LatticeMico32 Parallel Flash Controller

The LatticeMico32 parallel flash memory controller is a slave device for the WISHBONE architecture. It is used to interface with a parallel flash device that is compliant with the common flash memory interface (CFI).

Version

This document describes the 3.0 version (formerly the 7.0 SP2 version) of the LatticeMico32 parallel flash controller.

Features

The LatticeMico32 parallel flash memory controller includes the following features:

- WISHBONE B.3 interface
- Configurable data bus width up to 32 bits
- Configurable address bus width up to 32 bits
- Configurable read latency
- Configurable write latency
- Configurable extra flash signals: byte, write protect, reset

For additional details about the WISHBONE bus, refer to the *LatticeMico32 Processor Reference Manual*.

Functional Description

The asynchronous flash memory controller translates the synchronous WISHBONE bus signals into control strobes used to access an asynchronous flash PROM. The controller decodes the WISHBONE cycle type and generates asynchronous chip selects, byte enables, read enables, and write enables, as required. The controller interacts with the WISHBONE master port, using classic-mode registered-feedback bus cycle control strobes. For further information on the WISHBONE registered feedback bus cycle, refer to *WISHBONE Specifications, Version B3,* Chapter 4.

The memory controller has a configurable address bus and data bus width. The address bus can be up to 32 bits long. The data bus can be configured for 8-, 16-, or 32-bit widths. You can instantiate multiple flash controllers to permit access to memories of varying address and data bus sizes.

The controller also provides the ability to statically assert reset, write protect, and byte/word inputs for flash PROM devices with these controls.

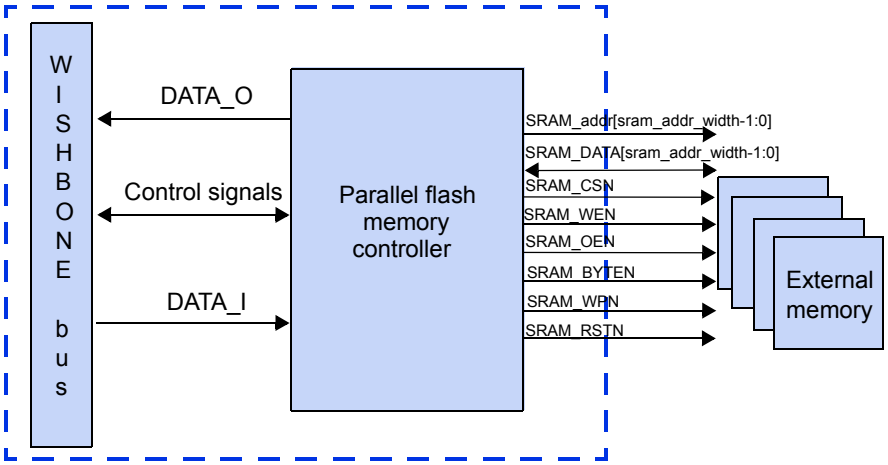
When in operation, the controller monitors the address bus, STB_I, and CYC_I to determine when a memory transaction is in progress. The address, STB_I, and CYC_I control signals are asserted or deasserted at the CLK_I rising edge. CLK_I may be transitioning at a rate much too high for the flash device to accept, so the memory controller must control and hold off the ACK_O control signal that indicates that the WISHBONE bus transaction is complete.

Since flash devices do not have a cycle acknowledge signal, the memory controller provides one. The ACK_O signal is controlled with a fixed read/write latency value. The read latency is independent of the write latency. Each increment in latency value represents an increase in the length of the bus transaction by one CLK_I time period. The read/write latency permits the controller to work with slower flash devices. The controller counts CLK_I cycles until the read/write latency value has been reached. When the latency period expires, ACK_O is asserted, and the WISHBONE cycle is terminated.

The memory controller does not implement dynamic or region-based latency. The read and write latency values are fixed during module generation. If different regions of the asynchronous memory space require different read/write latency values, you must generate one flash controller for each region. If the latency values cannot be set high enough to permit the flash to operate, it is necessary to obtain faster flash, reduce the CLK_I time period, or modify the HDL source for the controller.

Figure 1 shows how an application uses the flash memory controller.

Figure 1: Parallel Flash Usage



Configuration

The following sections describe the graphical user interface (UI) parameters, the hardware description language (HDL) parameters, and the I/O ports that you can use to configure and operate the LatticeMico32 parallel flash memory controller. The parallel flash shares the data and address buses with the on-board SRAM.

UI Parameters

Table 1 shows the UI parameters available for configuring the LatticeMico32 parallel flash controller through the Mico System Builder (MSB) interface.

Table 1: Parallel Flash Controller UI Parameters

Dialog Box Option	Description	Allowable Values	Default Values
Instance Name	Specifies the name of the parallel flash controller instance.	Alphanumeric and underscores	flash
Base Address	Specifies the base address for the device. The minimum boundary alignment is 0x4.	0X00000000–0XFFFFFFFF	0X00000000
Size	Specifies the size of the external memory, in bytes.	0 – 4294967296	1048576
Share External Ports (for HPE_mini Board)	Enables a common address and data bus for flash and SRAM components created for the platform. When this option is selected, each flash and SRAM component adds its instance name to the address or data bus port name.	1, 0	1

Table 1: Parallel Flash Controller UI Parameters (Continued)

Dialog Box Option	Description	Allowable Values	Default Values
Enable Extra Flash Signals	Enables the extra flash signals Byte, Write Protect, and Reset. Selecting this option allows the memory controller to drive these controls to a static logic level. These signals should all be set to Hold High for the LatticeMico32 System flash. If you are using a flash other than the one provided with LatticeMico32 System, consult the flash data sheet.	1, 0	1
Byte signal	Statically asserts an output that can be connected to a flash PROM's byte/word data bus control. Setting this control to 1 drives the output to Voh. This signal should be set to 1 (FLASH_BYTE = 1) for the LatticeMico32 System flash.	1, 0	1
Write Protect signal	Statically asserts an output that can be connected to a flash PROM's write protect control. Setting this control to 1 drives the output to Voh. This signal should be set to 1 (FLASH_WP = 1) for the LatticeMico32 System flash.	1, 0	1
Reset signal	Statically asserts an output that can be connected to a flash PROM's reset control. Setting this control to 1 drives the output to Voh. This signal should be set to 1 (FLASH_RST = 1) for the LatticeMico32 System flash.	1, 0	1
Settings			
Read Latency	Specifies the latency for reading the asynchronous memory, in clock cycles. The OE strobe is increased by one CLK_I cycle for each increment in the latency value.	1 – 15	7
Write Latency	Specifies the latency for writing the asynchronous memory, in clock cycles. The WE strobe is increased one CLK_I period for each increment in the latency value.	1 – 15	7
FLASH Address Width	Specifies the width of the flash address, in bits.	1 – 32	25

Table 1: Parallel Flash Controller UI Parameters (Continued)

Dialog Box Option	Description	Allowable Values	Default Values
FLASH Data Width	Specifies the width of the data bus of flash memory, in bits.	1 – 32	32
FLASH Byte Enable Width	Specifies the width of the byte enable, or control strobe, for each of the 8-bit pieces of logic that constitute the data bus in the asynchronous RAM. The byte enable indicates that LatticeMico32 is accessing the 8-bit sub-element of the larger combined data bus. The SRAM Byte Enable width must be assigned a value that is the data bus width modulo 8. For example, if the default value of the data bus width is 32, the SRAM BE width should be 4. Legal values for this field are 4, 2, and 1. The byte enables (BE[n], n= 3..0) are activated as follows: 32-bit bus: D31-24/BE3 D23-16/BE2 D15-8/BE1 D7-0/BE0 16-bit bus: D15-8/BE1 D7-0/BE0 8-bit bus: D7-0/BE0 Therefore, you use SRAM_BE_WIDTH to define the upper limit of the Verilog bus: output [SRAM_BE_WIDTH-1:0] sram_be;	4, 2, 1	4

HDL Parameters

Table 2 lists the parameters that appear in the HDL.

Table 2: Parallel Flash Controller HDL Parameters

Parameter Name	Description	Allowable Values
SRAM_DATA_WIDTH	Defines the width of data bus of the memory	8, 16, 32
SRAM_ADDR_WIDTH	Defines the width of the address	1 – 32
READ_LATENCY	Defines the latency for reading the flash	1 – 15
WRITE_LATENCY	Defines the latency for writing the flash	1 – 15
FLASH_BYTEN	Selects 8-bit or 16-bit mode	1, 0
FLASH_WPN	Hardware write protect pin, active low	1, 0
FLASH_RSTN	Hardware reset pin, active low	1, 0

I/O Ports

Table 3 describes the input and output ports of the LatticeMico32 parallel flash controller. The parallel flash controller shares the data and address buses with the on-board SRAM. The WEN and OEN common control signals are also shared, although each of these types of memory has its own chip select.

Table 3: Parallel Flash Controller I/O Ports

I/O Port	Active	Direction	Initial State	Description	Correlated Dialog Box Option
WISHBONE Side signals					
CLK_I	—	I	X	System clock signal	
RST_I	High	I	X	System reset signal	
CTI_I	—	I	X	Cycle-type identifier signal.	
BTE_I	—	I	X	Burst-type extension signal (only linear incremental burst is supported)	
ADR_I [31:0]	—	I	X	WISHBONE address bus signal	
DAT_I [31:0]	—	I	X	WISHBONE data bus input signal	
SEL_I [3:0]	High	I	X	Select output array signal, one bit for every byte	
WE_I	High	I	X	Write enable signal	
STB_I	High	I	X	Strobe signal indicating a valid data transfer	
CYC_I	High	I	X	Signal indicating a valid bus cycle in progress	
ACK_O	High	O	0	Signal indicating the normal termination of a bus	
DAT_O	—	O	0	WISHBONE data bus output	
Asynchronous Memory Interface Signals					
SRAM_addr [sram_ addr_width-1:0]	—	O	0	Flash address output signal	Flash Data Width
SRAM_DATA [sram_data_width-1:0]	—	Bi	0	Flash data input/output signal	Flash Address Width
SRAM_CSN	Low	O	0	Flash chip select signal	
SRAM_WEN	Low	O	0	Flash write enable signal	
SRAM_OEN	Low	O	0	Flash output enable signal	
Flash Interface Signals					
SRAM_BYTEN		O	0	Flash byte/word select	Byte signal

Table 3: Parallel Flash Controller I/O Ports (Continued)

I/O Port	Active	Direction	Initial State	Description	Correlated Dialog Box Option
SRAM_WPN		O	0	Flash write protect enable signal	Write protect signal
SRAM_RSTN		O	0	Flash reset enable signal	Reset signal

Timing Diagrams

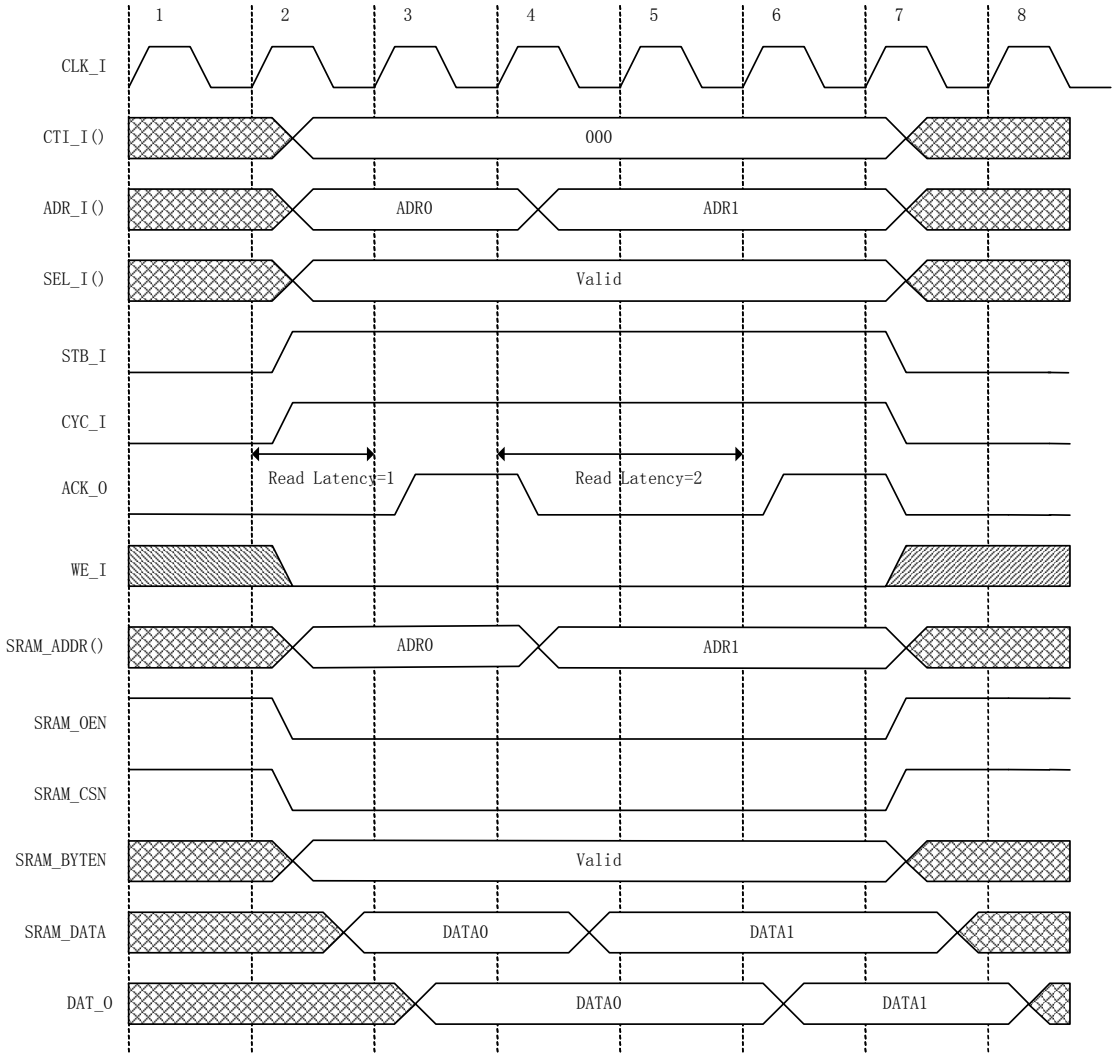
The timing diagrams featured in Figure 2 and Figure 3 show the timing of the parallel flash controller's WISHBONE and external signals.

Parallel Flash Read Logic

A read memory transaction begins with CYC_I and STB_I being asserted following a CLK_I rising edge, as shown in CLK_I Cycle 2 in Figure 2. The memory controller passes the address asserted on ADR_I to the SRAM_ADDR bus when CYC_I and STB_I are asserted. The memory controller counts CLK_I cycles until the read latency counter reaches its terminal count, causing the ACK_O strobe to be asserted. This is shown in clock cycle 3 and 6 in Figure 2.

The memory controller latches the data driven by the FLASH and drives it onto the DAT_O bus, starting on the CLK_I rising edge on which ACK_O is asserted. DAT_O is therefore valid in CLK_I Cycle 4, as shown in Figure 2.

Figure 2: Parallel Flash Read



Parallel Write Logic

A write memory transaction begins with CYC_I and STB_I being asserted following a CLK_I rising edge, as shown in CLK_I Cycle 2 in Figure 3. The memory controller passes the address asserted on ADR_I to the RAM_ADDR bus when CYC_I and STB_I are asserted. The SRAM_DATA bus follows the DAT_I bus.

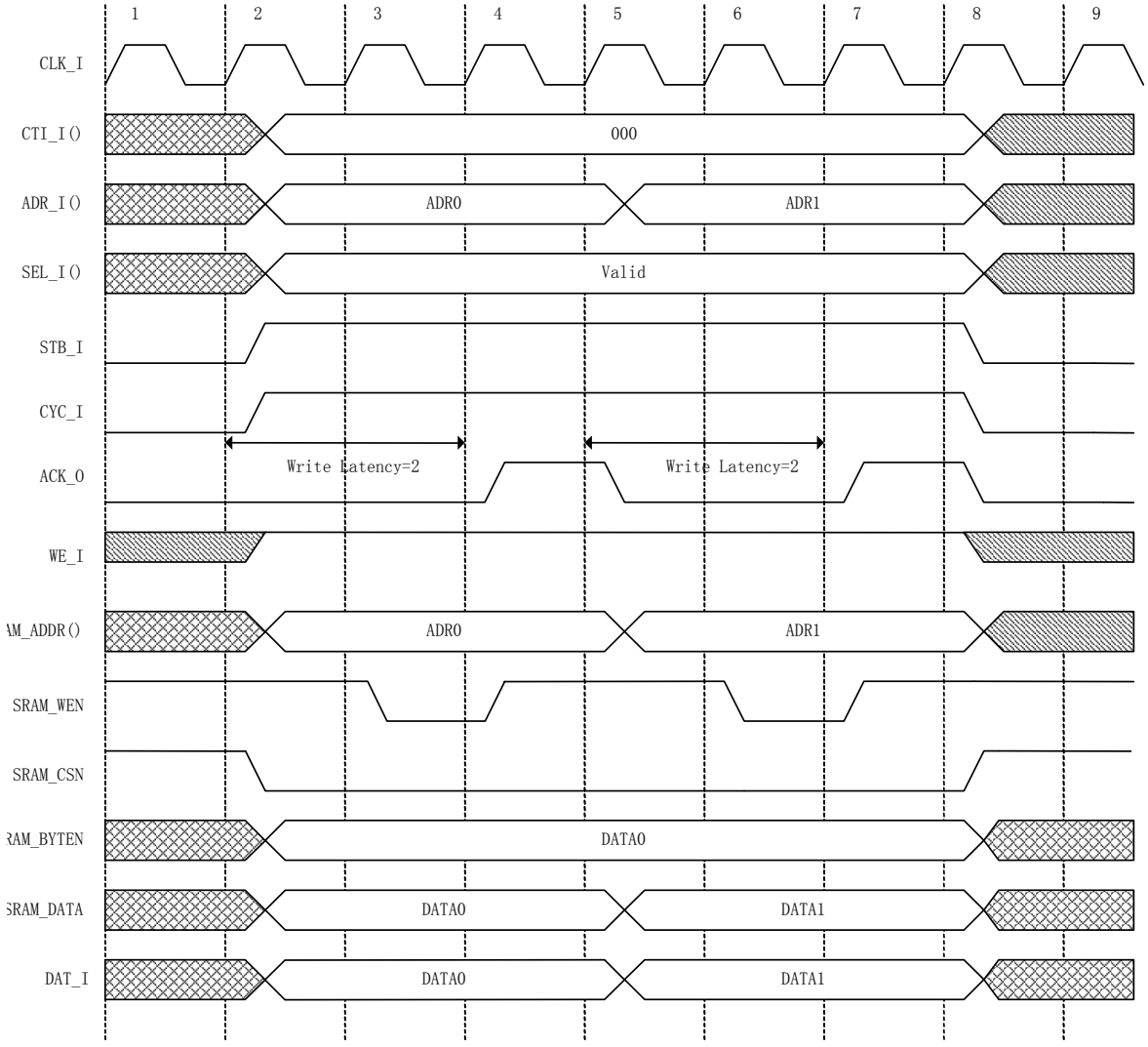
Once the WISHBONE cycle has started (that is, when CYC_I and STB_I are asserted) and the WE_I signal indicates a write cycle is in progress, the controller can assert the RAM_WEN strobe. The soonest that RAM_WEN can be asserted is in cycle 3.

The memory controller counts CLK_I cycles until the write latency counter reaches its terminal count, when the ACK_O strobe is asserted. The SRAM_WEN strobe is deasserted at the same time that ACK_O is asserted, as shown in clock cycle 4 and 7 in Figure 3.

The SRAM memory latches the data driven by the LatticeMico32 controller at the rising edge of the SRAM_WEN strobe. The rising edge of SRAM_WEN signals the completion of a single SRAM memory write transaction.

The asynchronous SRAM controller never uses the chip select to terminate a memory transaction.

Figure 3: Parallel Flash Write



EBR Resource Utilization

The LatticeMico32 parallel flash controller uses no EBRs.

Software Support

Software support for the LatticeMico32 parallel flash controller is relevant only if the component interfaces to a CFI-compliant parallel flash device.

For a read operation, this component behaves as a memory device, and no explicit driver support required. However, modifying the contents of the flash component requires a software implementation for erasing and writing locations within it. A set of SW services are provided with the component to assist with erasing and writing flash devices.

For detailed information on the CFI flash service, refer to the *LatticeMico32 Software Developer User Guide*.

Device Driver

This section describes the type definitions for the parallel flash controller device context structure.

This structure, shown in shown in Figure 4, contains parallel flash controller component-specific information and is dynamically generated in the DDStructs.h header file. This information is largely filled in by the MSB managed build process, which extracts the parallel flash controller component-specific information from the platform definition file. The members should not be manipulated directly, because this structure is for exclusive use by the device driver.

Figure 4: Parallel Flash Controller Device Context Structure

```
typedef struct st_CFIFlashDevCtx_t {
const char* name;
unsigned int base;
unsigned int byteSize;
DeviceReg_t lookupReg;
unsigned int end;
void * cfgFnTbl;
CFIInfo_t CFIInfo;
void * prev;
void * next;
} CFIFlashDevCtx_t;
```

Table 4 describes the parameters of the parallel flash memory device context structure shown in Figure 4.

Table 4: Parallel Flash Memory Device Context Structure Parameters

Parameter	Data Type	Description
name	const char *	Specifies the name of the instance. It is given by the managed build process.
base	unsigned int	Specifies the base address of the LatticeMico32 flash component. It is given by the managed build process.
bytesize	unsigned int	Specifies the size of the LatticeMico32 flash instance, in bytes. It is given by the managed build process.
lookupReg	DeviceReg_t	Used by the device driver to register the parallel flash controller component instance with the LatticeMico32 lookup service. Refer to the *LatticeMico32 Software Developer User Guide* for a description of the DeviceReg_t data type.
end	unsigned int	Specifies the end address of the LatticeMico32 flash component instance. It is given by the LatticeMico32 flash component initialization routine.
cfgFnTbl	void *	Is a pointer to the configuration's function-table structure, as described in the *LatticeMico32 Software Developer User Guide*. It is given by the CFI flash service on successfully identifying the underlying flash configuration. It is declared in the LatticeMico32CFI.h header file.
CFIInfo	CFIInfo_t	Is a CFI parameter table populated by the CFI flash service. It is used by configuration-specific functional implementation. It is declared in the CFIInfo_t.h and CFIRoutines.h header files.

Functions

This section describes the implemented device-driver-specific functions.

LatticeMico32InitCFIFlashDriver Function

```
void LatticeMico32InitCFIFlashDriver (struct
st_CFIFlashDevCtx_t*);
```

This function initializes a LatticeMico32 parallel flash memory component. It is called as a part of platform initialization and is responsible for identifying and populating appropriate flash configuration function-handler routines, as well as for reading the CFI parameters from the flash device.

Table 5 describes the parameter in the LatticeMico32InitCFIFlashDriver function syntax.

Table 5: LatticeMico32InitCFIFlashDriver Function Parameter

Parameter	Description
struct st_CFIFlashDevCtx_t *	Pointer to a valid flash device context

Note: For a managed build, the instance-specific structure is located in DDStructs.c.

For detailed information on the CFI flash service and associated functions, refer to the *LatticeMico32 Software Developer User Guide*.

Service

The CFI flash device driver registers flash component instances with the LatticeMico32 lookup service, using their instance names for device names and “CFIFlash Device” as the device type.

For detailed information on services provided for LatticeMico32 flash components, refer to the *LatticeMico32 Software Developer User Guide*.

Software Usage Example

For a software usage example, see the application template located in the following folder:

<*install_dir*>\micosystem\utilities\templates\CFIFlashProgrammer