIIf sends the address stored in the ADDRREG serially, right after the opcode. In the DontCare State, some control bits needed for different types of flash memories (ATMEL, ST, and NEXFLASH) are sent serially. In the ReadData State, SPI Interface starts to read the data coming from the flash memory, as much as specified in the READLENGTHREG. During this period, SPIIf sends dummy bits to DoutSPI. In the WriteData State SPIIf sends 32 bit data serially to DoutSPI. In the Hold State the SPInCS is holded 5 SPIClock cycle without toggling the SPIClock_Out signal.

SPIIf states are shown in Figure 5-5.

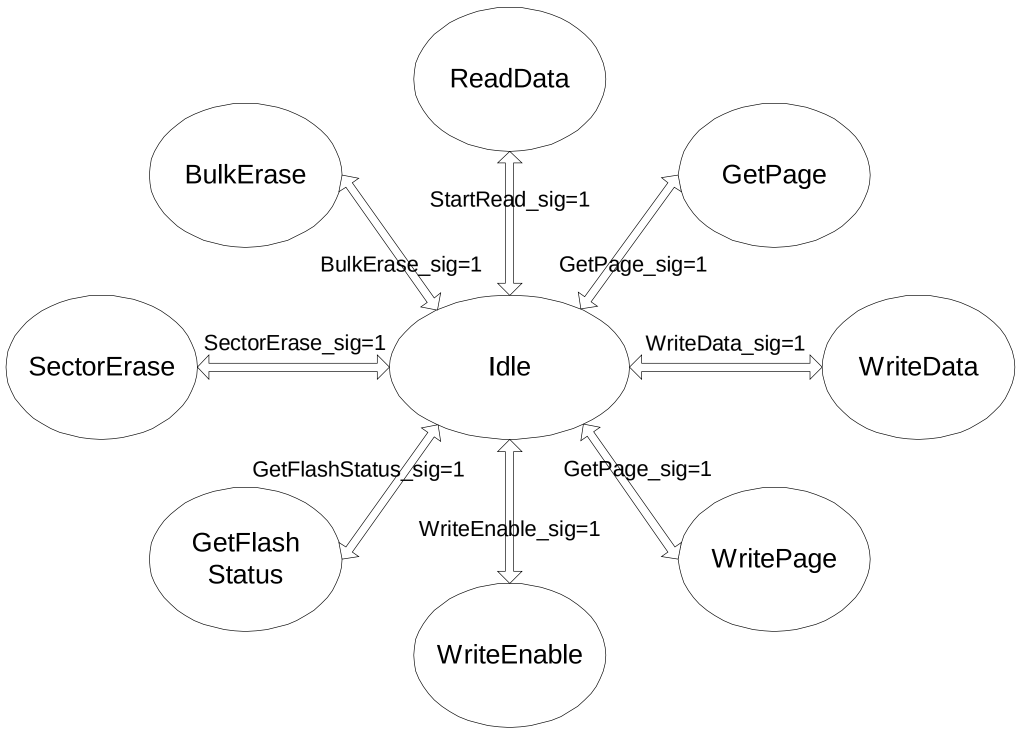


Figure 5-5 SPIIf states

For ReadDataState, SPIIf sends ReadData opcode, address bits and control data. Then it gets data from external peripheral device as much as specified in READLENGTHREG. The CS pin must remain low during the loading of the opcode, the address bits, the control bits, and the reading of data. When StartRead_sig goes '1', current state changes from Idle to ReadDataState. Internal states of ReadDataState are shown in Figure 5-6.

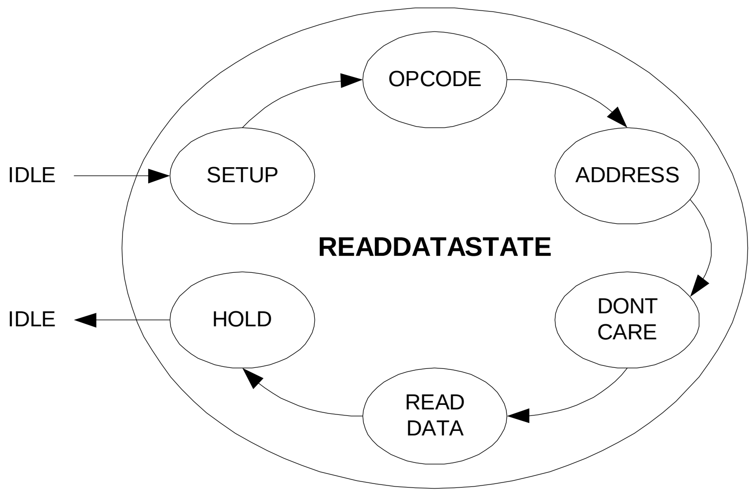


Figure 5-6 Internal states of ReadDataState

For GetPageState (for ATMEL and NEXFLASH), SPIIf sends GetPage opcode, address bits and control bits (only for NEXFLASH). The CS pin must be low while toggling the serial clock pin to load the opcode and the address bits. The transfer of the page of data from the main memory to the buffer will begin when the CS pin transitions from a low to a high state. The internal states of GetPageState are shown in Figure 5-7.

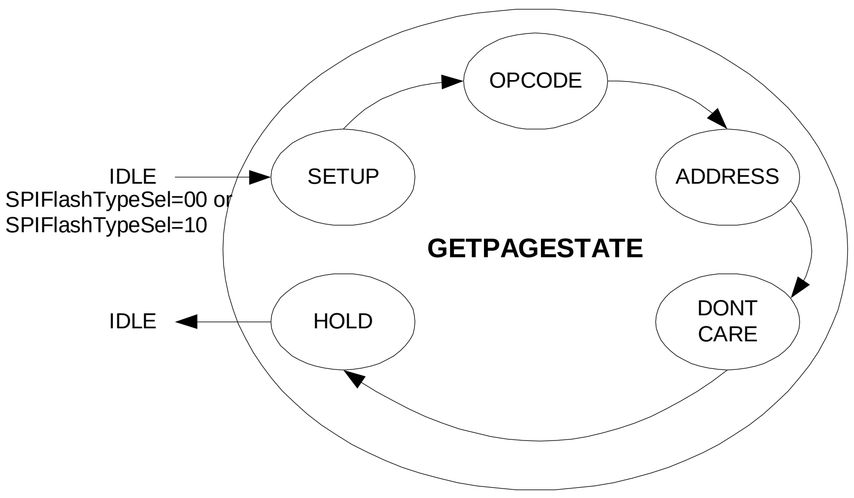


Figure 5-7 Internal states of GetPageState

For WriteDataState, SPIIf sends WriteData opcode, address bits, 32 bit data and control bits (only for NEXFLASH) to the external peripheral device. While sending the content of WRITEDATAREG, bytes are reversed. After the last address byte has been clocked into the device, data can then be clocked in on subsequent clock cycles. Data will continue to be loaded into the flash memory buffer until a low-to-high transition is detected on the CS pin. Figure 5-8 shows the internal states of WriteDataState.

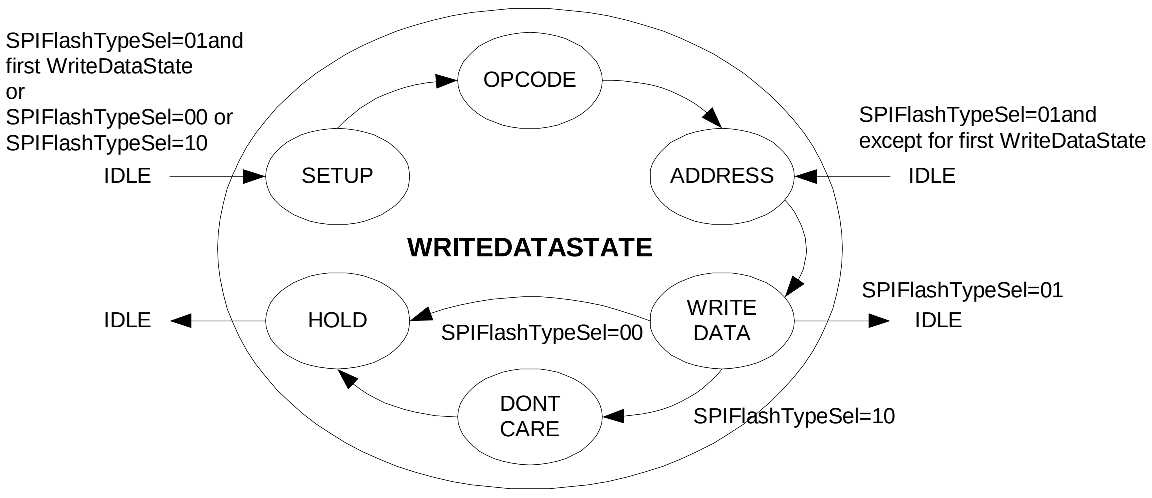


Figure 5-8 Internal states of WriteDataState

For WritePageState, SPIIf sends WritePage opcode (for ATMEL and NEXFLASH), address bits (for ATMEL and NEXFLASH) and control bits (only for NEXFLASH). When a low-to-high transition occurs on the CS pin, the data is stored in the buffer into the specified page in main memory. The internal states of WritePageState are shown in Figure 5-9.

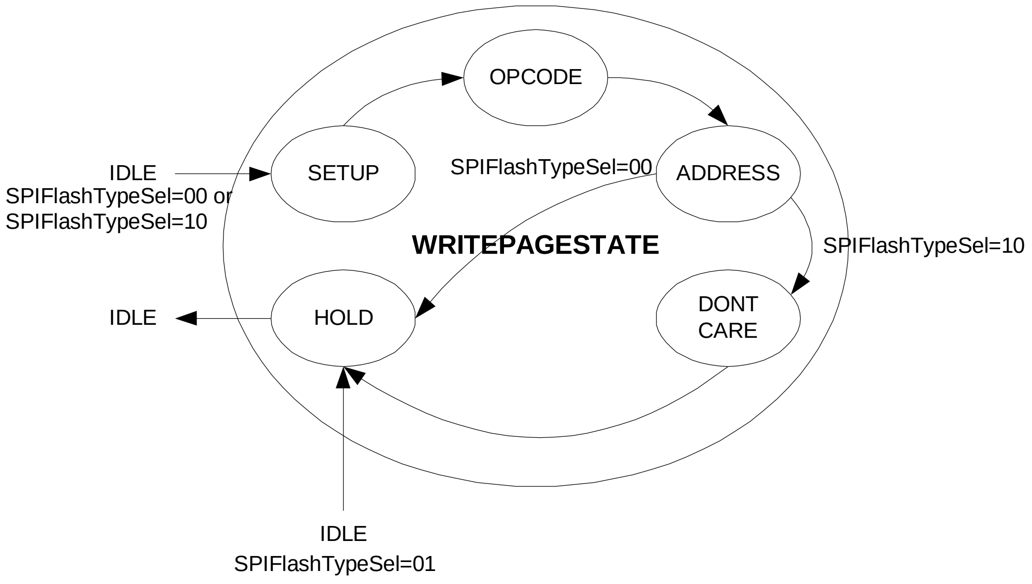


Figure 5-9 Internal states of WritePageState

For WriteEnable state (for NEXFLASH and ST), SPIIf sends opcode and control bits (only for NEXFLASH). The internal states of WriteEnableState are shown in Figure 5-10.

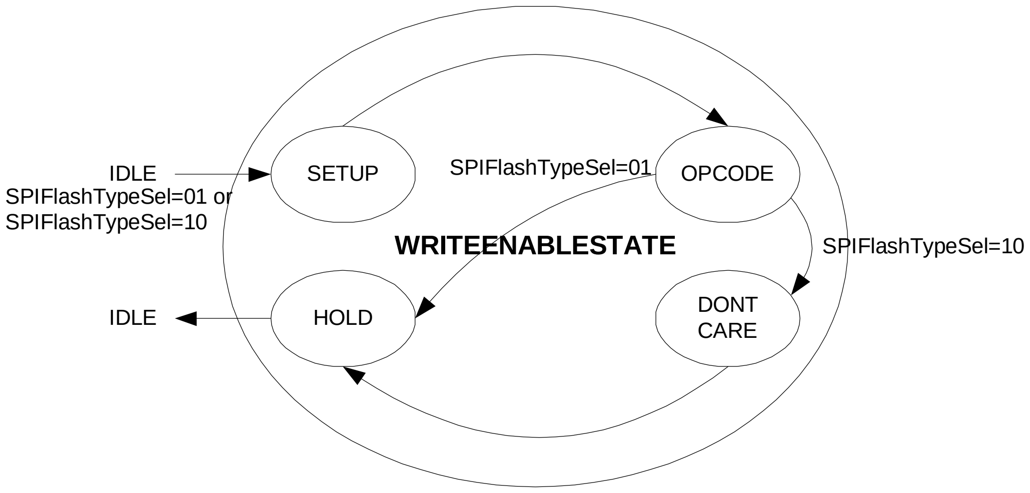


Figure 5-10 Internal states of WriteEnableState

For GetFlashStatusState, SPIIf sends ReadStatus opcode. Status information received from flash is sent to RxFIFO aligned to MSB. The status register can be used to determine the flash memory's ready/busy status. The internal states of GetFlashStatus State are shown in Figure 5-11.

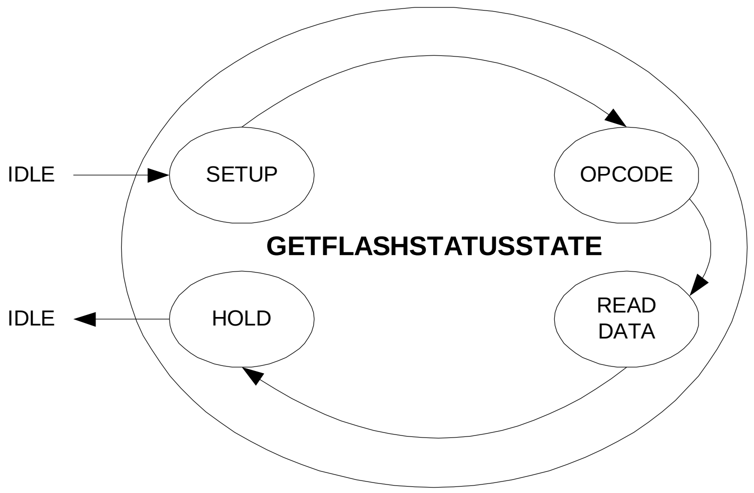


Figure 5-11 Internal states of GetFlashStatusState

For SectorEraseState (only for ST), SPIIf sends opcode and address of sector to be erased. The internal states of SectorEraseState are shown in Figure 5-12.

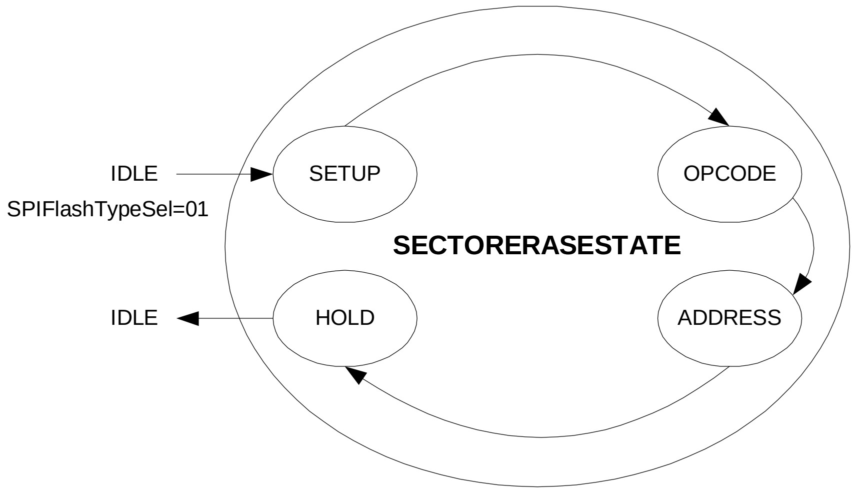


Figure 5-12 Internal states of SectorEraseState

For BulkEraseState (only for ST), SPIIf sends only opcode. The internal states of BulkEraseState are shown in Figure 5-13.

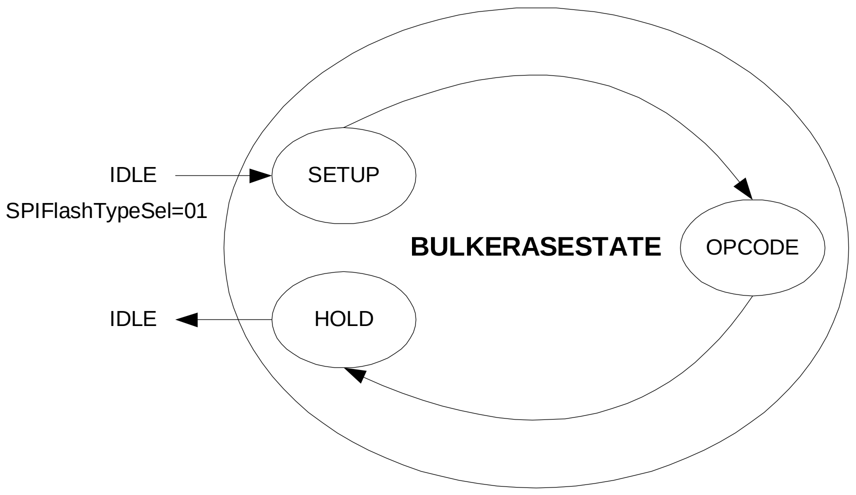


Figure 5-13 Internal states of BulkEraseState

States and opcodes related to these states are described in Table 5-4. These opcodes are different for each Flash type.

State	**Opcode**	**Flash Type**
ReadDataState	0x68	ATMEL
	0x03	ST
	0x50	NEXFLASH
GetPageState	0x53	ATMEL
	-	ST
	0x53	NEXFLASH
WriteDataState	0x84	ATMEL
	0x02	ST
	0x72	NEXFLASH
WritePageState	0x83	ATMEL
	-	ST
	0xF3	NEXFLASH
WriteEnableState	-	ATMEL
	0x06	ST
	0x06	NEXFLASH
GetFlashStatusState	0x57	ATMEL
	0x03	ST
	0x84	NEXFLASH
SectorEraseState	-	ATMEL
	0xD8	ST
	-	NEXFLASH
BulkEraseState	-	ATMEL
	0xC7	ST
	-	NEXFLASH

Table 5-4 States and Opcodes Used In SPI Interface

The waveforms of SPIIf for Atmel flash memory are shown in following figures.

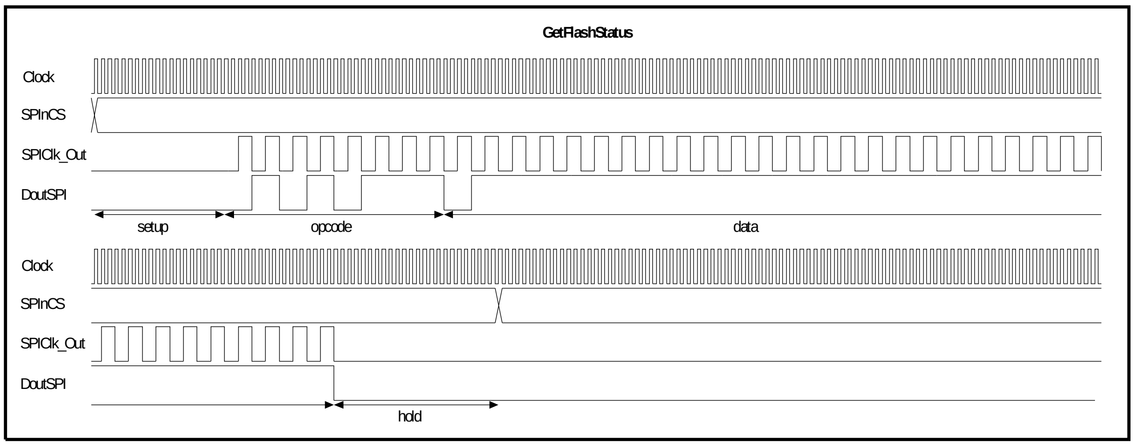


Figure 5-14 GetFlashStatusState waveforms for Atmel Flash Memory

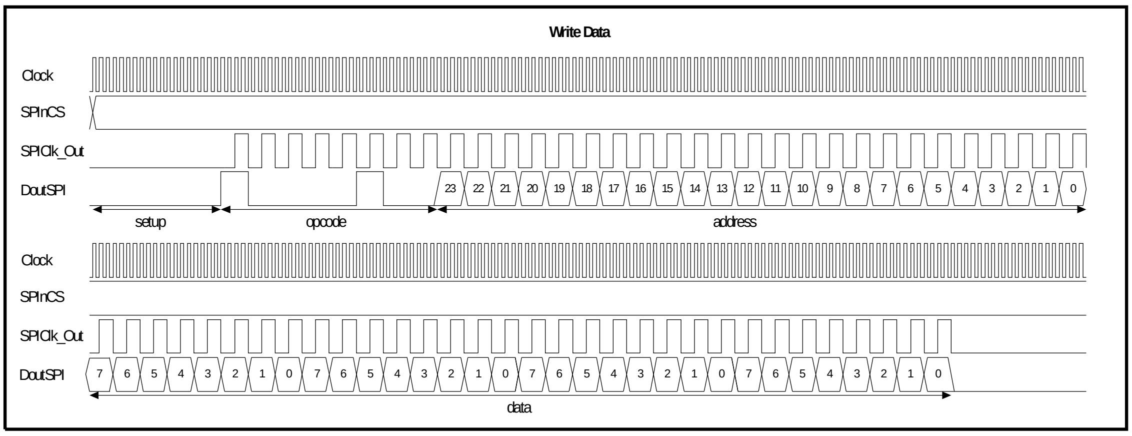


Figure 5-15 WriteDataState waveforms for Atmel Flash Memory

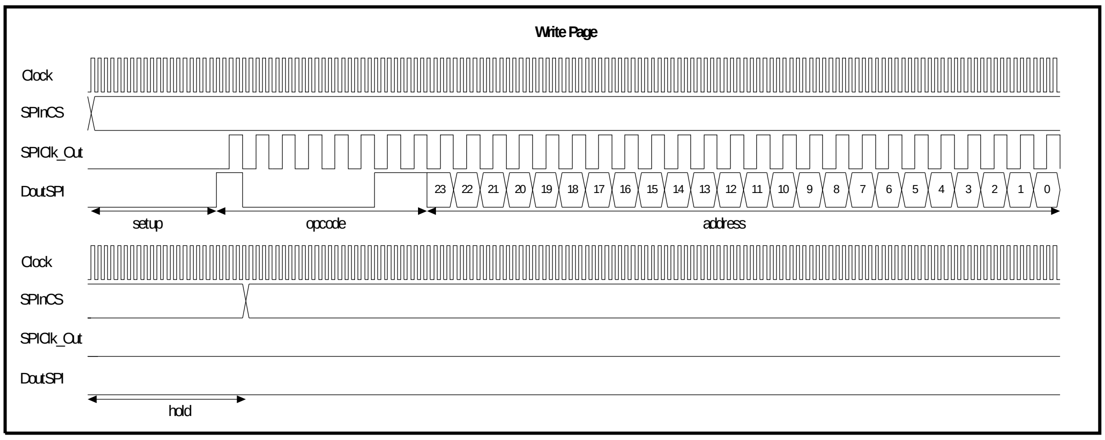


Figure 5-16 WritePageState waveforms for Atmel Flash Memory

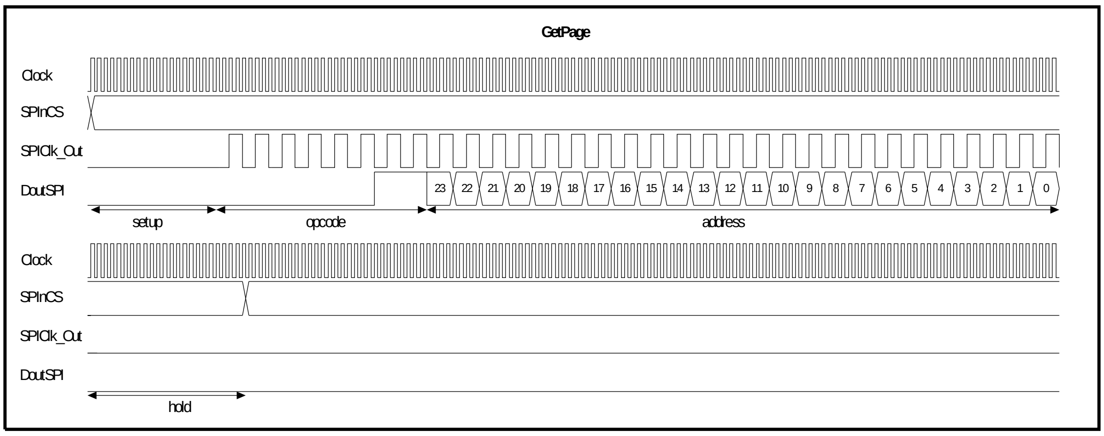


Figure 5-17 GetPageState waveforms for Atmel Flash Memory

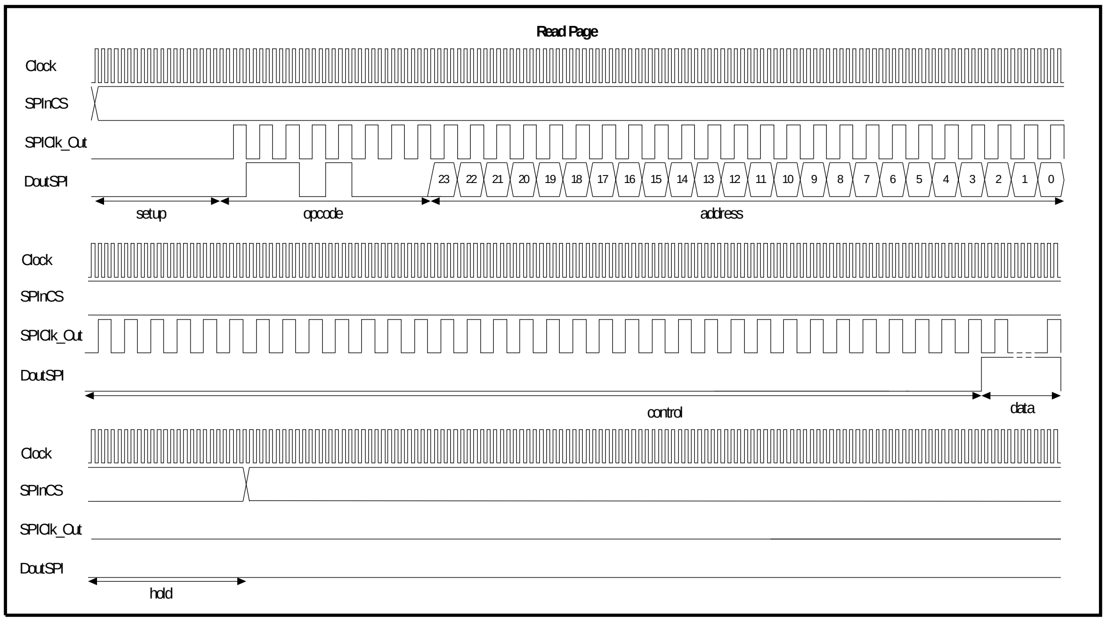


Figure 5-18 ReadPageState waveforms for Atmel Flash Memory

The waveforms of SPIIf for NexFlash flash memory are shown in following figures.

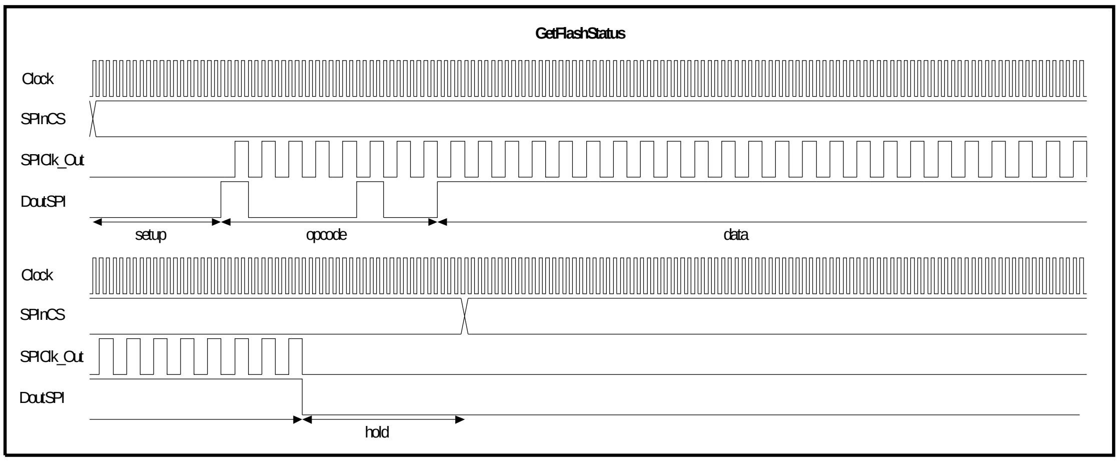


Figure 5-19 GetFlashStatusState waveforms for NexFlash Flash Memory

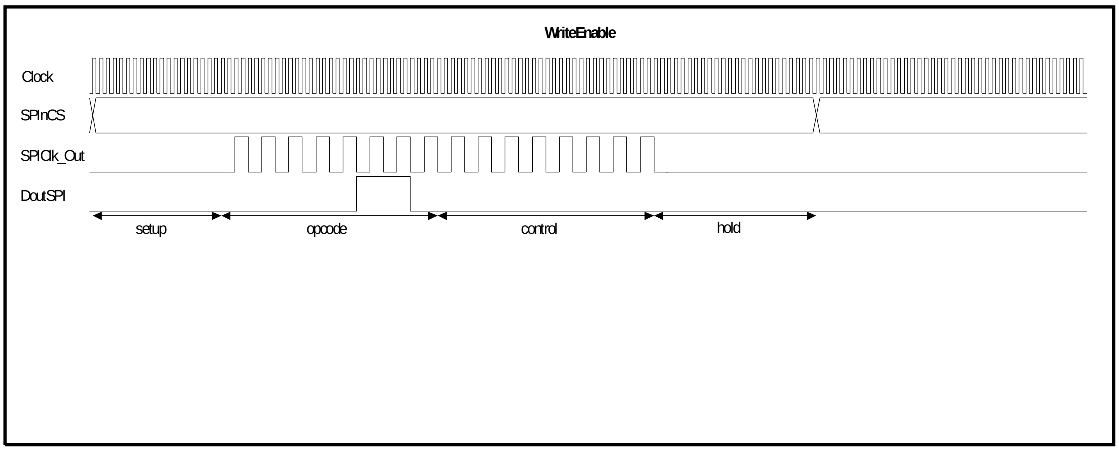


Figure 5-20 WriteEnableState waveforms for NexFlash Flash Memory

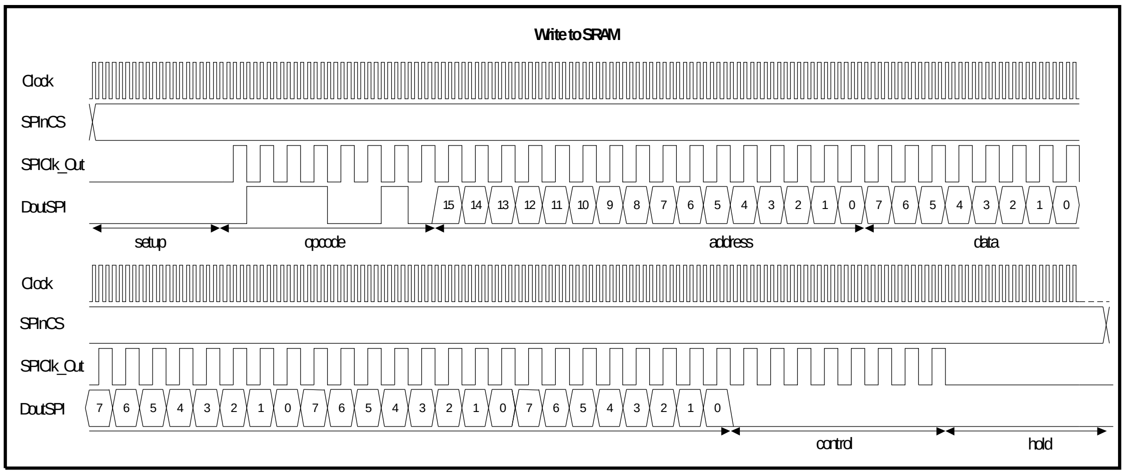


Figure 5-21 Write to SRAM waveforms for NexFlash Flash Memory

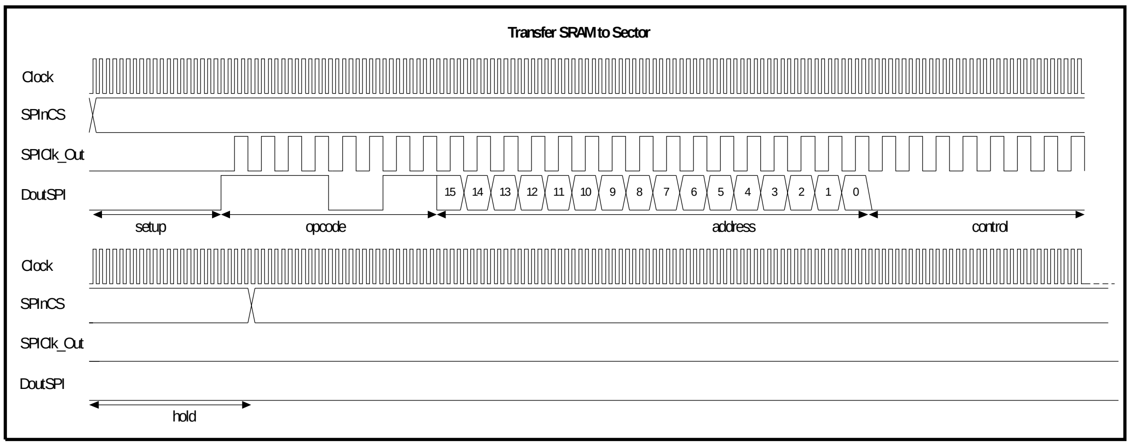


Figure 5-22 Transfer SRAM to Sector waveforms for NexFlash Flash Memory

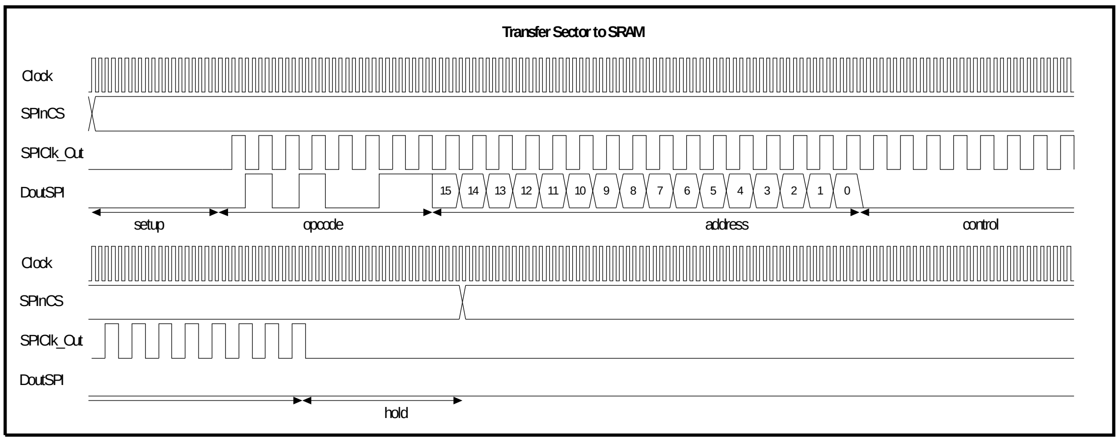


Figure 5-23 Transfer Sector to SRAM waveforms for NexFlash Flash Memory

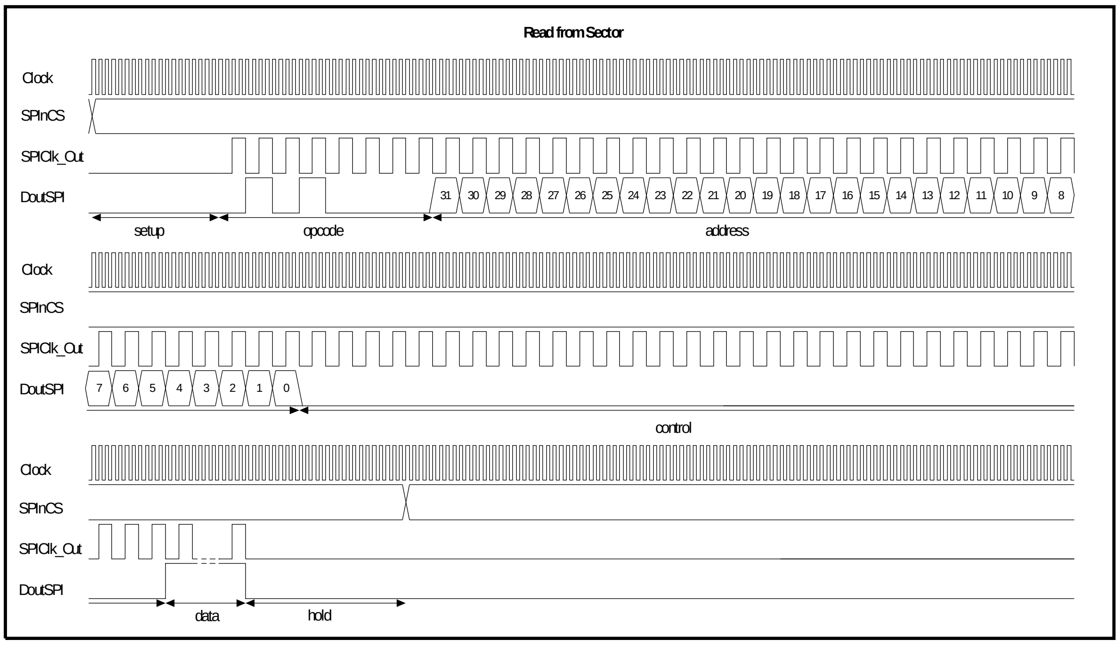


Figure 5-24 Read from Sector waveforms for NexFlash Flash Memory

The waveforms of SPIIf for ST flash memory are shown in following figures.

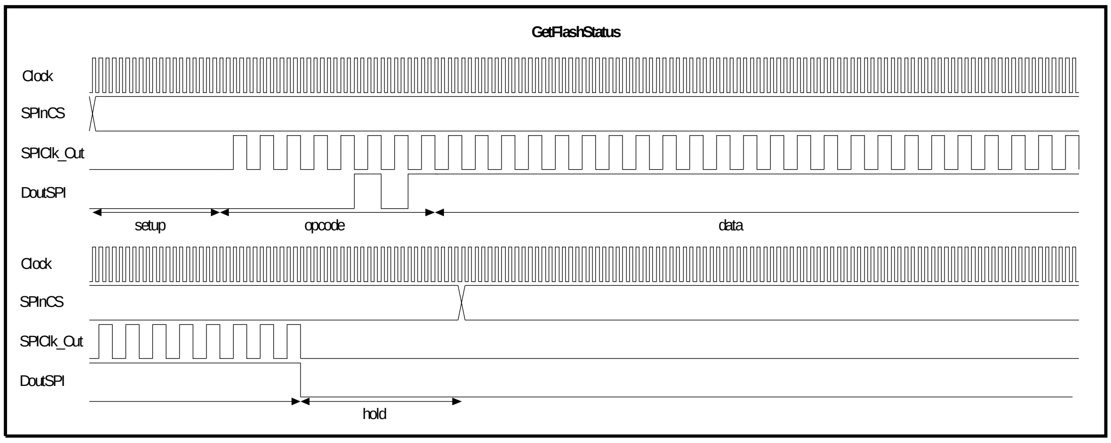


Figure 5-25 GetFlashStatusState waveforms for ST Flash Memory

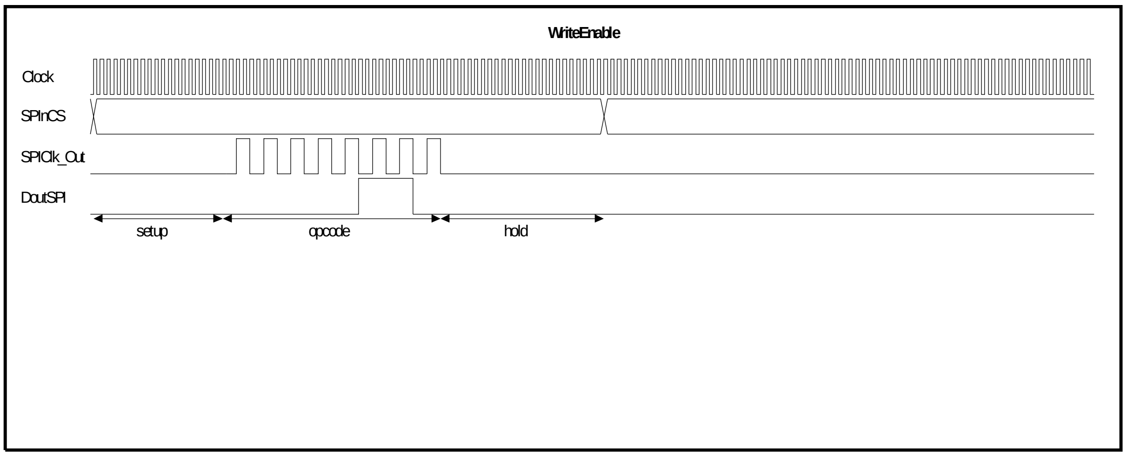


Figure 5-26 WriteEnableState waveforms for ST Flash Memory

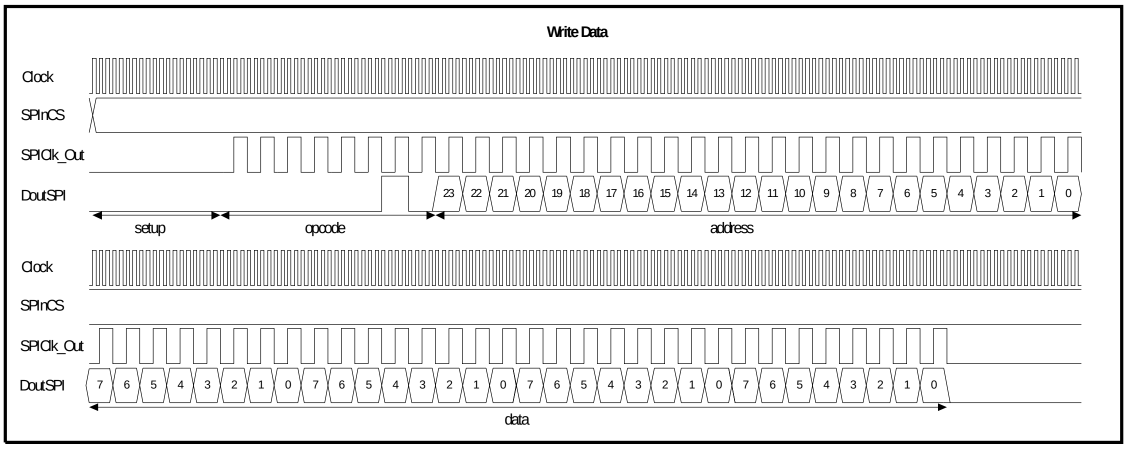


Figure 5-27 WriteDataState waveforms for ST Flash Memory

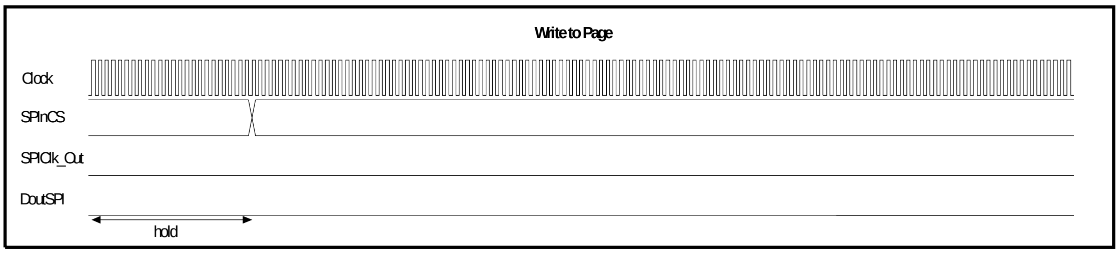


Figure 5-28 WritePageState waveforms for ST Flash Memory

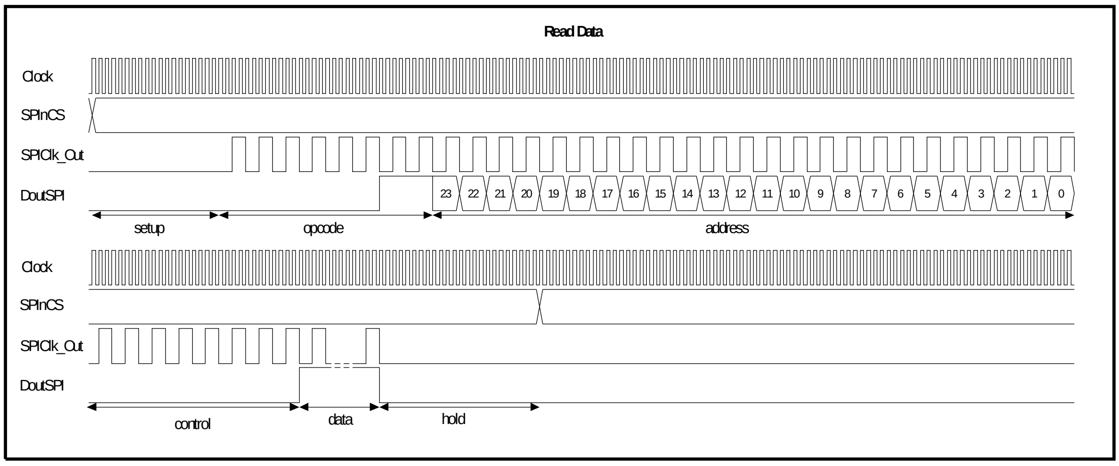


Figure 5-29 ReadDataState waveforms for ST Flash Memory

5.1.3. SPIClockLogic

5.1.3.1. Interfaces

Name	Size	Direction	Description
APBClock	1	I	Clock
APBResetNot	1	I	Reset
RxFIFO Interface			
FifoFullFlag	1	I	FifoFull input from RxFIFO
SPIIF Interface			
SPIEnable	1	O	Indicates the rising edge of the SPI clock
SPIEnableFall	1	O	Indicates the falling edge of the SPI clock
SPIClock	1	O	SPI Clock output to SPIIf block

Table 5-5 List of I/O Interfaces for SPIIf Block

5.1.3.2.Detailed Functional Description

SPIClockLogic module produces SPIEnable, SPIEnableFall and SPIClock signals for SPI Interface and RxFifo. SPIClock frequency is 1/4 of the Clock signal for ATMEL Flash, ST Flash, and NexFlash Flash. SPIEnable signal is used for receiving the data coming from DinSPI on the rising edge of Clock signal. It's an active high one Clock cycle signal. On the other hand, SPIEnableFall signal stands for the falling edge of the SPIClock and it is used to send the data to DoutSPI. Its duration is also one Clock cycle.

5.1.4.RxFIFO

5.1.4.1.Interfaces

Name	Size	Direction	Description
APBclk	1	I	Clock
APBrst	1	I	Reset
APBIf Interface			
APBrd	1	I	Read Enable input from APBIf
full	1	O	FifoFull output to APBIf
empty	1	O	FifoEmpty output to APBIf
APBDataRead	32	O	Data output to APBIf
SPIIf Interface			
SPIwr	1	I	Write Enable input from SPIIf
full	1	O	FifoFull output to SPIIf
SPIDataWrite	32	I	Data input from SPIIf
SPIClockLogic Interface			
full	1	O	FifoFull output to SPIClockLogic
SPIEnable	1	I	Indicates the rising edge of the SPI clock

Table 5-6 List of I/O Interfaces for RxFifo Block

5.1.4.2.Detailed Functional Description

RxFifo is a 32 bit wide 4 location deep receive first in first out memory buffer. Received data from the serial interface are stored in the buffer until read out by the CPU across the AMBA APB interface. SPI interface writes data into the RxFIFO on the rising edge of Clock signal when SPIEnable is high. APB interface reads RxFIFO on the rising edge of Clock signal when read enable is high. When RxFIFO is empty, it produces empty flag and when it is full it produces full flag.

5.2. Upload and Download Timings

Calculations of upload and download timings at 10 MHz for ATMEL, ST and NexFLASH serial flash memories are given in following sections.

5.2.1. ATMEL:

Calculation for Upload Time:

WriteData Time = 5 cycle (setup time for Chip Select)+ 5 cycle (hold time for Chip Select)+8 cycle (opcode)+24 cycle (address)+32 cycle (data) = 74 cycle

WritePage Time = 5 cycle (setup time for Chip Select)+ 5 cycle (hold time for Chip Select)+8 cycle (opcode)+24 cycle (address) = 42 cycle

Max PageProgram Time = 20ms

Upload Time =(WriteData Time x 1056/4 + WritePage Time) / ($10x10^6$) + Max PageProgram Time = 21.96ms

Calculation for Download Time:

ReadData Time = 5 cycle (setup time for Chip Select)+ 5 cycle (hold time for

Chip Select)+8 cycle (opcode)+24 cycle (address)+32 cycle (control) = 74 cycle

Download Time = (ReadData Time + 1056 x 8) / ($10x10^6$) = 0.85ms

5.2.2. ST:

Calculation for Upload Time:

WriteEnable Time = 5 cycle (setup time for Chip Select)+ 5 cycle (hold time for Chip Select)+8 cycle (opcode) = 18 cycle

WriteData Time = 5 cycle (setup time for Chip Select)+8 cycle (opcode)+24 cycle (address)+32 cycle (data) = 69 cycle (this timing is for first 32 bit (4 byte) data, 252 byte data remains. Except first WriteData operation (first 32 bit data), all the other WriteData states send only 32 bit data serially to ST Flash)

WritePage Time = 5 cycle (hold time for Chip Select) = 5 cycle

Max PageProgram Time = 5ms

Upload Time =(WriteEnable Time + WriteData Time + WritePage Time + 32 x 252/4) / (10×10^6) + Max PageProgram Time = 5.21ms

Calculation for Download Time:

ReadData Time = 5 cycle (setup time for Chip Select)+ 5 cycle (hold time for Chip Select)+8 cycle (opcode)+24 cycle (address) = 42 cycle

Download Time = (ReadData Time + 256 x 8) / (10×10^6) = 0.21ms

5.2.3. NEXFLASH:

Calculation for Upload Time:

WriteEnable Time = 5 cycle (setup time for Chip Select)+ 5 cycle (hold time for Chip Select)+8 cycle (opcode)+8 cycle (control) = 26 cycle (this state is required only for first write operation. The remaining write data states don't require WriteEnable State).

WriteData Time = 5 cycle (setup time for Chip Select)+ 5 cycle (hold time for Chip Select) +8 cycle (opcode)+16 cycle (address)+32 cycle (data)+8 cycle (control) = 74 cycle

WritePage Time = 5 cycle (setup time for Chip Select)+ 5 cycle (hold time for Chip Select)+8 cycle (opcode)+16 cycle (address)+16 cycle (control) = 50 cycle

Max PageProgram Time = 15ms

Upload Time =(WriteEnable Time + WriteData Time x 528/4 + WritePage Time) / (10x10^6) + Max PageProgram Time = 15.98ms

Calculation for Download Time:

ReadData Time = 5 cycle (setup time for Chip Select)+ 5 cycle (hold time for Chip Select)+8 cycle (opcode)+32 cycle (address) + 32 cycle (control) = 82 cycle

Download Time = (ReadData Time + 528 x 8) / (10x10^6) =0.43 ms

5.3. Functionality Comparison

The following table gives the comparison between several SPI blocks and our serial flash controller block according to the functional features.

Serial Peripheral Interface	APB Compliance	SPI Mode	Phase/ Polarity	FIFO Size	Word Size	Transmission Speed
Serial Flash Controller	Yes	Master	Only Mode0	32bit x 4 for Rx	32 bit	1/4 of system clock
Actel	Yes	Master/ Slave	-	32byte for Rx and Tx	-	Up to 1/8 of system clock
Alma-Tech	No	Master/ Slave	Not Programmable	-	8 bit	Up to 1/2 of system clock
Atmel	Yes	Master/ Slave	Programmable	-	Programmable up to 16 bit	Up to 1/2 of system clock
Cadence	Yes	Master/ Slave	Programmable	8byte for Rx and Tx	-	1/2 of system clock
Palmchip	No	Master/ Slave	Programmable	8byte for Rx and Tx	8 bit	-
Scenix	No	Master/ Slave	Programmable	-	Programmable up to 16 bit	Up to 1.72MHz
Sota	Yes	Master	Programmable	-	Programmable	Programmable

Table 5-7 Comparisons with other serial peripheral interface blocks

Our serial flash memory controller module has APB interface. It’s an important feature since the module can be easily adapted to processor platform. Our serial flash memory controller module can only work in master mode. Actually working in slave mode is not needed for our module, because the slave is always the serial flash memory. Our serial flash memory controller module has a receive FIFO. This is necessary to have faster download speed. Our module needs only one clock input. It produces the SPI clock internally. Some of serial peripheral interface controllers need external SPI clock.

Different serial peripheral interface controllers have different control schemes. In general, the processor programs serial peripheral interface controllers to send the data according to the frame format required by flash memory. The frame includes dummy bits for setup and hold times, opcodes and data bits. Software should take care of all these while sending data to flash memory. Serial peripheral interface controller only serially sends the data to the serial flash memory. If different flash memory is used, then software has to be changed completely according to new flash memory opcodes and timings. Therefore the transfer is completely controlled by software. For example lets assume that serial peripheral interface controller has 4x32 bit transmit FIFO. Then to write 0x12345678 to first address of the buffer of Atmel serial flash memory, following data should be written to transmit FIFO.

First address:

Opcode: 0x84, Address: 0x000000

10000100000000000000000000000000

Second address:

Data: 0x12345678

00010010001101000101011001111000

After the processor fills the transmit FIFO, it should start serial transfer by writing control register of serial peripheral interface controller. The operation given in above example is very simple one.

The serial peripheral interface controllers use the methods given below to transmit the data serially to flash memory:

- It sends the contents of transmit FIFO continuously.
- It stops the serial data clock between the transfer of the data in first address and the data in second address. Also it keeps the serial flash memory to be selected until the transfer is completed.

Our serial flash memory controller module doesn't need transmit FIFO, since it works based on commands.

Also serial peripheral interface controllers should have a programmable chip select time. This is required for the setup and hold time requirements of chip select signal.

In our serial flash controller module, the processor doesn't have to care about the opcodes and the timings. Software only initiates the transfer by sending related command to serial flash memory controller module. The transfer of data is completely controlled by hardware. So it is simpler to program our module for the required operation.

Serial flash controller module sends opcode and data according to setup and hold time requirements of selected flash memory. When it finishes its operation, it generates one cycle interrupt. While reading from flash memory, processor programs serial flash controller for the length of data to be read and then it only waits for the interrupts. Each time the processor receives interrupt from serial flash controller, it reads the FIFO. So the processor needs less cycle to send and receive the data.

The serial flash memory performs steps below for different operations:

- Steps for Reading Data from Flash:
  - § Software should follow the steps below:
    - ⇒ Write starting page address to ADDRREG.
    - ⇒ Write # of words to be read from Flash to READLENGTHREG.
    - ⇒ Write 0x00000001 to CTRLREG to start the read operation.
    - ⇒ An interrupt is produced when FIFO is not empty.
    - ⇒ Read data from RxFIFO.
    - ⇒ While reading, poll notEmpty bit of STATUSREG to avoid reading when FIFO is empty.
    - ⇒ When SPI fills # of data determined by READLENGTH, it sets Done bit of STATUSREG to 1, and an interrupt is generated.
- Steps for Page to Buffer Transfer of Flash:
  - § Software should follow the steps below:
    - ⇒ Write page address to ADDRREG.

⇒ Write 0x00000002 to CTRLREG to start the GetPage operation.

⇒ When SPI sends the opcode related with GetPage, it fires Done bit of STATUSREG to 1, and an interrupt is generated.

⇒ Check the flash status before starting a new operation.

- Steps for Writing Data into Flash Buffer:

  § Software should follow the steps below:

  ⇒ Write starting byte address to ADDRREG.

  ⇒ Write one word data to be sent to WRITEDATAREG.

  ⇒ Write 0x00000004 to CTRLREG to start the WriteData operation.

  ⇒ When SPI sends last byte of data, it fires Done bit of STATUSREG to 1, and an interrupt is generated.

- Steps for Writing Flash Buffer into Page:

  § Software should follow the steps below:

  ⇒ Write page address to ADDRREG.

  ⇒ Write 0x00000008 to CTRLREG to start the WritePage operation.

  ⇒ When SPI sends the opcode related with WritePage, it fires Done bit of STATUSREG to 1, and an interrupt is generated.

  ⇒ Check the flash status before starting a new operation.

- Steps for GetFlashStatus State:

  § Software should follow the steps below:

  ⇒ Write 0x00000010 to CTRLREG to start the GetFlashStatus operation.

  ⇒ An interrupt is produced when FIFO is not empty.

  ⇒ Read data from RxFIFO. The MSB contains Flash Status.

  ⇒ When SPI finishes GetFlashStatus operation, it fires Done bit of STATUSREG to 1, and an interrupt is generated.

- Steps for WriteEnable State

- § Software should follow the steps below:
  - ⇒ Write 0x00000020 to CTRLREG to start the write operation.
  - ⇒ When SPI finishes WriteEnable operation, it fires Done bit of STATUSREG to 1, and an interrupt is generated.

- Steps for Sector Erase State
  - § Software should follow the steps below:
    - ⇒ Write 0x00000040 to CTRLREG to start the Sector Erase operation.
    - ⇒ When SPI finishes Sector Erase operation, it fires Done bit of STATUSREG to 1, and an interrupt is generated.
    - ⇒ Check the flash status before starting a new operation.
- Steps for Bulk Erase State
  - § Software should follow the steps below:
    - ⇒ Write 0x00000080 to CTRLREG to start the Bulk Erase operation.
    - ⇒ When SPI finishes Bulk Erase operation, it fires Done bit of STATUSREG to 1, and an interrupt is generated.
    - ⇒ Check the flash status before starting a new operation.

For setup and hold time requirements SPI sends extra 5 bits after making SPInCS active and before making it inactive.

6. SERIAL FLASH MEMORY CONTROLLER VERIFICATION

After the design has been captured in HDL, it is essential to verify that the code matches the required functionality. Our verification environment consists of processor platform, serial flash memory controller module (ApbSPI) and serial flash memory models. To simplify the testbench development, the processor model is replaced by SoftFrog model. SoftFrog is a mechanism to make a C-program communicate with the VHDL-simulator. SoftFrog can handle the interrupts.

To verify the serial flash memory controller module, following tests were executed:

- Verify of reset values of the control and status registers.
- Modify the control and status registers, and check the new values.
- Configure the serial flash memory controller for different serial flash memories and perform read and write operations.

The following sections include the test scenarios for functional verification of serial flash memory controller module for different flash memories. Verification Navigator was used to specify the code coverage of testbenches. We have a statement coverage about 98%.

6.1. Test Scenarios

6.1.1. ATMEL Flash Test

AT45DB642 (64 Mbit) Denali Flash model is used to test SPI Block for ATMEL Flash. Steps followed for testing the ATMEL Flash are listed below:

- Select Flash: Firstly, Flash type is selected by applying "00" to the FlashTypeSel input of SPI block.
- Select SPInCS: The CS signal of 64Mbit ATMEL Flash is connected to the SPInCS(0) pin of SPI block. In order to select this pin to be active, "00" is written into the SPInCSREG.
- Write to Buffer1: Fill all the 1056 bytes of Buffer1 of the ATMEL Flash with data. When writing the data into the Buffer1, the order should be like this: First, write to the ADDRESSREG, then write to WRITEDATAREG, and last write to CTRLREG.
- Transfer Buffer1 to Page1: After writing the data to the Buffer1, its content is transferred to Page1 by writing to ADDRESSREG and CTRLREG. During transfer, check the status of Flash whether it is busy or not.
- Fill Buffer1 with zeros: If the Flash is ready, Buffer1 is filled with zeros to clear it.
- Transfer Page1 to Buffer1: Page1 is transferred back to Buffer1 by writing to ADDRESSREG and CTRLREG. During transfer, check the status of Flash whether it is busy or not.
- Change Buffer1: If the Flash is ready, change the content of Buffer1.
- Transfer Buffer1 to Page1: New Buffer1 content is transferred to Page1 again by writing to ADDRESSREG and CTRLREG. During transfer, check the status of Flash whether it is busy or not.
- Write to Buffer1: If the Flash is ready, Buffer1 is loaded with a different 1056-byte data.
- Transfer Buffer1 to Page2: Buffer1 is transferred to Page2 by writing to ADDRESSREG and CTRLREG. During transfer, check the status of Flash whether it is busy or not.
- Read Two Pages Continuously: When Flash is ready, Page1 and Page2 are read successively by writing to ADDRESSREG and CTRLREG. In continuous reading mode FIFO becomes empty after each reading. If data is not continuously read, FIFO becomes full after 4 words are stored in it. Both conditions (FIFO full and FIFO empty) are tested for SPI verification.

6.1.2. ST Flash Test:

M25P80 (8 Mbit) VHDL model is used to test SPI Block for ST Flash. Each SPInCS bits connected to one ST Flash. Steps followed for testing the ST Flash are listed below:

- Select Flash: Firstly, Flash type is selected by applying "01" to the FlashTypeSel input of SPI block.
- Select SPInCS: In order to select the second bit of the SPInCS output to be active, "01" is written into the SPInCSREG.
- Send Write Enable Opcode: Send Write Enable opcode to the Flash to make the Flash Pages writable. This operation is performed by writing to the CTRLREG. Before every write operation, this opcode is required by ST Flash.
- Write to Page1: Write all the 256 bytes of Page1 of second Flash with data by using WriteData state. When writing the first 32-bit of data into Page1, the order should be like this: First, write to the ADDRESSREG (including page and byte addresses), then write to WRITEDATAREG, and last write to CTRLREG. The remaining 32-bit data must be sent successively by writing to WRITEDATAREG. After sending all the data, set CTRLREG to start WritePage State. Before performing new instruction after write page state, check the status of Flash whether it is busy or not.
- Send Write Enable Opcode: If the Flash is ready, send again Write Enable opcode to the Flash to make the Flash Pages writable. This operation is performed by writing to the CTRLREG.
- Write to Page2: Write all the 256 bytes of Page2 of second Flash with different data as described in "Write to Page1" paragraph. Check the status of Flash whether it is busy or not.
- Read Two Pages Continuously: When Flash is ready, Page1 and Page2 of second Flash are read successively by writing to ADDRESSREG and CTRLREG. In continuous reading mode FIFO becomes empty after each reading. If data is not continuously read, FIFO becomes full after 4 words are stored in it. Both conditions (FIFO full and FIFO empty) are tested for SPI verification.
- Select SPInCS: In order to select the third bit of the SPInCS output to be active, "10" is written into the SPInCSREG.

- Send Write Enable Opcode: Send WriteEnable opcode to the Flash.
- Write to Page1: Write all the 256 bytes of Page1 of the third Flash with data as described in "Write to Page1" paragraph. Check the status of Flash whether it is busy or not.
- Read Page1: If the Flash is ready, read Page1 of the third Flash.

Note: SectorErase and BulkErase tests are done partially. That is, we send only Sector Erase and Bulk Erase opcodes to ST Flash and observe whether the ST Flash receives these opcodes or not by checking the Flash Status.

6.1.3. NexFLASH Flash Test:

NX25F641C (64 Mbit) verilog model is used to test SPI Block for NEXFLASH Flash and it is connected to SPInCS(3). Steps followed for testing the NEXFLASH Flash are listed below:

- Select Flash: Firstly, Flash type is selected by applying "10" to the FlashTypeSel input of SPI block.
- Select SPInCS: The CS signal of 64Mbit NEXFLASH Flash is connected to the SPInCS(3) signal of SPI block. In order to select this signal to be active, "11" is written into the SPInCSREG.
- Send Write Enable Opcode: Send Write Enable opcode to the Flash to make the Flash Pages writable. This operation is performed by writing to the CTRLREG. This operation should be performed only once after power on.
- Write to SRAM: Fill all the 528 bytes of SRAM of the NEXFLASH Flash with data. When writing the data into the SRAM, the order should be like this: First, write to the ADDRESSREG, then write to WRITEDATAREG, and last write to CTRLREG.
- Transfer SRAM to Sector1: After writing the data to the SRAM, its content is transferred to Sector1 by writing to ADDRESSREG and CTRLREG. During transfer, check the status of Flash whether it is busy or not.
- Change the content of SRAM: If the Flash is ready, change the content of SRAM with new data.

- Transfer SRAM to Sector2: SRAM is transferred to Sector2 by writing to ADDRESSREG and CTRLREG. During transfer, check the status of Flash whether it is busy or not.
- Read Two Sectors Continuously: When Flash is ready, Sector1 and Sector2 are read successively by writing to ADDRESSREG and CTRLREG. In continuous reading mode FIFO becomes empty after each reading. If data is not continuously read, FIFO becomes full after 4 words are stored in it. Both conditions (FIFO full and FIFO empty) are tested for SPI verification.
- Transfer Sector1 to SRAM: Sector1 is transferred back to SRAM by writing to ADDRESSREG and CTRLREG. During transfer, check the status of Flash whether it is busy or not. (check transfer bit of Flash status)
- Change SRAM: If the Flash is ready, change the content of SRAM.
- Transfer SRAM to Sector3: New SRAM content is transferred to Sector3 again by writing to ADDRESSREG and CTRLREG. During transfer, check the status of Flash whether it is busy or not.
- Read Sector3: If the Flash is ready, read Sector3 of the ST Flash.

Waveforms for the simulations of functional tests are given in Appendix A.

7. SERIAL FLASH MEMORY CONTROLLER SYNTHESIS

7.1. Synthesis

Synthesis provides a link between a HDL and a netlist similarly to the way that a C compiler provides a link between C code and machine language. Once a HDL model is complete two items are required to proceed: a logic synthesizer and a cell library that is called the target library. The HDL code is mapped to cells from this library. It is possible to effectively translate designs captured in high level languages to designs optimized for area and speed by using of logic synthesis.

Serial Flash Memory Controller block was synthesized with Synopsys Design Compiler Synthesis tool in CMOS 0.35 μm technology. A synthesis script, that includes all necessary constraints, is used for synthesis. This script is given in Appendix B.

As a synthesis methodology, “top-down” synthesis way was used since it provides a push-button approach and our design is not so large. All constraints were applied to the top-level block, which is called ApbSPI.

Logic synthesis results in a netlist, which contains sequential non-scan cells and other combinational gates from the technology library. Full scan methodology is used in synthesis of Serial Flash Memory Controller block. In this methodology all the sequential cells in the netlist are replaced by scan cells. Full scan designs achieve higher fault coverage. Scan cells have two different modes of operation: normal and scan. In normal mode, the scan cell’s functionality is same as that of sequential non-scan cell. In scan mode, the scan cells are linked in the form of a shift register.

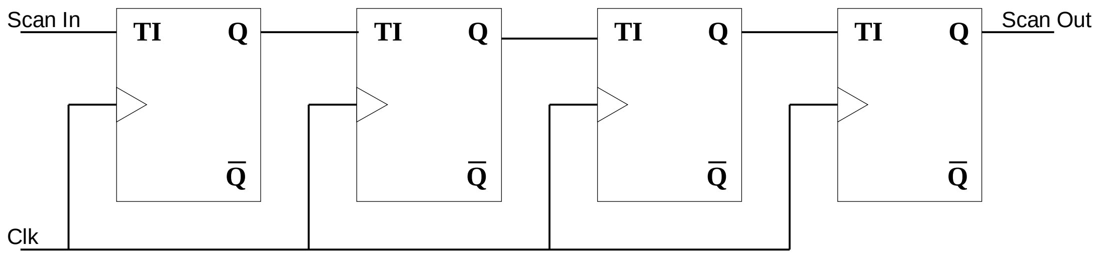


Figure 7-1 Scan cells linked to form a scan chain

When scan cells are linked to form a scan chain as shown in Figure 7-1, all the scan cells are controllable and observable.

Scan insertion results in design overheads such as, the use of extra scan ports, an increase in silicon area due to use of scan flops, and greater timing delays due to the insertion of the scan cells for the sequential non-scan cells.

After inserting the test scan logic in the design, the ATPG algorithm is used to generate test patterns.

7.2. IO Timing Constraints

Atmel Flash	
DoutSPI Setup Time (min) wrt ↑ of SPIClk_Out	5 ns
DoutSPI Hold Time (min) wrt ↑ of SPIClk_Out	10 ns
DinSPI Valid Time (max) wrt ↓ of SPIClk_Out	20 ns

Table 7-1 IO timing constraints for ATMEL flash memory

ST Flash	
DoutSPI Setup Time (min) wrt ↑ of SPIClk_Out	5 ns
DoutSPI Hold Time (min) wrt ↑ of SPIClk_Out	5 ns
DinSPI Valid Time (max) wrt ↓ of SPIClk_Out	15 ns

Table 7-2 IO timing constraints for ST flash memory

Timing values are at the flash interface. Board delays are not taken into account.

7.3. Synthesis Results

Schematic view of synthesized SPIClockLogic module is shown in Figure 7-2.

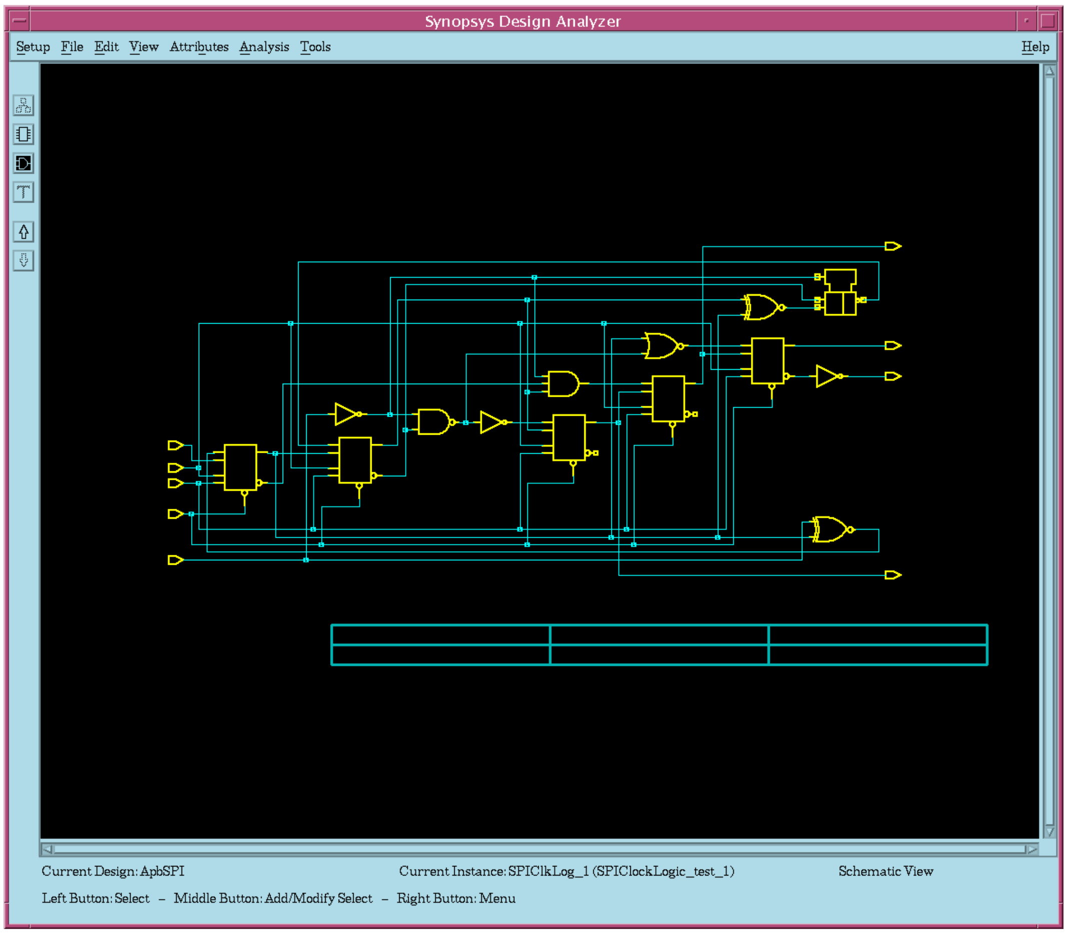

Synopsys Design Analyzer
Setup File Edit View Attributes Analysis Tools Help
Current Design: ApbSPI
Current Instance: SPIClkLog_1 (SPIClockLogic_test_1)
Schematic View
Left Button: Select – Middle Button: Add/Modify Select – Right Button: Menu

Figure 7-2 Synthesised SPIClockLogic block

Schematic view of synthesized RxFIFO module is shown in Figure 7-3.

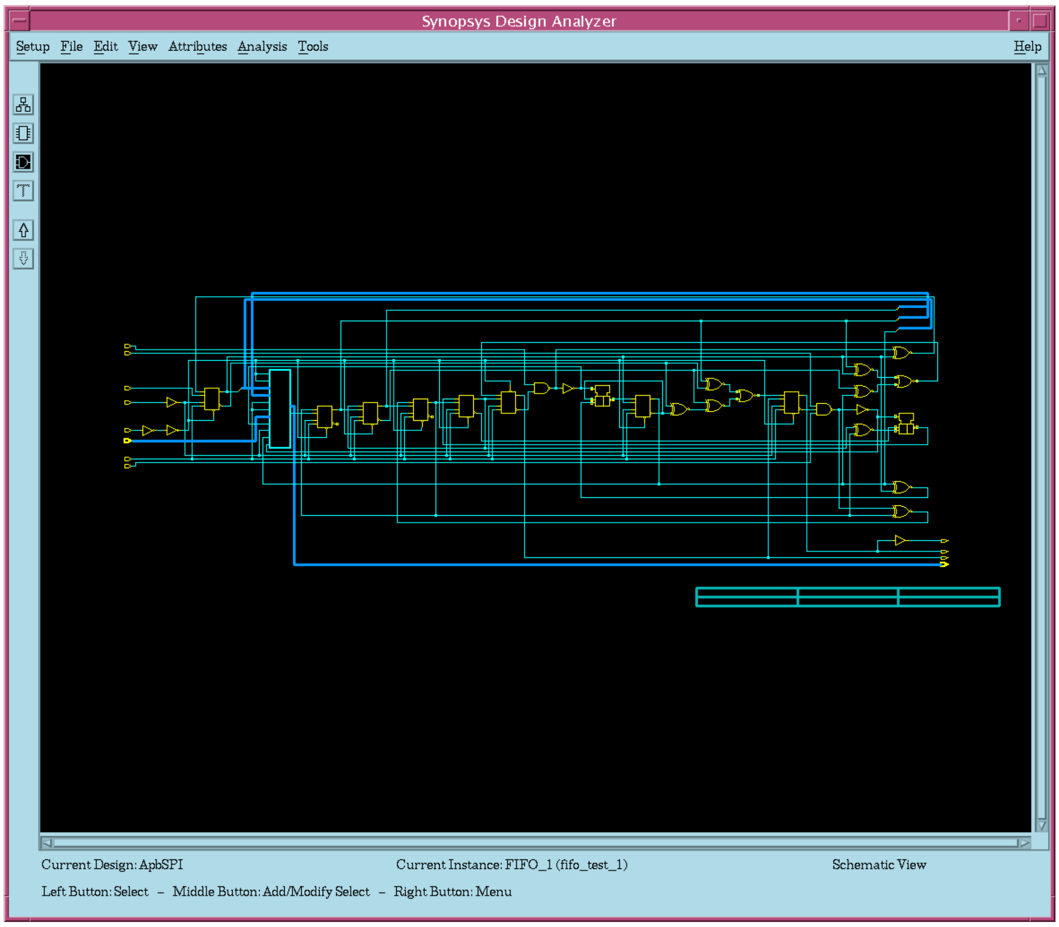

Synopsys Design Analyzer
Setup File Edit View Attributes Analysis Tools
Help
Current Design: ApbSPI
Current Instance: FIFO_1 (fifo_test_1)
Schematic View
Left Button: Select – Middle Button: Add/Modify Select – Right Button: Menu

Figure 7-3 Synthesised RxFIFO block

80 MHz system clock frequency is used in synthesis of serial flash memory controller module. The system has not any problem at this operation frequency in terms of critical timing issues and we do not get any violations.

Results of area achieved for this synthesis can be seen in Table 7-3.

Block		ApbIf	SPIIf	SPIClockLogic	RxFIFO	ApbSPI (Top Level)
Combinational Area	m^2	26608.39	357138.59	709.79	46792.19	432668.59
	Gates	487.3	6540.99	12.99	856.99	7924.3
Noncombinational Area	m^2	48175.4	111839	2093	70343	232450.4
	Gates	882.33	2048.33	38.33	1288.33	4257.33
Net Interconnect Area	m^2	5850	56772	198	306	132498
	Gates	107.14	1039.78	3.62	5.6	2426.7
Total Cell Area	m^2	74783.79	468977.59	2802.79	117135.2	665119
	Gates	1369.67	8589.33	51.33	2145.33	12181.7
Total Area	m^2	80633.79	525749.62	3000.79	117441.2	797617
	Gates	1476.8	9629.11	54.96	2150.93	14608.37

Table 7-3Area report

Power estimation reports given by Synopsys Design Analyzer for operating frequency is shown in Table 7-4.

Operating Conditions		WORST				
Global Operating Voltage (V)		3.3				
Library		csx_3.3V				
Power Consumption Estimation						
Block		ApbIf	SPIIf	SPIClockLogic	RxFIFO	ApbSPI (Top level)
Cell Internal Power	**mW**	2.69	7.26	0.29	1.86	20.09
Net Switching Power	**mW**	6.15	7.6	0.16	3.6	51.03
Total Dynamic Power	**mW**	8.84	14.85	0.45	5.45	71.12
Cell Leakage Power (Static Power)	**nW**	80.03	493.41	3.15	82.64	696.35

Table 7-4 Power report

7.4. Gate-level Simulations

The netlist of the synthesised Serial Flash Memory Controller module was saved in verilog format and then sdf (standard delay file) file was generated for the gate-level simulations. This sdf file is generated by Synopsys Design Analyzer and it includes the estimated timings for each of library elements.

In gate level simulations verilog model of NexFlash serial flash memory was used. There is no special reason to select NexFlash serial flash memory for gate level simulations. It was selected as a representative memory.

The waveforms for the gate level simulations are given in Appendix C.

8. PLACE & ROUTE FOR SERIAL FLASH MEMORY CONTROLLER

Cadence Silicon Ensemble is used for place&route.

8.1. Floorplanning and Placement

As it is said in Smith’s book, the input to a floorplanning tool is a hierarchical netlist that describes the interconnection of the blocks, the logic cells within the blocks, and the logic cell connectors [3]. The netlist is a logical description of the ASIC; the floorplan is a physical description of an ASIC. Floorplanning is thus mapping between the logical description (the netlist) and the physical description (the floorplan).

Floorplanning step helps to define the dimensions of the chip layout and place modules of the design in specific regions. At this stage of the design flow, this is a very rough estimate of the actual placement. Also, no actual routing is done at this stage. Floorplanning allows us to predict interconnect delay by estimating interconnect length.

For floorplanning of serial flash memory controller module, the aspect ratio was set to be 1, which was square. The distance of the IO to core was selected as 100 μm. This distance was large enough to leave room to route power and ground wires. Flip every other row and abut rows options were selected to create rows where VDD and GND alternates. Block halo per side, the distance between blocks and the rows, was set to be 20 μm. Row Utilization factor was selected as 90%. It indicates how densely Silicon Ensemble will pack each row. Higher numbers give you more dense designs, but smaller numbers will make it run faster.

After completing floorplan, placement of the logic cells begins. After floorplanning and placement, we have more accurate estimates of the capacitive loads that each logic cell must drive. The goal of a placement tool is to arrange all the logic cells within the flexible blocks on a chip. Ideally, the objectives of the placement step are:

- Guarantee the router can complete the routing step.
- Minimize all the critical net delays.
- Make the chip as dense as possible.

First step of placement of serial flash memory controller module was to place the IO cells automatically. After this power planning was started. Power rings surrounding all cells were generated and connected to power pads. Metal1 and metal2 were used for the power ring. The width of core supply ring and channel were selected as 10 μm and 5 μm. After the power routing was planned, the cells were placed inside the rows.

8.2. Routing

Once the designer has floorplanned a chip and the logic cells within the blocks have been placed, it is time to make the connections by routing the chip. Routing is split into global routing followed by detailed routing.

The input to the global router is a floorplan that includes the locations of all the fixed and flexible blocks; the placement information for flexible blocks; and the locations of all the logic cells. The goal of global routing is to provide complete instructions to the detailed router on where to route every net. The objectives of global routing are:

- Minimize the total interconnect length.
- Maximize the probability that the detailed router can complete the routing.
- Minimize the critical path delay.

The global routing step determines the channels to be used for each interconnect. Using this information, detailed router decides the exact location and layers for each interconnect. The goal of detailed routing is to complete all the connections between logic cells. . The objectives of detailed routing are:

- The total interconnect length and area.
- The number of layer changes that the connections have to make.
- The delay of critical paths.

First step of the routing of serial flash memory controller module was to connect all blocks to the power rings in the design. Block and allport options were selected in this step. Block option connects pins to the closest rings. Allport option connects all ports of the pins to the closest rings; otherwise only one port is connected. After connecting rings, final and global routing was started. After routing, the gaps between the cells were closed by placing filler cells.

The script, used in backend flow, is given in Appendix D.

8.3. Place&Route Results

The number of components, models, pins, nets, instances of each model, and the routing tracks available for our serial flash memory controller block are given below.

Number of macros	190
Number of components	6527
Number of pins	28412
Number of regular pins	15333
Number of special pins	8344
Number of unused pins	215
Number of nets	4585
Average number of pins per net	6.15
Number of subnets	0
Number of routing tracks available	1439
Number of GCELLS per layer	5250

Table 8-1 Design summary

Rows are locations for the placement of cells, either horizontally or vertically. A row is a one-dimensional array of uniform placement grids in which to place of a matching type. For maximum chip area utilization, the cells are placed as close as possible to each other within each row and not violating design rules. Utilization of rows for our design is given below.

Type	Number	Length	Area	% Row Space
Standard Rows	59	4551260	5916638000	
Standard Cells	6432	4551260	5916638000	100.00

Table 8-2 Utilization of rows

Area of chip: 9431322600 (square DBU)

Area required for all cells: 5916638000 (square DBU)

Area utilization of all cells: 62.73%

Total layers: 7

Routing layers: 3

The area estimated after the synthesis was about 0.8 mm^2. The final area at the end of the place&route is 0.94 mm^2. It is 17.5 % larger than the expected one. It's not a big difference. Moreover, since we don't have any routing error, we can still reduce the area.

Layer information given in Table 8-3.

Layer	Routing	Process Order	Routing Order
MET1	horizontal	6	3
MET2	vertical	4	2
MET3	horizontal	2	1
VIA	can't route	5	
VIA2	can't route	3	
OVERLAP	can't route	1	
VIRTUAL	can't route	7	

Table 8-3 Layer information

Wiring information is given below.

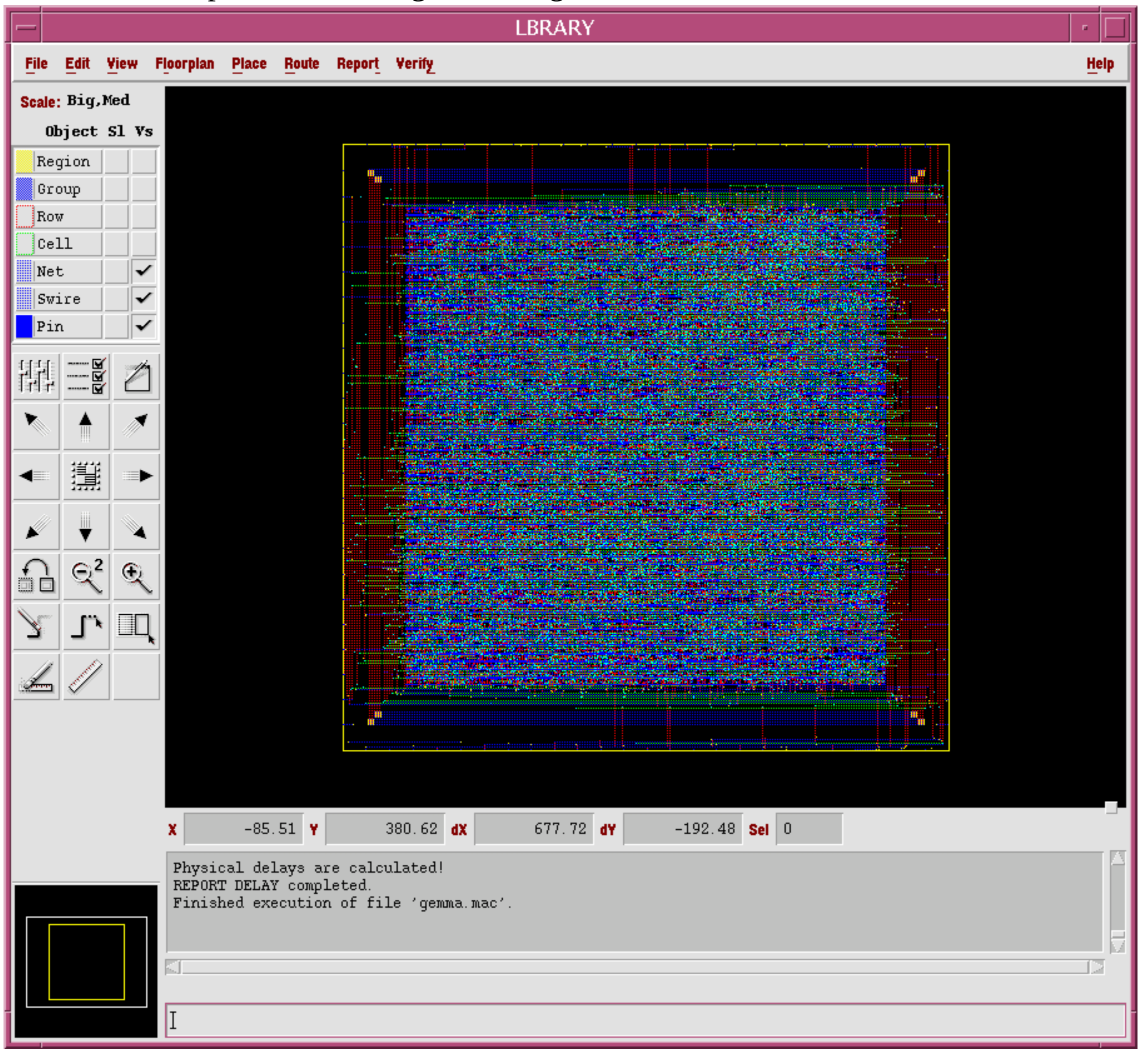
Total vias in regular wiring: 19475
Total segments in regular wiring: 38897
Total vias in special wiring: 154
Total segments in special wiring: 227

Wire lengths for different layers given in Table 8-4.

	MET1 (microns)	MET2 (microns)	MET3 (microns)	Total (microns)
Length of regular wires	40489.40	257515.70	288972.00	586977.10
Length of special wires	95737.60	4960.80	.00	100698.40
Length of regular and special wiring	136227.00	262476.50	288972.00	687675.50

Table 8-4 Wire lengths

The result of place& route is given in Figure 8-1.



```
Physical delays are calculated!
REPORT DELAY completed.
Finished execution of file 'gemma.mac'.
```

Figure 8-1 Result of place&route

8.4. Post-layout Simulations

Verilog netlist of Serial Flash Memory Controller module and sdf file was exported at the end of place&route. These files were used in post-layout simulations. Since NexFlash serial flash memory model was used in gate level simulations, we used it also for post-layout simulations. Also the testbench is the one that was used in gate level tests.

9. FPGA IMPLEMENTATION

The serial flash memory controller module was implemented to FPGA for final verification. XS40 FPGA board and M25P80 serial flash memory were used in this application.

9.1. XS40 board

The arrangement of components for XS40 board is shown in Figure 9-1.

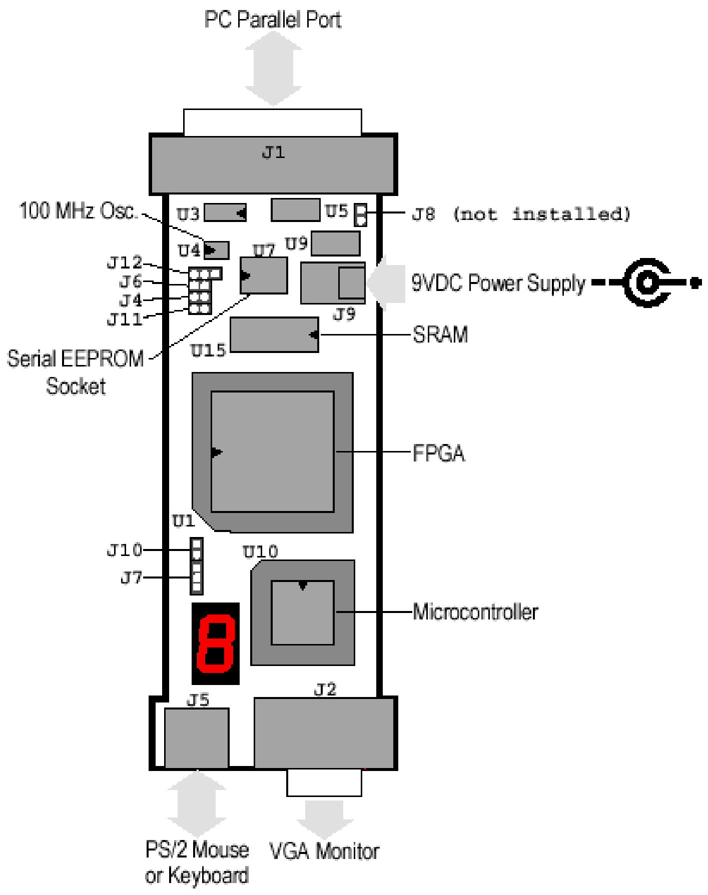


Figure 9-1 Arrangement of components on XS40 board

XS40 board can be connected to PC from the parallel port. It has GXSLOAD, GXSPORT, GXSTEST and GXSSETCLK utility programs.

XS40 board has a 100 MHz programmable oscillator (a Dallas Semiconductor DS1075Z-100). The 100 MHz master frequency can be divided by factors of 1, 2, ... up to 2052 to get clock frequencies of 100 MHz, 50 MHz, ... down to 48.7 KHz, respectively. The divided frequency is sent to the rest of the XS40 board circuitry as a clock signal. In our FPGA implementation, we programmed oscillator to get clock frequency of 25 MHz. The divisor was stored into the oscillator chip by using the GUI-based GXSSETCLK utility.

The FPGA used in XS40 board is XC4010XL with 84 pin.

Xilinx ISE 4.2 was used in our FPGA application. This program integrates all Foundation tools into a unified environment. The final result of the project is a bit stream file that can be downloaded into a reprogrammable FPGA device.
After the design was implemented to FPGA, the generated bit stream file was downloaded from PC into XS40 board by using GXSLOAD utility.

The microcontroller and the RAM inside the XS40 board are not used in our application. For this reason RAM and microcontroller are held in an inactive state. The microcontroller is held in the RESET state by placing a high level on its RST pin. The RAM is deactivated by placing a high level on its /CS input. We used FPGA pins attached to the RAM and microcontroller as general-purpose I/O. Pin assignments of FPGA for our application is given in Table 9-1.

Port Name	Location
Clock	p13
ResetNot	p44
DinSPI	p3
SPInCS	p4
SPIClk_Out	p5
DoutSPI	p6
FlashWP	p7
FlashHold	p8
UCRst	p36
SRAMcs	p65
Led<0>	p19
Led<1>	p23
Led<2>	p26
Led<3>	p25
Led<4>	p24
Led<5>	p18
Led<6>	p20

Table 9-1 Pin assignments for FPGA

XS40 board pin descriptions and its detailed schematic given in Appendix E.

9.2. Test Scenario for FPGA Implementation

To verify the serial flash memory controller module, the block named as ApbGen was designed. ApbGen block generates APB signals according to the test scenario. The block diagram for the top level, which integrates ApbGen and ApbSPI, is given below.

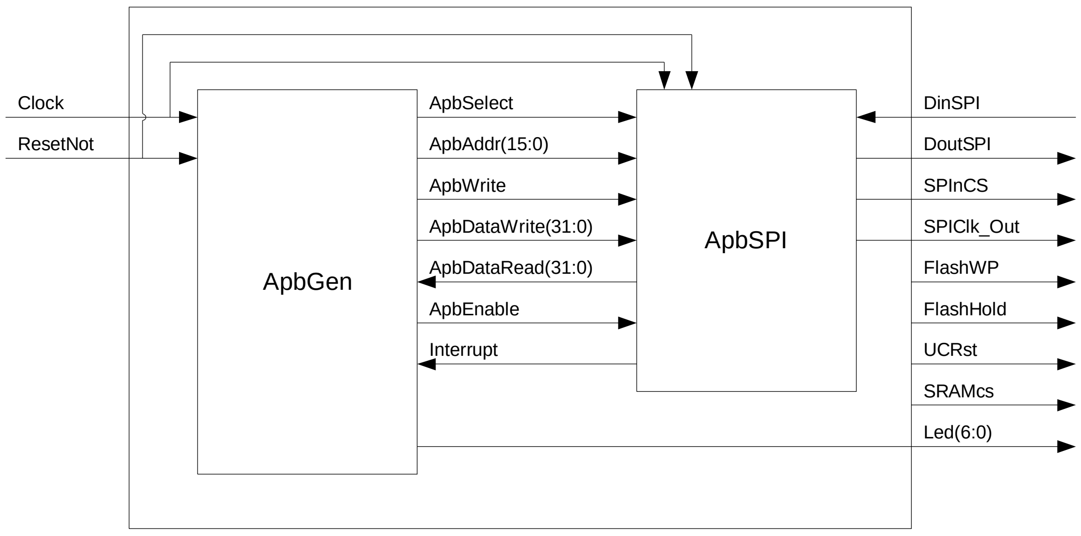


Figure 9-2 Block diagram for integration of ApbGen and ApbSPI

The state machine controlling the operation of ApbGen block is shown in Figure 9-3

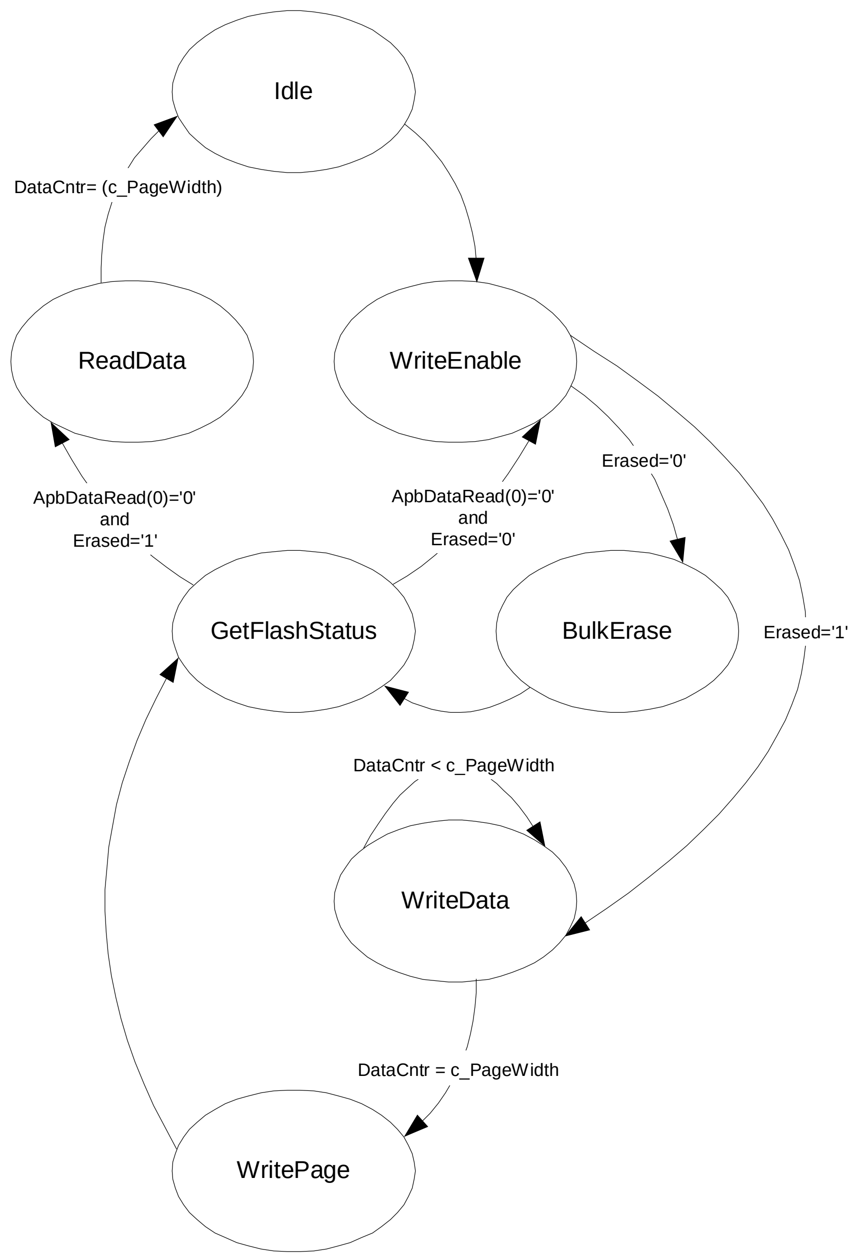


Figure 9-3 State machine controlling ApbGen

Erased signal is the input of FSM that controls ApbGen block. This signal is set to 1, when the BulkErase operation is finished by serial flash memory.

In WriteEnable State, ApbSPI block is programmed for WriteEnable operation.

In BulkErase State, if ApbSPI is not busy, it is programmed for BulkErase operation. There is a process inside ApbGen, to read the status register of ApbSPI when Interrupt input is high. So ApbGen can detect whether the ApbSPI block is busy or not.

In GetFlashStatus State, if ApbSPI is not busy, it is programmed for GetFlashStatus operation. After ApbSPI finishes GetFlashStatus operation, the status data is read from the FIFO. This will continue until the serial flash memory is ready for new operation.

In WriteData State, if ApbSPI is not busy, it is programmed for WriteData operation. The length of the data that is sent to serial flash memory is 256 byte. This length equals to the length of one page of serial flash memory used for verification. Because of the limited size of the FPGA that we used, we can only write one page of data to the serial flash memory.

In WritePage State, if ApbSPI is not busy, it is programmed for WritePage operation.

In ReadData State, ApbSPI is programmed for ReadData operation. ApbGen reads the status of ApbSPI when the Interrupt input is high. If there is data written into the FIFO, it will be read by ApbGen. ApbGen compares the received data with the data sent to serial flash memory. If there is any difference between these, it sets Error_Detected signal.

After all the data is read, if Error_Detected signal is high, ApbGen writes E to seven-segment display on the XS40 board. Otherwise it writes C to seven-segment display. This means that the data received is correct.

A simple pull up resistor (10 kohm) was connected to chip select pin of flash memory to insure safe and proper power up and power down. Also 0.1 uF capacitor was connected between Vcc and Gnd of serial flash memory to stabilise the Vcc feed [7].

The waveforms for RTL simulations given in Appendix F.

9.3. FPGA Implementation Results

Device utilization summary for FPGA implementation is given in Table 9-2.

Number of External IOBs	16 out of 61 26%
Flops	7
Latches	0
Number of Global Buffer IOBs	1 out of 8 12%
Flops	0
Latches	0
Number of CLBs	400 out of 400 100%
Total Latches	0 out of 800 0%
Total CLB Flops	438 out of 800 54%
4 input LUTs	728 out of 800 91%
3 input LUTs	157 out of 400 39%
Number of BUFGLSs	2 out of 8 25%
Number of STARTUPs	1 out of 1 100%

Table 9-2 Device utilization summary

The main problem while implementing the design to the FPGA is the number of CLBs. To fit the design into the FPGA some of the functionalities were given up. This version of design can only work with ST serial flash memories. Also it has only one chip select output.

10. CONCLUSIONS

This thesis has presented the SPI protocol and the design, digital implementation, functional and gate-level verification, synthesis and place&route of serial flash memory controller block in digital CMOS 0.35μm technology.

While designing serial flash memory controller block, our main purpose was to create easy programmable, easy adaptable module with minimum area. For this reason we made research on current solutions for communication with serial flash memories, SPI protocol and structure of processor platform.

The design consists of four main blocks:

- ApbIf block: APB interface that provides communication with the processor.
- SPIIf block: SPI interface, which provides communication with serial flash memory.
- SPIClockLogic: Produces clock and enable signals for communication with serial flash memory.
- RxFIFO: FIFO for continuos operation in receive mode.

Serial flash memory controller block can be programmed by the processor for the required operation and the structure of receive or transmit frame. This block can easily be integrated into the processor platform for embedded applications. Since the serial flash memory controller block has a generic structure, it can be used with different types of flash memories.

The synthesis of serial flash memory controller block was done for 80 MHz clock speed. SPIClockLogic block produces 20 MHz SPI clock from this 80 MHz clock.

Serial flash memory controller block is tested and verified for all flash memory access operations. All the tests were done from the top level by using model of processor and flash memories.

REFERENCES

1. ARM, *PrimeCell Synchronous Serial Port (PL022) Technical Reference Manual*, July 2001.
2. ARM, *AMBA Specification (Rev 2.0)*, May 1999.
3. Michael John Sebastian Smith, *Application Specific Integrated Circuits*, 1997.
4. Pran Kurup, Taher Abbasi, *Logic Synthesis Using Synopsys (Second Edition)*, 1997.
5. Atmel, *64-megabit 2.7-volt Only Dual-interface Data Flash AT45DB642*, Rev. 1638D–11/01.
6. Atmel, *32-Bit Embedded Core Peripheral, Serial Peripheral Interface (SPI)*, Rev. 1244D-CASIC–01/03.
7. STMicroelectronics, *8 Mbit, Low Voltage, Serial Flash Memory with 25 MHz SPI Bus Interface M25P80*, December 2002.
8. NexFlash Technologies, *63M-bit Serial Flash Memory with 4 Pin SPI Interface NX25F641C*, April 2002.
9. Fred G. Martin, *D.4 Serial Peripheral Interface*, November 1995.
10. Motorola, *MC68HC11 Reference Manual*, Prentice Hall 1989.
11. Ohio State University, Information and Electronics Group, *The Serial Peripheral Interface, Nautilus Chip-Project DEEPSEA (Digital Exportation of an Established Protocol from Sensing Encoded Analog)*, January 2001.
12. Andrew Chu, Chris Ohlmann, VHDL Implementation of Serial Peripheral Interface, 2002.
13. Cadence, *Serial Peripheral Interface Technical (SPI) Technical Data Sheet*, December 2002.
14. Palmchip, *Serial Peripheral Interface Controller*, January 2002.

15. ST Microelectronics, *SPI Communication Between ST7 and EEPROM*, 1999.

16. Xilinx, *CoolRunner Serial Peripheral Interface Master*, December 2002.

17. Alma Technologies, *SPI Master/Slave Core*, 2002.

18. Dave Bursky, *Serial Flash Memories Rise To Meet Changing System Needs*, Marc 1999.

19. Brett Glass, *There in a Flash: Flash Memory for Embedded Systems*, CMP Media Inc. 2000.

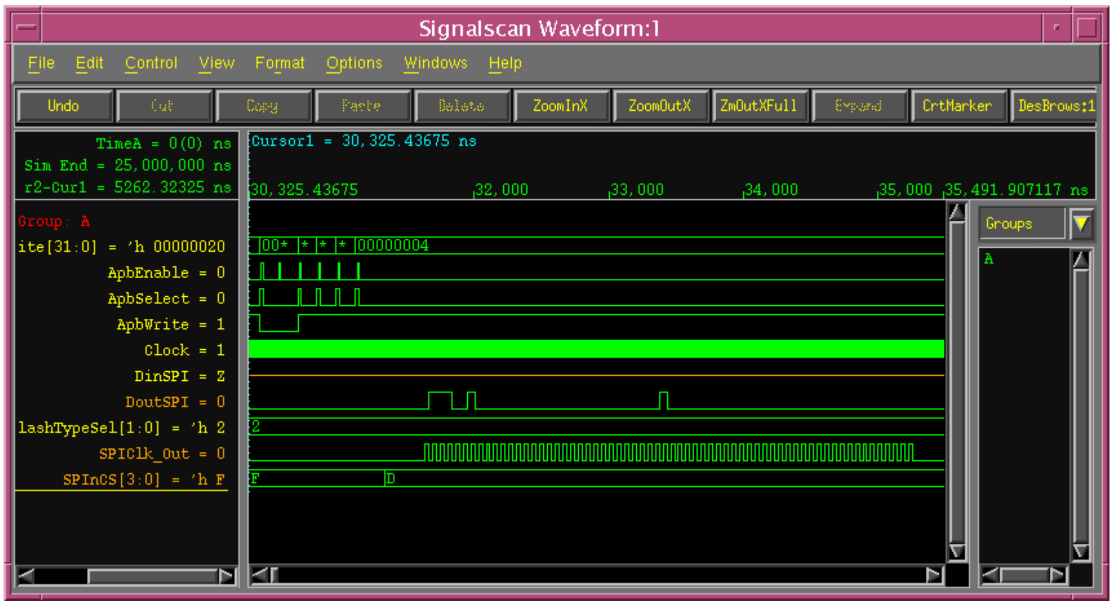
20. ST Microelectronics, *Flash Memories*, B979M - June 2003.

21. ST Microelectronics, *Using Serial Flash Memories for Code Storage in Computer and Peripherals Applications*, May 2002.

22. XESS Corporation, *XS40, XSP Board V1.4 XS40, XSP Board V1.4 XS40, XSP Board V1.4 XS40, XSP Board V1.4 User Manual,* 9/21/2001.

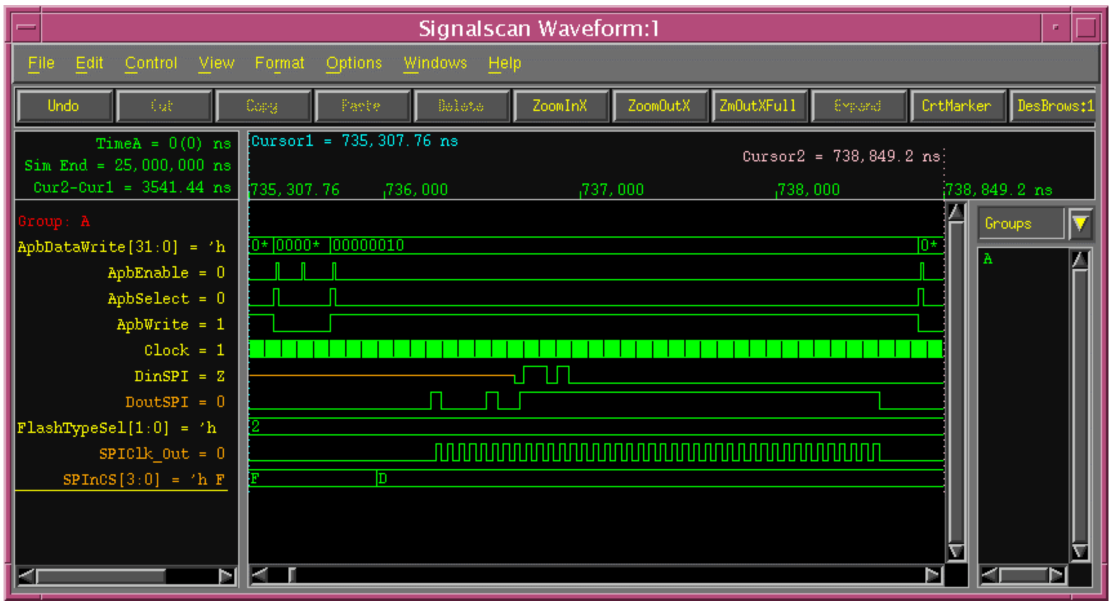
11. APPENDIX A: FUNCTIONAL SIMULATIONS

Below figures shows write and read operations for NexFlash flash memories.



```
TimeA = 0(0) ns
Sim End = 25,000,000 ns
r2-Cur1 = 5262.32325 ns
Cursor1 = 30,325.43675 ns
30,325.43675  32,000  33,000  34,000  35,000  35,491.907117 ns
Group: A
ite[31:0] = 'h 00000020     00*  *  *  *  00000004
ApbEnable = 0
ApbSelect = 0
ApbWrite = 1
Clock = 1
DinSPI = Z
DoutSPI = 0
lashTypeSel[1:0] = 'h 2     2
SPIClk_Out = 0
SPInCS[3:0] = 'h F          F  D
```

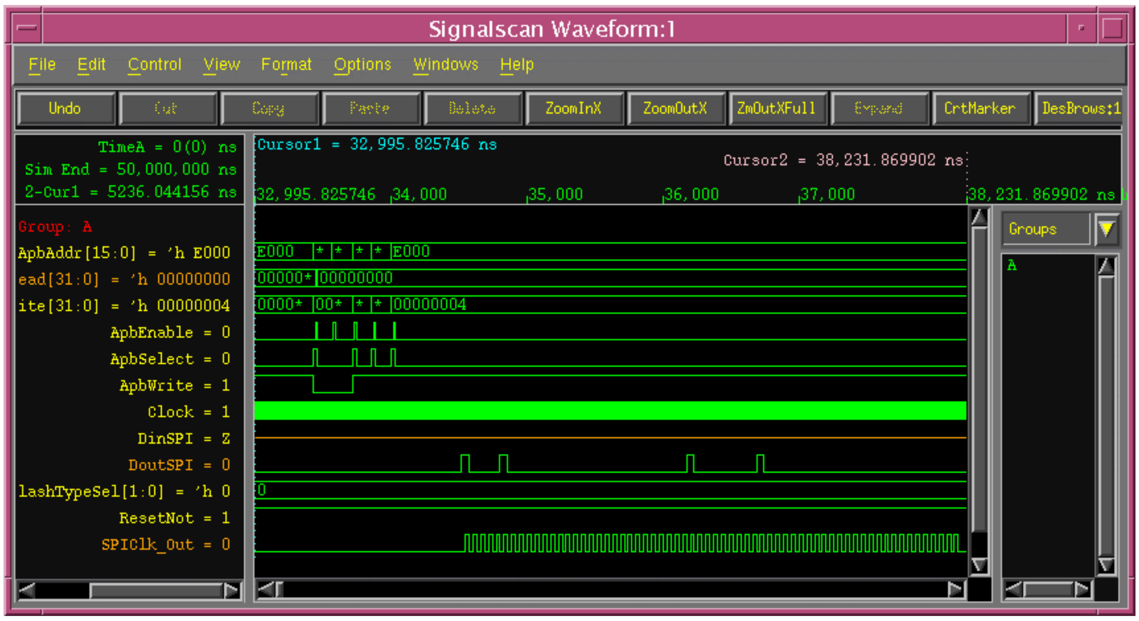
Figure 11-1 Write operation for NexFlash serial flash memory



```
TimeA = 0(0) ns
Sim End = 25,000,000 ns
Cur2-Cur1 = 3541.44 ns
Cursor1 = 735,307.76 ns
Cursor2 = 738,849.2 ns
735,307.76  736,000  737,000  738,000  738,849.2 ns
Group: A
ApbDataWrite[31:0] = 'h     0*  0000*  00000010  0*
ApbEnable = 0
ApbSelect = 0
ApbWrite = 1
Clock = 1
DinSPI = Z
DoutSPI = 0
FlashTypeSel[1:0] = 'h      2
SPIClk_Out = 0
SPInCS[3:0] = 'h F          F  D
```

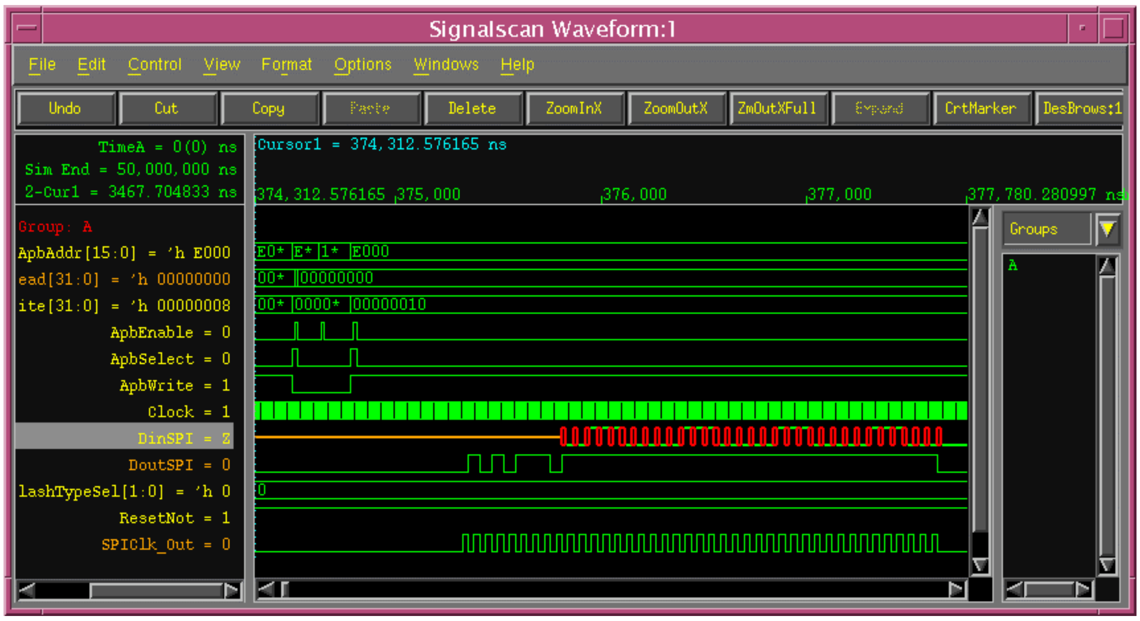
Figure 11-2 Read operation for NexFlash serial flash memory

Below figures shows read and write operations for Atmel flash memories.



```
TimeA = 0(0) ns
Sim End = 50,000,000 ns
2-Cur1 = 5236.044156 ns
Cursor1 = 32,995.825746 ns
Cursor2 = 38,231.869902 ns
32,995.825746  34,000  35,000  36,000  37,000  38,231.869902 ns
Group: A
ApbAddr[15:0] = 'h E000        E000 * * * * E000
ead[31:0] = 'h 00000000        00000* 00000000
ite[31:0] = 'h 00000004        0000* 00* * * 00000004
ApbEnable = 0
ApbSelect = 0
ApbWrite = 1
Clock = 1
DinSPI = Z
DoutSPI = 0
lashTypeSel[1:0] = 'h 0        0
ResetNot = 1
SPIClk_Out = 0
```

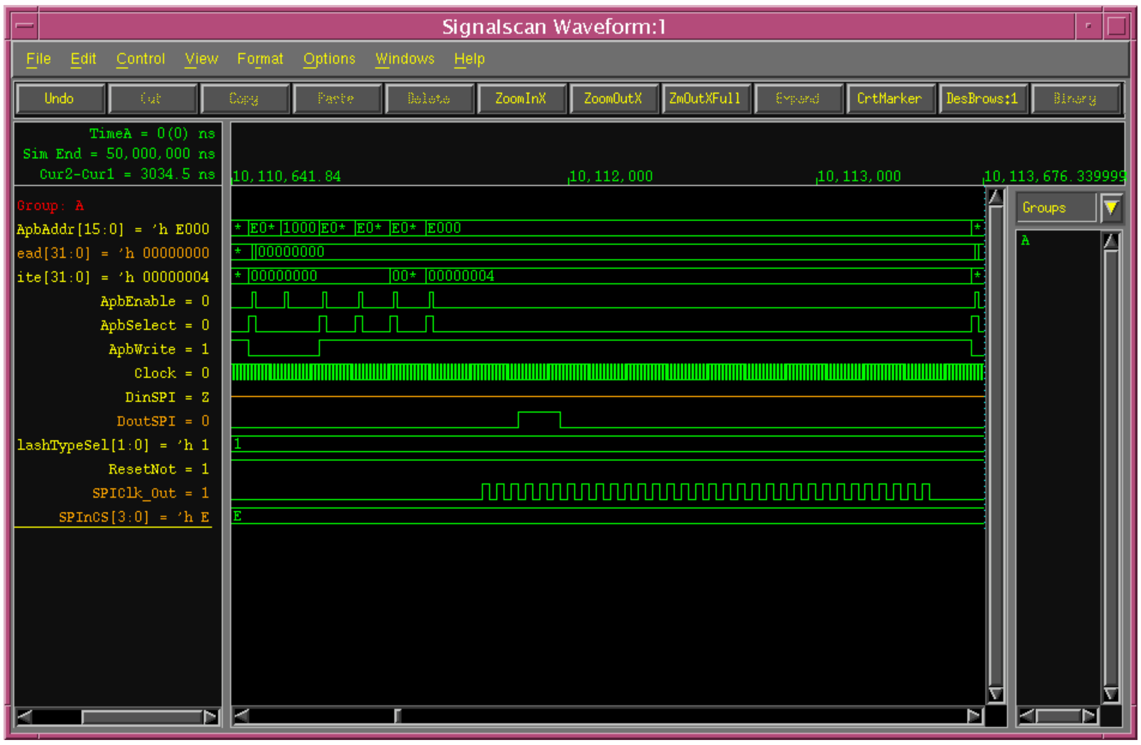
Figure 11-3 Write operation for Atmel serial flash memory



```
TimeA = 0(0) ns
Sim End = 50,000,000 ns
2-Cur1 = 3467.704833 ns
Cursor1 = 374,312.576165 ns
374,312.576165  375,000  376,000  377,000  377,780.280997 ns
Group: A
ApbAddr[15:0] = 'h E000        E0* E* 1* E000
ead[31:0] = 'h 00000000        00* 00000000
ite[31:0] = 'h 00000008        00* 0000* 00000010
ApbEnable = 0
ApbSelect = 0
ApbWrite = 1
Clock = 1
DinSPI = Z
DoutSPI = 0
lashTypeSel[1:0] = 'h 0        0
ResetNot = 1
SPIClk_Out = 0
```

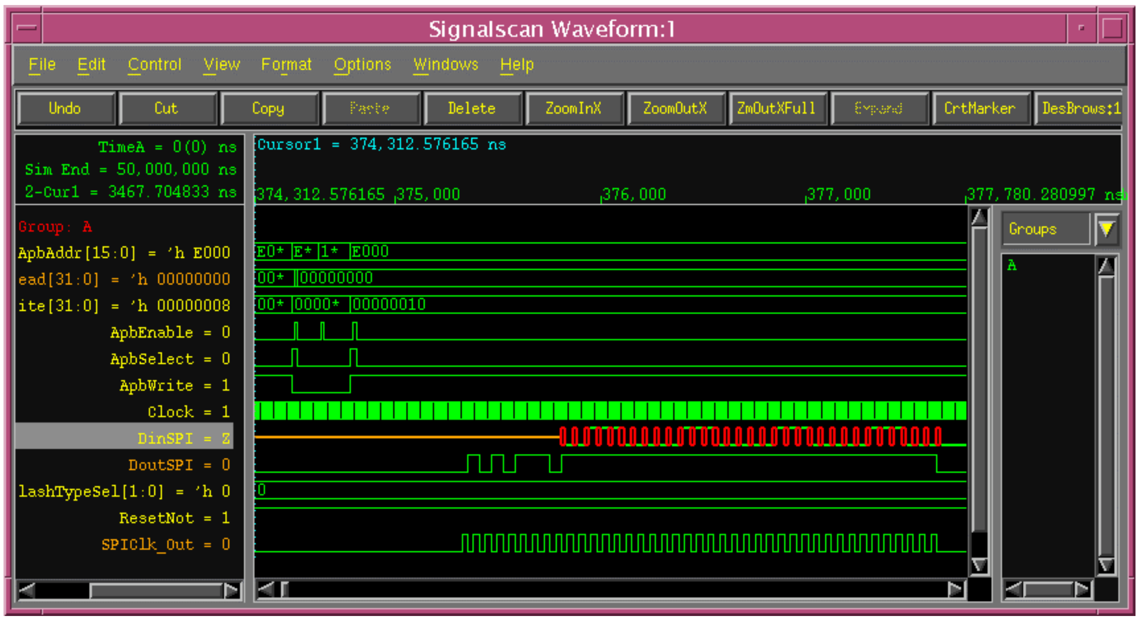
Figure 11-4 Read operation for Atmel serial flash memory

Below figures read and write operations for ST flash memories.


Signalscan Waveform:1
File Edit Control View Format Options Windows Help
Undo Cut Copy Paste Delete ZoomInX ZoomOutX ZmOutXFull Expand CrtMarker DesBrows:1 Binary
Groups
A

```
TimeA = 0(0) ns
Sim End = 50,000,000 ns
Cur2-Cur1 = 3034.5 ns
10,110,641.84   10,112,000   10,113,000   10,113,676.339999
Group: A
ApbAddr[15:0] = 'h E000
ead[31:0] = 'h 00000000
ite[31:0] = 'h 00000004
ApbEnable = 0
ApbSelect = 0
ApbWrite = 1
Clock = 0
DinSPI = Z
DoutSPI = 0
lashTypeSel[1:0] = 'h 1
ResetNot = 1
SPIClk_Out = 1
SPInCS[3:0] = 'h E
```

Figure 11-5 Write operation for ST serial flash memory


Signalscan Waveform:1
File Edit Control View Format Options Windows Help
Undo Cut Copy Paste Delete ZoomInX ZoomOutX ZmOutXFull Expand CrtMarker DesBrows:1
Groups
A

```
TimeA = 0(0) ns
Sim End = 50,000,000 ns
2-Cur1 = 3467.704833 ns
Cursor1 = 374,312.576165 ns
374,312.576165   375,000   376,000   377,000   377,780.280997 ns
Group: A
ApbAddr[15:0] = 'h E000
ead[31:0] = 'h 00000000
ite[31:0] = 'h 00000008
ApbEnable = 0
ApbSelect = 0
ApbWrite = 1
Clock = 1
DinSPI = Z
DoutSPI = 0
lashTypeSel[1:0] = 'h 0
ResetNot = 1
SPIClk_Out = 0
```

Figure 11-6 Read operation for ST flash memory

12. APPENDIX B: SYNTHESIS SCRIPT

To synthesise ApbSPI following scripts should be run.

```
dc_shell -f analyze.scr
dc_shell -f ApbSPI.rtl.script
```

In the script called analyze.scr all the hdl files included in ApbSPI database are analyzed.

ApbSPI.rtl.script is given below.

```
elaborate ApbSPI -arch "STR" -lib LIB_APB_SPI -update
set_scan_configuration -methodology full_scan
set_scan_configuration -style multiplexed_flip_flop

OPERATING_CONDITIONS = "WORST"
REF_DRIVER_PIN ="csx_HRDLIB/DFA/Q"
REF_DRIVER_CELL ="csx_HRDLIB/DFA"
REF_LOAD ="csx_HRDLIB/NA2/A"
DEFAULT_MAX_TRANSITION =2.6
CLOCK_NAME = "Clock"
CLOCK_PERIOD = "11.00"
CLOCK_PERIOD_1 = "40.00"
RESET_NAME = "ResetNot"
DEFAULT_INPUT_DELAY = "1"
DEFAULT_OUTPUT_DELAY = "5"
DEFAULT_LOAD = "1"
DEFAULT_MAX_CAPACITANCE = 5.0 * load_of(REF_LOAD)
DEFAULT_DRIVE = drive_of(REF_DRIVER_PIN)

create_clock CLOCK_NAME -period CLOCK_PERIOD
set_input_delay DEFAULT_INPUT_DELAY -clock CLOCK_NAME all_inputs()
remove_input_delay CLOCK_NAME -clock CLOCK_NAME
set_max_capacitance DEFAULT_MAX_CAPACITANCE all_inputs()
remove_attribute CLOCK_NAME max_capacitance
set_max_transition DEFAULT_MAX_TRANSITION find(design,"*")
set_load DEFAULT_LOAD all_outputs()
MAX_AREA_CONSTRAINT = "0.0"
set_max_area MAX_AREA_CONSTRAINT
set_operating_conditions OPERATING_CONDITIONS
set_fix_multiple_port_nets –all
```

```
compile -scan
write -hierarchy -output ApbSPI.db
write -hierarchy -format verilog -output ApbSPI.GAT.v
write_constraints -format sdf -output ApbSPI.sdf
report_area                    > ApbSPI.rtl.reports
report_timing                 >> ApbSPI.rtl.reports
report_reference              >> ApbSPI.rtl.reports
check_design                  >> ApbSPI.rtl.reports
report_constraints -verbose   >> ApbSPI.rtl.reports
report_hierarchy              >> ApbSPI.rtl.reports
report_port                   >> ApbSPI.rtl.reports
quit
```

13. APPENDIX C: GATE-LEVEL SIMULATIONS

Below figures show the results of gate level simulations for NexFlash serial flash memories.

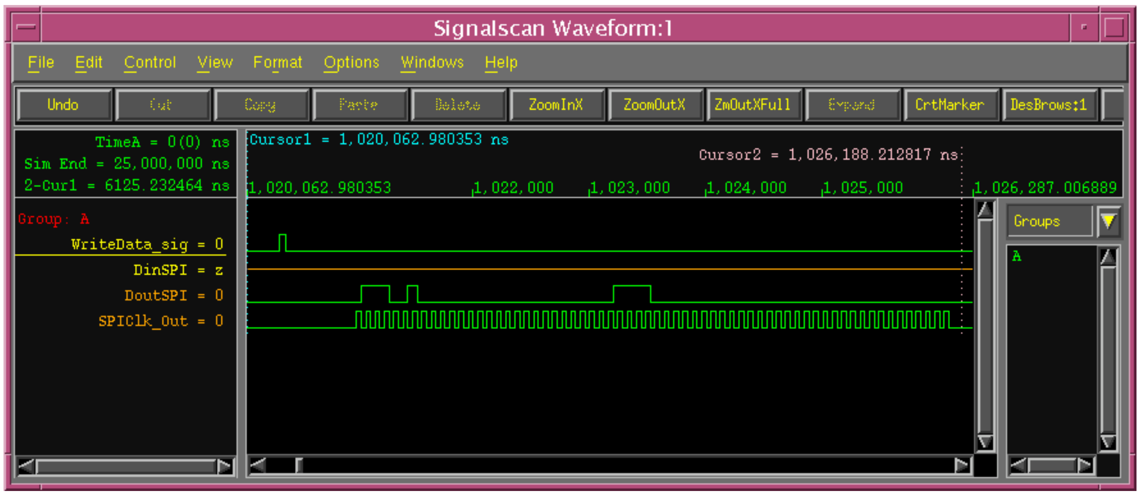


Figure 13-1 WriteData operation for NexFlash serial flash memory

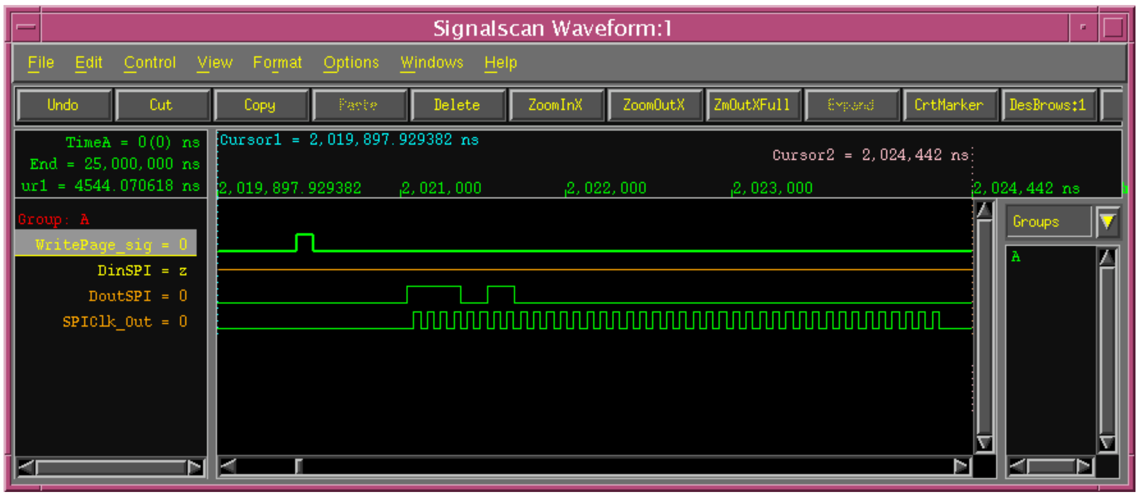


Figure 13-2 WritePage operation for NexFlash serial flash memory

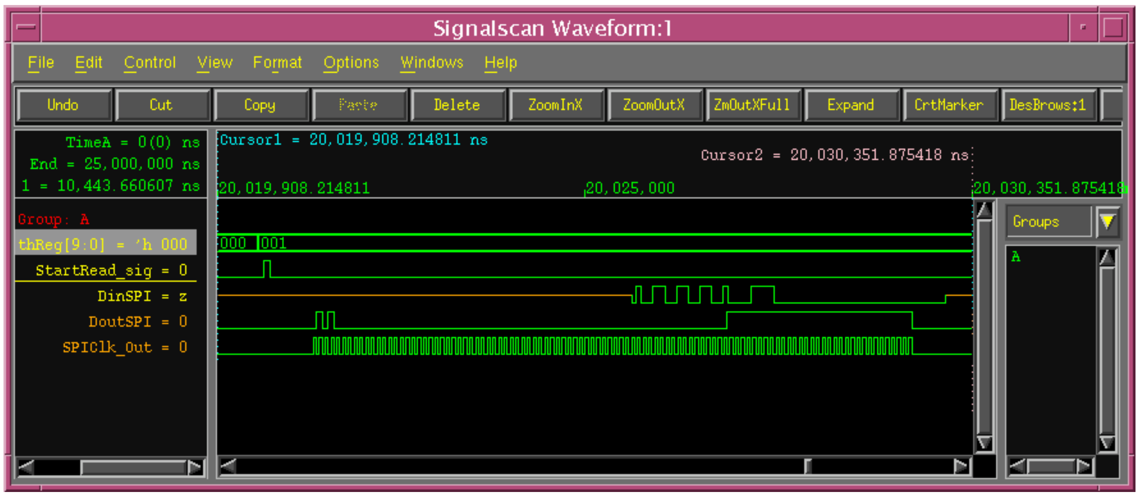


Figure 13-3 Read Data operation for NexFlash serial flash memory

14. APPENDIX D: BACKEND SCRIPT

```
##-------------------------------------------------------
##--
##--  Silicon Ensemble Macro File Template
##--
##-------------------------------------------------------

##-- set colors to make vias visible
##--
set v DRAW.SWIRE.LAYERSET      "1 2 3 4 5 6" ;
set v DRAW.WIRE.LAYERSET       "1 2 3 4 5 6" ;

set v DRAW.SWIRE.4.COLOR 5 ;
set v DRAW.SWIRE.5.COLOR 6 ;
set v DRAW.WIRE.4.COLOR 5 ;
set v DRAW.WIRE.5.COLOR 6 ;

set v DRAW.LAYER.ORDER "4 1 2 3 5 6";

##-- Set Off Congestion Map Drawing
SET VAR DRAW.SCORE.GRAPHICS.AT OFF ;

##-- Set Design Directory
##--
SET VAR DB.DESIGN.DIR "./DB" ;
SET VAR VERIFY.TECHNOLOGY.MIN.FEATURESIZE 5 ;
SET VAR SROUTE.VIA.SNAPMANUFACTURINGGRID TRUE ;
SET VAR SROUTE.STRIPE.SNAP.RGRID "GRID" ;
SET VAR HYPEREXTRACT.RULES.FILE
"/usr/local/cds/lib/ams_v3.40/artist/HK_0.35/LEF/csd/csd_he.rules";

##-- Set Variables for VERILOG Import
##--
SET VAR INPUT.VERILOG.CREATE.IO.PINS FALSE ;
SET VAR INPUT.VERILOG.ADD.LEADING.DELIM FALSE ;
set var INPUT.VERILOG.GROUND.NET  "gnd! gnd3r! gnd3o!" ;
set var INPUT.VERILOG.POWER.NET  "vdd! vdd3o! vdd3r!";
set var INPUT.VERILOG.SPECIAL.NETS "vdd! vdd3o! vdd3r! gnd! gnd3o! gnd3r!" ;
set var INPUT.VERILOG.LOGIC.0.NET gnd! ;
set var INPUT.VERILOG.LOGIC.1.NET vdd! ;

##-- Import Library Data
```

```
##-- LEF
FINPUT LEF F /usr/local/cds/lib/ams_v3.40/artist/HK_0.35/LEF/csd/csd.lef ;
INPUT LEF F /usr/local/cds/lib/ams_v3.40/artist/HK_0.35/LEF/csd/HRDLIB_3M.lef ;
INPUT LEF F /usr/local/cds/lib/ams_v3.40/artist/HK_0.35/LEF/csd/IOLIB_3M.lef ;
## INPUT LEF F <addional LEF files> ;

##-- CTLF Timing
##-- GCF File
INPUT CTLF INITFILE "./csd3.3V.gcf" ;

##-- Import Design Data
##-- Verilog
 INPUT VERILOG FILE ./VERILOG/csx_HRDLIB.v LIB DesignLib;
 INPUT VERILOG FILE ./VERILOG/ApbSPI.GAT.v LIB DesignLib
     REFLIB "DesignLib" DESIGN DesignLib.ApbSPI:hdl ;

##-- Import Timing Contraints
##-- INPUT GCF FILENAME "./constr3.3V.gcf" REPORTFILE "importgcf.rpt" ;

##-- To Set Rows On Grid
SET VAR PLAN.IOROW.SNAPGRID.X 10 ;
SET VAR PLAN.IOROW.SNAPGRID.Y 10 ;

##-- Save design
##--
SAVE loaded ;

##-- Initialize the floorplan
FINIT FLOORPLAN  rowu 0.90 rowsp 0 blockhalo 2000 f a 1 abut xio 10000 yio
10000 ;
WINDOW FIT ;
##-- Place the Periphery Cells
IOPLACE AUTOMATIC STYLE EVEN ;

##-- Cut Rows around Blocks
CUT ROW BLOCKHALO 2000;
##-- Power Routing
##--
BUILD CHANNEL ;

##-- Add Power Stripes
##--
#ADD STRIPE    NET vdd! NET gnd! DIRECTION Vertical LAYER MET2 WIDTH
500

##-- Add Power Rings
##--
CONSTRUCT RING NET "vdd!" NET "gnd!"
   LAYER MET1 CORERINGWIDTH 1000 SPACING CENTER
BLOCKRINGWIDTH 500
```

```
   LAYER MET2 CORERINGWIDTH 1000 SPACING CENTER
BLOCKRINGWIDTH 500;

SAVE power_plan ;
##-- Add Cap cells
SROUTE ADDCELL MODEL LCAP PREFIX lcap
    SPIN vdd! NET vdd! SPIN gnd! NET gnd!
    AREA ( -43500 -43480) ( 43640 43610 ) PREENDCAP ;
SROUTE ADDCELL MODEL RCAP PREFIX rcap
    SPIN vdd! NET vdd! SPIN gnd! NET gnd!
    AREA ( -43500 -43480 ) ( 43640 43610 ) POSTENDCAP ;

##-- Place Standard Cells
QPLACE NOCONFIG ;

SAVE qplaced ;
SET VAR SROUTE.LPR.VIASATCROSSOVER TRUE ;
SET VAR SROUTE.STACKVIASATCROSSOVER TRUE ;

##-- Finish Power Routing
##-- Connect Blocks
CONNECT RING NET "vdd!" NET "gnd!" BLOCK ALLPORT ;

##-- Connect Power Pads
CONNECT RING NET "vdd!" NET "gnd!" IOPAD ALLPORT WIDTH 1000
IORING;

##-- Route the Clock nets
##CLOCKROUTE ALL ;

##-- Route all the nets
SET VAR WROUTE.GROUTE.ONLY FALSE ;
SET VAR WROUTE.FINAL TRUE ;
SET VAR WROUTE.GLOBAL TRUE ;
SET VAR WROUTE.SEARCHREPAIR TRUE ;
SET VAR WROUTE.INCREMENTAL.FINAL FALSE ;
WROUTE NOCONFIG ;

##-- Add Feedthru Cells
##--
#EXEC fillcore.mac ;
SROUTE ADDCELL MODEL FEED25 PREFIX fillcore NO FS
    SPIN 'vdd!' NET 'vdd!' SPIN 'gnd!' NET 'gnd!'
    AREA (  -43500 -43480 ) ( 43640 43610 ) ;
SROUTE ADDCELL MODEL FEED10 PREFIX fillcore NO FS
    SPIN 'vdd!' NET 'vdd!' SPIN 'gnd!' NET 'gnd!'
    AREA (  -43500 -43480 ) ( 43640 43610 ) ;
SROUTE ADDCELL MODEL FEED5 PREFIX fillcore NO FS
    SPIN 'vdd!' NET 'vdd!' SPIN 'gnd!' NET 'gnd!'
    AREA (  -43500 -43480 ) ( 43640 43610 ) ;
```

```
SROUTE ADDCELL MODEL FEED2 PREFIX fillcore NO FS
    SPIN 'vdd!' NET 'vdd!' SPIN 'gnd!' NET 'gnd!'
    AREA (  -43500 -43480 ) ( 43640 43610 ) ;
SROUTE ADDCELL MODEL FEED PREFIX fillcore NO FS
    SPIN 'vdd!' NET 'vdd!' SPIN 'gnd!' NET 'gnd!'
    AREA (  -43500 -43480 ) ( 43640 43610 ) ;

SET VAR DRAW.ROW.AT "OFF";
SET VAR DRAW.CELL.AT "OFF";
SET VAR DRAW.CELL.UNPLACED.AT "OFF";
SET VAR DRAW.PIN.AT "On";
SET VAR DRAW.SWIRE.AT "On";
SET VAR DRAW.SWIRE.GEOM "On";
SET VAR DRAW.WIRE.AT "On";
SET VAR DRAW.WIRE.GEOM "On";
SET VAR DRAW.BLOCKAGE.AT "On";
REFRESH

##-- Save the design
SAVE "final" ;

##-- Save the design as DEF
OUTPUT DEF FILENAME "./DEF/ApbSPI.def" ;
OUTPUT GDSII MAPFILE gds2.map STRUCTURENAME ApbSPI FILE
ApbSPI_se.gds2  ;
SET VAR OUTPUT.VERILOG.PWR.AND.GND.PORTS "TRUE";
OUTPUT VERILOG FILE "./VERILOG/ApbSPI.v" ;

##-- Write RSPF
REPORT RC FILE ApbSPI.rspf ;

##-- Write Logical SDF
SET VAR TIMING.RSPF.FILE "ApbSPI.rspf";
REPORT DELAY SDFOUTPUT FILENAME ApbSPI.sdf USERSPF;
```

15. APPENDIX E: PIN DESCRIPTIONS AND SIMPLIFIED SCHEMATICS OF XS40 BOARD

XS40 Pin	Connects to	Description
25	S0 BLUE0	These pins drive the individual segments of the LED display (S0-S6). They also drive the color and horizontal sync signals for a VGA monitor.
26	S1 BLUE1	
24	S2 GREEN0	
20	S3 GREEN1	
23	S4 RED0	
18	S5 RED1	
19	S6 HSYNCB	
13	CLK	An input driven by the 100 MHz programmable oscillator
44	PC D0	These pins are driven by the data output pins of the PC parallel port. Clocking signals can only be reliably applied through pins 44 and 45 since these have additional hysteresis circuitry. Pins 32 and 34 are mode signals for the FPGA so you must adjust your design to account for the way that the Foundation tools handle these pins. Pins 32 and 34 are not usable as general-purpose I/O on the Spartan FPGA on the XSP Board.
45	PC D1	
46	PC D2	
47	PC D3	
48	PC D4	
49	PC D5	
32	PC D6	
34	PC D7	
37	XTAL1	Pin that drives the uC clock input
36	RST	Pin that drives the uC reset input
29	ALEB	Pin that monitors the uC address latch enable
14	PSENB	Pin that monitors the uC program store enable
7	P1 0	These pins connect to the pins of Port 1 of the uC. Some of the pins are also connected to the status input pins of the PC parallel port. Pin 67 drives the vertical sync signal for a VGA monitor.
8	P1 1	
9	P1 2	
6	P1 3	
77	P1 4 PC S4	
70	P1 5 PC S3	
66	P1 6 PC S5	
67	P1 7 VSYNCB	
69	P3 1(TXD) PC S6	These pins connect to some of the pins of Port 3 of the uC. The uC has specialised functions for each of the port pins indicated in parentheses. Pin 62 connects to the data write pin of the uC and the write-enable pin of the SRAM. Pin 69 connects to a status input pin of the PC parallel port and the PS/2 data line. Pin 68 connects to the PS/2 clock line
68	P3 4(T0) PS/2 CLK	
62	P3 6(WRB) WEB	
27	P3 7(RDB)	
41	P0 0(AD0) D0	These pins connect to Port 0 of the uC, which is also a multiplexed address/data port. These pins also connect to the data pins of the SRAM.
40	P0 1(AD1) D1	
39	P0 2(AD2) D2	
38	P0 3(AD3) D3	

35	P0 4(AD4) D4	
81	P0 5(AD5) D5	
80	P0 6(AD6) D6	
10	P0 7(AD7) D7	
59	P2 0(A8) A8	These pins connect to Port 2 of the uC, which also outputs the upper address byte. These pins also connect to the upper address bits of the SRAM. Pins 28 and 16 are connected to the 128 KB SRAM address pins only on the XS40+ Board. Pins 28 and 16 do not connect to the 32 KB SRAM on the XS40 Board.
57	P2 0(A9) A9	
51	P2 0(A10) A10	
56	P2 0(A11) A11	
50	P2 0(A12) A12	
58	P2 0(A13) A13	
60	P2 0(A14) A14	
28	P2 0(A15) A15	
16	A16	
3	A0	These pins drive the 8 lower address bits of the SRAM.
4	A1	
5	A2	
78	A3	
79	A4	
82	A5	
83	A6	
84	A7	
61	OEB	Pin that drives the SRAM output enable
65	CEB	Pin that drives the SRAM chip enable
75	PC S7	Pin that drives a status input pin of the PC parallel port

Table 15-1 Pin descriptions of XS40 board

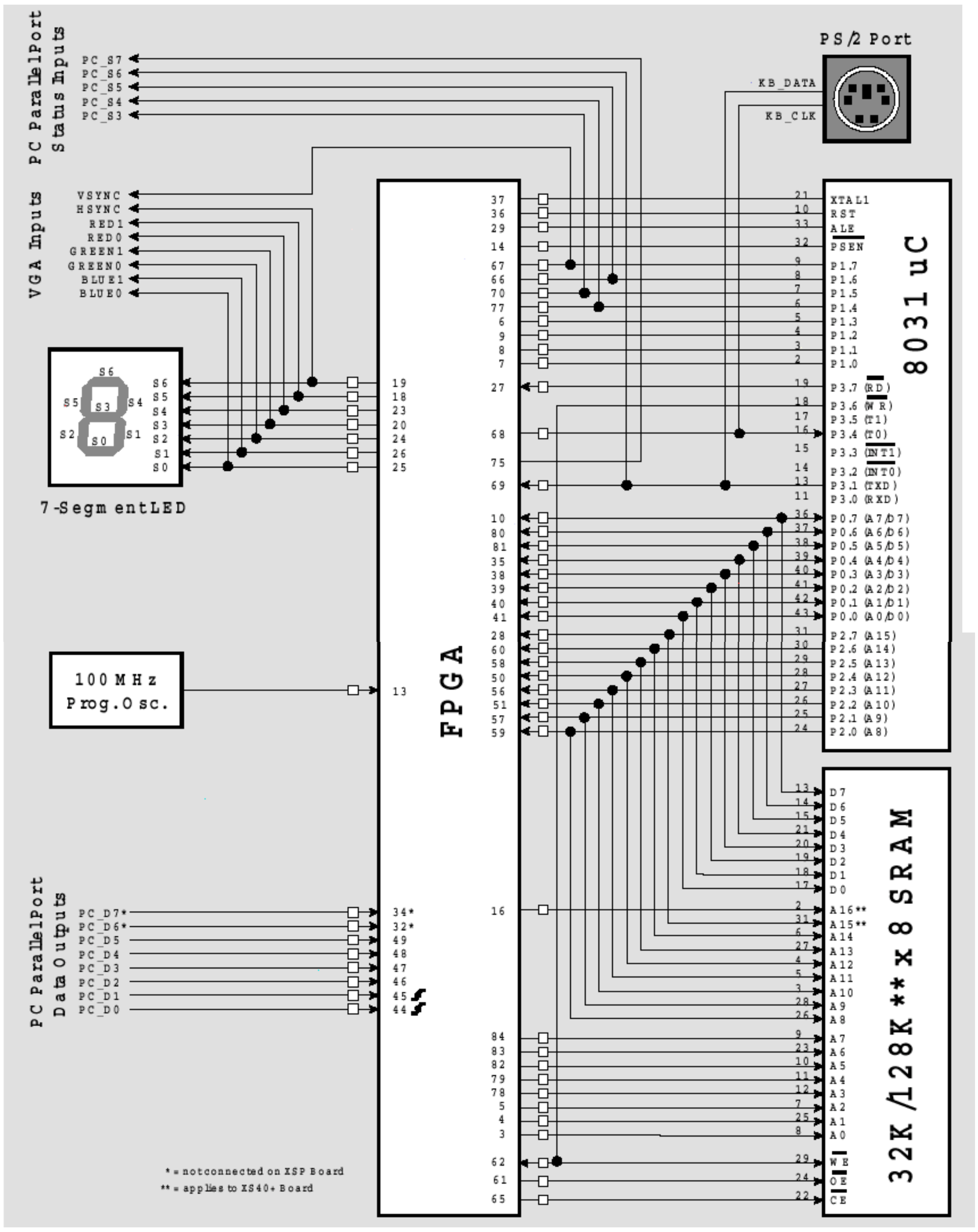


Figure 15-1 Simplified schematic of XS40 board

16. APPENDIX F: SIMULATIONS FOR FPGA IMPLEMENTATION

The following figures show the waveforms for functional simulations in FPGA implementation.

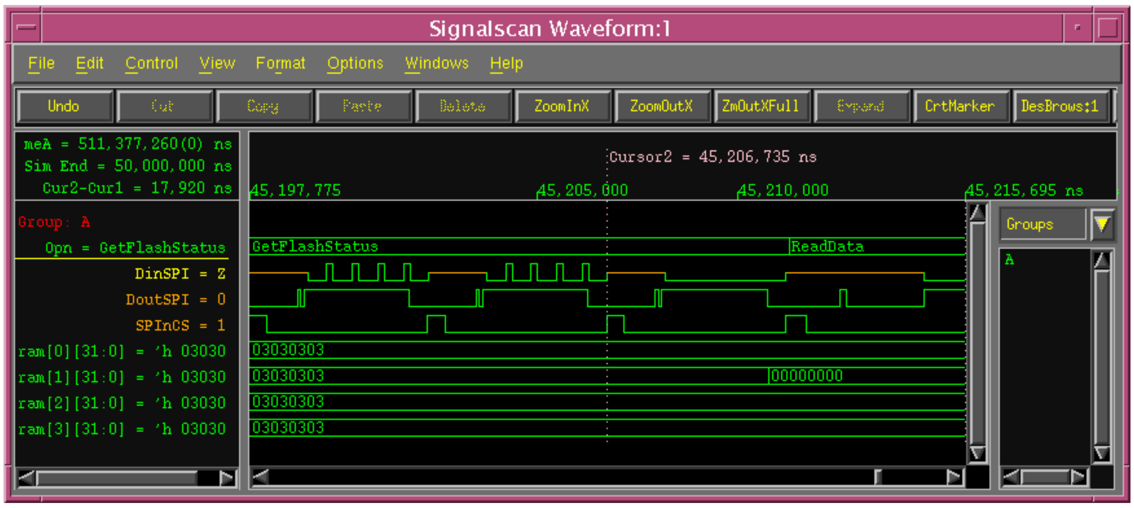


Figure 16-1 GetFlashStatus operation

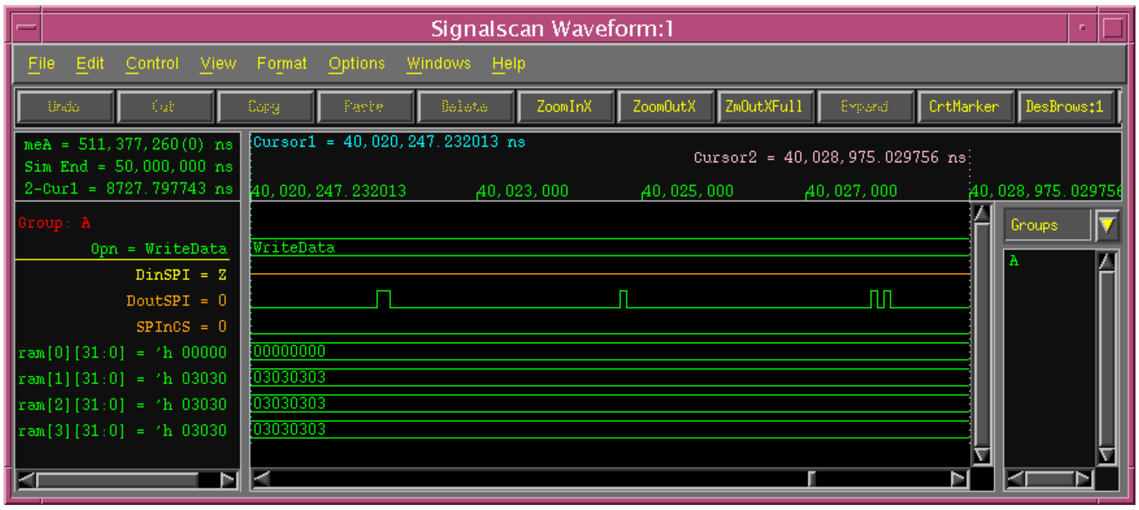
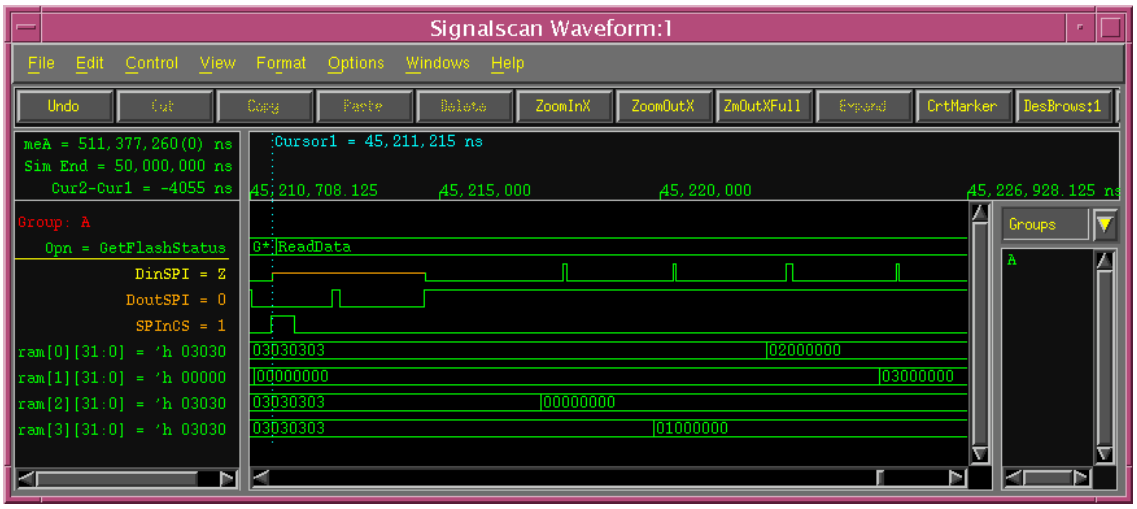
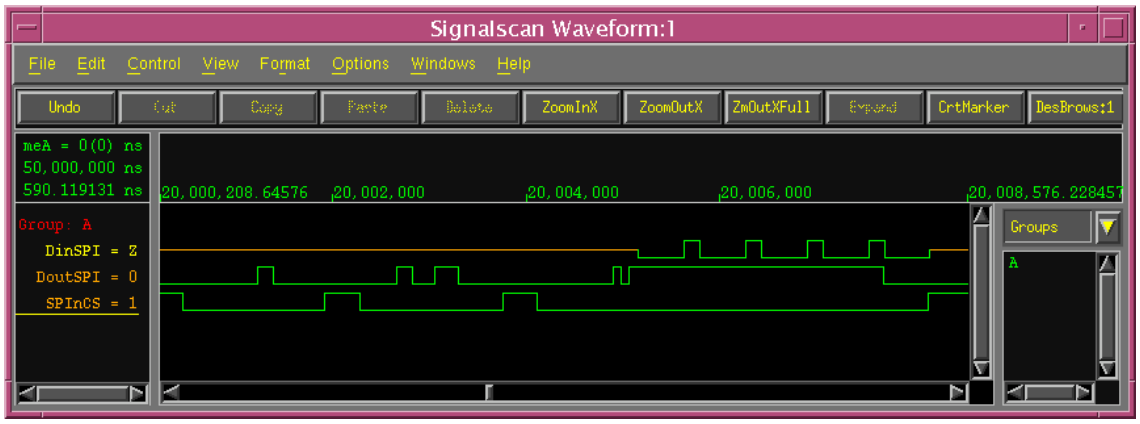


Figure 16-2 Write operation


Signalscan Waveform:1
File Edit Control View Format Options Windows Help
Undo Cut Copy Paste Delete ZoomInX ZoomOutX ZmOutXFull Expand CrtMarker DesBrows:1
Groups
A

```
meA = 511,377,260(0) ns
Sim End = 50,000,000 ns
Cur2-Cur1 = -4055 ns
Cursor1 = 45,211,215 ns
45,210,708.125   45,215,000   45,220,000   45,226,928.125 ns
Group: A
Opn = GetFlashStatus          G* ReadData
DinSPI = Z
DoutSPI = 0
SPInCS = 1
ram[0][31:0] = 'h 03030       03030303   02000000
ram[1][31:0] = 'h 00000       00000000   03000000
ram[2][31:0] = 'h 03030       03030303   00000000
ram[3][31:0] = 'h 03030       03030303   01000000
```

Figure 16-3 Read operation


Signalscan Waveform:1
File Edit Control View Format Options Windows Help
Undo Cut Copy Paste Delete ZoomInX ZoomOutX ZmOutXFull Expand CrtMarker DesBrows:1
Groups
A

```
meA = 0(0) ns
50,000,000 ns
590.119131 ns
20,000,208.64576   20,002,000   20,004,000   20,006,000   20,008,576.228457
Group: A
DinSPI = Z
DoutSPI = 0
SPInCS = 1
```

Figure 16-4 WriteEnable, BulkErase and GetFlashStatus operations

Following figure is the output of the logic analyzer for the test with XS40 board and M25P80 serial flash memory. It shows the WriteEnable, BulkErase and GetFlashStatus operations of serial flash memory controller module.

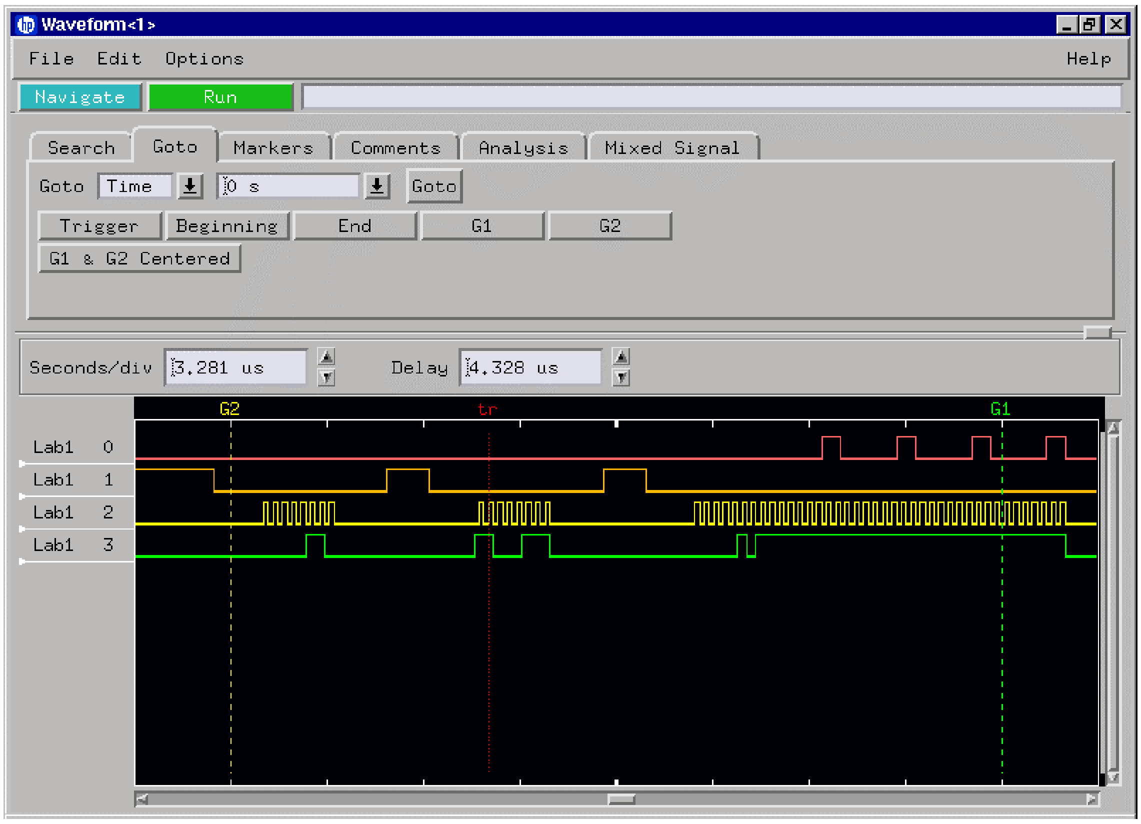


Figure 16-5 Output of logic analyzer

REFERENCES

1. ARM, *PrimeCell Synchronous Serial Port (PL022) Technical Reference Manual*, July 2001.
2. ARM, *AMBA Specification (Rev 2.0)*, May 1999.
3. Michael John Sebastian Smith, *Application Specific Integrated Circuits*, 1997.
4. Pran Kurup, Taher Abbasi, *Logic Synthesis Using Synopsys (Second Edition)*, 1997.
5. Atmel, *64-megabit 2.7-volt Only Dual-interface Data Flash AT45DB642*, Rev. 1638D–11/01.
6. Atmel, *32-Bit Embedded Core Peripheral, Serial Peripheral Interface (SPI)*, Rev. 1244D-CASIC–01/03.
7. STMicroelectronics, *8 Mbit, Low Voltage, Serial Flash Memory with 25 MHz SPI Bus Interface M25P80*, December 2002.
8. NexFlash Technologies, *63M-bit Serial Flash Memory with 4 Pin SPI Interface NX25F641C*, April 2002.
9. Fred G. Martin, *D.4 Serial Peripheral Interface*, November 1995.
10. Motorola, *MC68HC11 Reference Manual*, Prentice Hall 1989.
11. Ohio State University, Information and Electronics Group, *The Serial Peripheral Interface, Nautilus Chip-Project DEEPSEA (Digital Exportation of an Established Protocol from Sensing Encoded Analog)*, January 2001.
12. Andrew Chu, Chris Ohlmann, VHDL Implementation of Serial Peripheral Interface, 2002.
13. Cadence, *Serial Peripheral Interface Technical (SPI) Technical Data Sheet*, December 2002.
14. Palmchip, *Serial Peripheral Interface Controller*, January 2002.

15. ST Microelectronics, *SPI Communication Between ST7 and EEPROM*, 1999.

16. Xilinx, *CoolRunner Serial Peripheral Interface Master*, December 2002.

17. Alma Technologies, *SPI Master/Slave Core*, 2002.

18. Dave Bursky, *Serial Flash Memories Rise To Meet Changing System Needs*, Marc 1999.

19. Brett Glass, *There in a Flash: Flash Memory for Embedded Systems*, CMP Media Inc. 2000.

20. ST Microelectronics, *Flash Memories*, B979M - June 2003.

21. ST Microelectronics, *Using Serial Flash Memories for Code Storage in Computer and Peripherals Applications*, May 2002.

22. XESS Corporation, *XS40, XSP Board V1.4 XS40, XSP Board V1.4 XS40, XSP Board V1.4 XS40, XSP Board V1.4 User Manual,* 9/21/2001.