

OnBoard MemCfg Reference Project

Revision: March 30, 2009



This document was produced by
Digilent Romania. For questions,
contact support@digilent.ro.

Overview

The OnBoard MemCfg reference project contains the design for a Digilent on-board memory controller. This configuration, in conjunction with a PC and communication module (e.g., Digilent on-board USB circuitry), allows the user to read/write/erase the RAM and Flash 16-bit memory chips loaded on Digilent system boards (Nexys, Nexys 2, etc.) The PC application program can be a Digilent utility (MemUtil, TransPort, etc.) or a custom application. Digilent Adept Suite software provides both utility programs and DLLs to be used with custom applications for interfacing with Digilent communication modules.

The first part of this document describes the structure and behavior of the project. The second part shows how to use the project in conjunction with a PC.

Functional Description

The OnBoard MemCfg reference project implements an EPP interface (EppCtrl) that is able to communicate with the EPP port emulated by the Digilent on-board USB circuitry. The EPP interface controls the EPP data registers implemented in the memory module controller (OnBoardMemCtrl). The memory module controller generates the signal sequence needed to read, write, or erase the Flash and to read or write the RAM chips. The CompSel component generates a selection signal to activate the NexysOnBoardMemCtrl whenever the EppAdr register is in range "00000000" to "00000111".

For detailed descriptions of each component, see the *Digilent Component Library* and the VHDL source file header and comments.

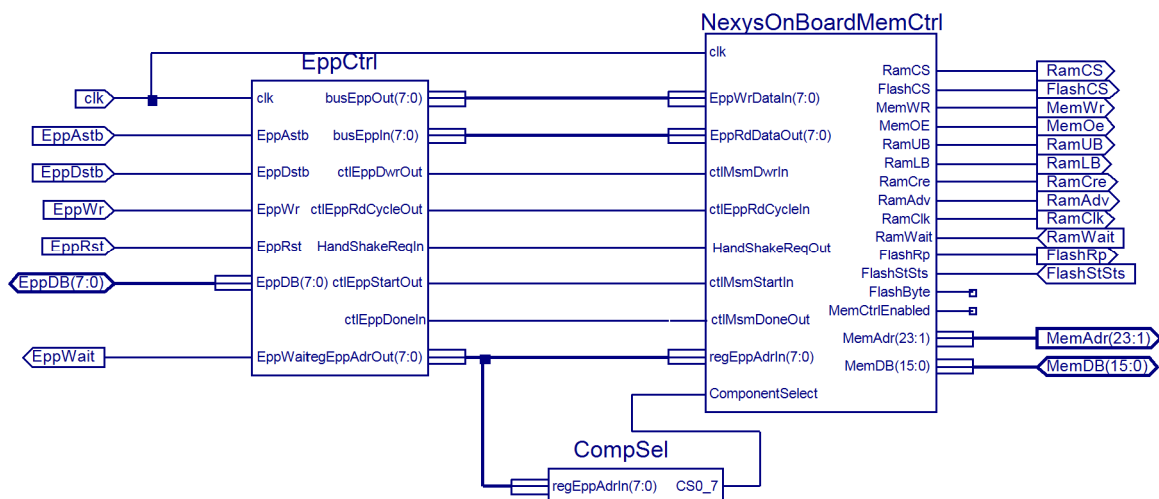


Figure 1 The OnBoard MemCfg Reference Project

Port Definitions

Clock Signal

clk in, system clock (50MHz)

EPP Bus Signals

EppAstb in, Address strobe
EppDstb in, Data strobe
EppWr in, write signal
EppRst in, reset signal
EppDB in/out, 8-bit data bus
EppWait out, wait signal

Memory Bus Signals

MemDB inout _vector(15 downto 0), -- Memory data bus
MemAdr out _vector(23 downto 1), Memory Address bus
FlashByte out, Byte enable('0') or word enable('1')
RamCS out, RAM CS
FlashCS out, Flash CS
MemWR out, memory write
MemOE out, memory read (Output Enable), also controls the MemDB direction
RamUB out, RAM Upper byte enable
RamLB out, RAM Lower byte enable
RamCre out, Cfg Register enable
RamAdv out, RAM Address Valid pin
RamClk out, RAM Clock
RamWait in, RAM Wait pin
FlashRp out, Flash RP pin
FlashStSts in, Flash ST-STs pin
MemCtrlEnabled out, MemCtrl takes bus control

The EppCtrl Interface

This file contains the design for an EPP interface controller. This configuration, in conjunction with a communication module (e.g., Digilent USB, Serial, Network, or Parallel module or Digilent on-board USB circuitry), allows the user to interface some other FPGA implemented "client" components (Digilent Library components or custom components) to a PC application (a Digilent or custom utility).

Functional Description

All the Digilent communication modules above emulate an EPP interface at the FPGA (board connector) pins, compatible to the EppCtrl controller.

The controller performs the following functions:

- it manages the EPP standard handshake
- it implements the standard EPP Address Register
- it provides the signals needed to read/write EPP Data Registers.
- It defines two types of data register access:

Register Transfer reads or writes a client data register with no handshake to the client component.

Process Launch launches a client process and waits for it to complete according to a handshake protocol described below.

The "client" component(s) is (are) responsible for implementing the specific required data registers, as explained below:

- declare the data read and write registers
- assign an EPP address for each.

An EPP address can be assigned to a Read register or a Write register or to a couple consisting of both a Read register and a Write register. A unique address must be assigned to each register (couple) throughout all the client components connected to the same EppCtrl.

The totality of assigned addresses builds the component address range. If fewer than 256 (couples of) registers are required, "mirror" or "alias" addresses can be used (incomplete regEppAdrOut(7:0) decoding). The mirror addresses are not allowed to overlap throughout all the client components connected to the same EppCtrl.

- use the same clock signal for all data registers as well as for EppCtrl component Write Data registers
- connect the inputs of all write registers to busEppOut(7:0)
- decode regEppAdrOut(7:0) to generate the CS signal for each write register
- use ctlEppDwrOut as WE signal for all the write registers
- connect the outputs of all read registers to busEppIn(7:0) THROUGH A MUX
- use the regEppAdrOut(7:0) as MUX address lines.

The client process is required to conform to the handshake protocol described below. The client component decides to which type the current Data Register Access belongs: a clock period (20ns for 50MHz clock frequency) after ctlEppDwrOut becomes active, EppCtrl samples the HandShakeReqIn input signal. If inactive, the current transfer cycle is a Register Transfer. It completes without a handshake protocol. If active (HIGH), the current transfer cycle is a Process Launch. It uses a handshake protocol.

The Handshake Protocol

- the busEppOut, ctlEppRdCycleOut and regEppAdrOut(7:0) signals freeze
- (for a WRITE cycle, ctlEppDwrOut pulses LOW for one CK period, the selected write register is set)
- the ctlEppStartOut signal is set active (HIGH)
- (for a READ cycle, client application places data on busEppIn(7:0))

- the controller waits for the `ctlEppDoneIn` signal to become active (HIGH)
- (for a READ cycle, the data transfer is performed one CK period later)
- the `ctlEppStartOut` signal is set inactive (LOW)
- the controller waits for the `ctlEppDoneIn` signal to become inactive (LOW)
- a new transfer cycle can begin (if required by the PC application)

A client component can use the handshake protocol feature for enlarging the `EppCtrl` cycles, as described below:

1. Activate the `HandShakeReqIn` input signal, either:
 - a. Permanently; every EPP bus cycle will be enlarged.
 - b. Only when the `regEppAdrOut(7:0)` value identifies a Data Register that requires an internal process. `ctlEppRdCycleOut` could be used to further discriminate between read or write EPP bus cycles. Only these EPP bus cycles will be enlarged.
2. Wait for the `ctlEppStartOut` signal to become active.
3. Keep the `ctlEppDoneIn` signal inactive (LOW) for the desired time
 - a. a fixed amount of time
 - b. a variable time, up to the completion of the internal process.
4. Activate the `ctlEppDoneIn` signal.
5. Wait for the `ctlEppStartOut` signal to become inactive.
6. Inactivate `ctlEppDoneIn`.
7. Cycle from 1.

When enlarging EPP bus cycles, be aware that the PC application might use an internal WatchDog timer. If `ctlEppDoneIn` is held inactive for a long time, the whole EPP cycle could become longer than the WatchDog time, resulting in a PC application time-out error.

Port Definitions

Clock Signal

clk in, system clock (50MHz)

EPP Bus Signals

EppAstb in, Address strobe

EppDstb in, Data strobe

EppWr in, write signal

EppRst in, reset signal

EppDB in/out, 8-bit data bus

EppWait out, wait signal

User Signals

busEppOut out, 8-bit Data Output bus

busEppIn in, 8-bit Data Input bus

ctlEppDwrOut out, Data Write pulse

ctlEppRdCycleOut in/out, Indicates a READ EPP cycle

regEppAdrOut in/out, = "00000000", 8-bit EPP Address Register content

HandShakeReqIn in, User Handshake Request

ctlEppStartOut out, Automatic process Start

ctlEppDoneIn in, Automatic process Done EPP interface signals

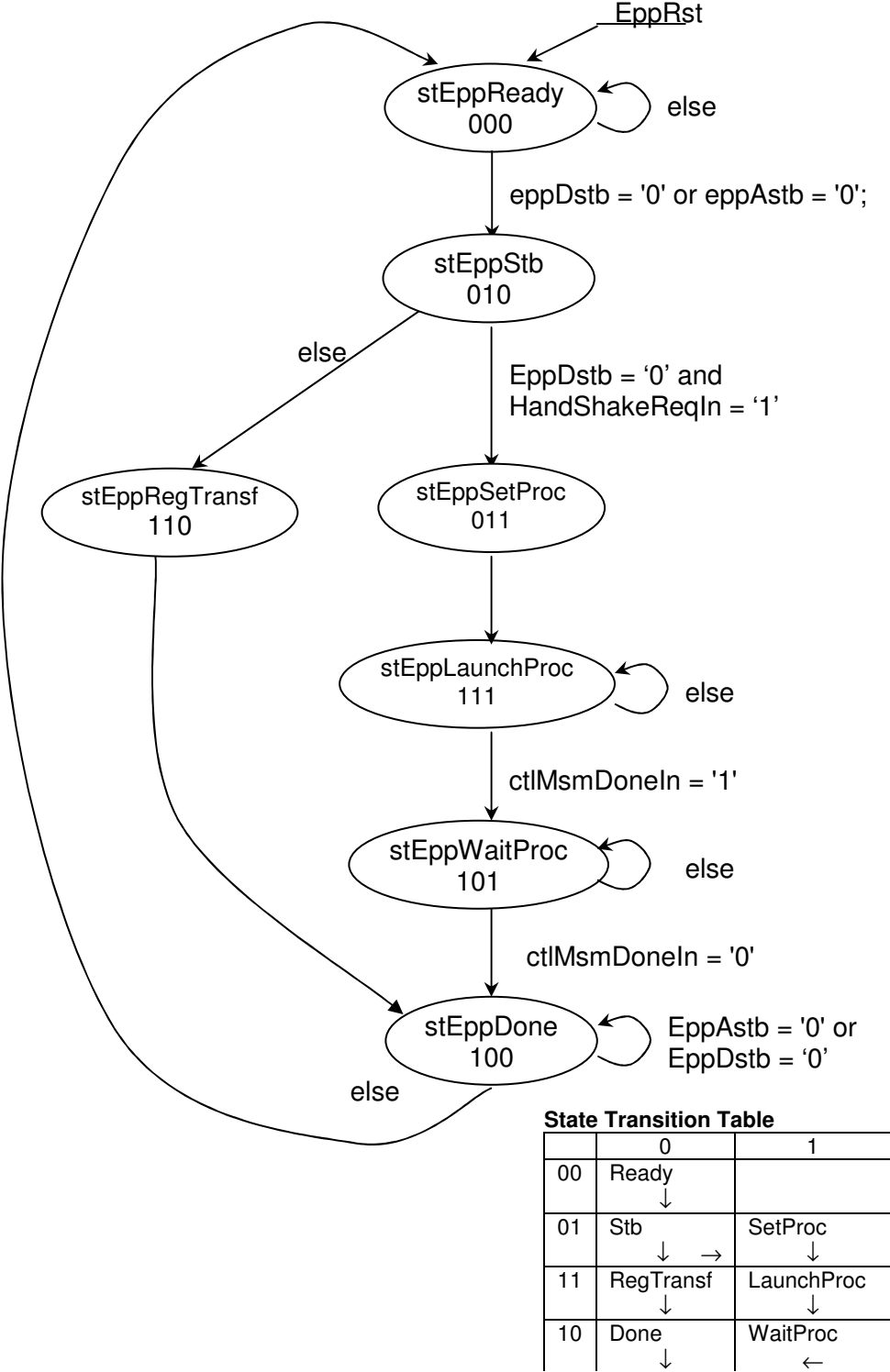


Figure 2 EppCtrl State Diagram and State Transition Table

NexysOnBoardMemCtrl

This component, in conjunction with a communication module (Nexys on-board USB controller), a PC application (a Diligent or custom utility) and the EppCtrl Diligent Library component allows the user to read or write the On Board RAM and Flash memory chips on the Diligent boards (Nexys, Nexys 2).

Functional Description

NexysOnBoardMemCtrl acts as a "client" for EppCtrl. It implements the following EPP data registers:

Register	Function
0	Memory control register (read/write)
1	Memory address bits 0-7 (read/write)
2	Memory address bits 8-15 (read/write)
3	Memory address bits 16-23 (read/write)
4	Memory data write holding register (read/write) (see Note 1)
5	Memory data read register (read) (see Note 2)
6	RAM auto read/write register (read/write) (see Note 3)
7	Flash auto read/write register (read/write) (see Note 4)

Reading from or writing to registers 0-5 generates a simple Register Transfer (see the EppCtrl.vhd header). The HandShakeReqOut is inactive when regEppAdrlIn(2:0) = 0-5.

Registers 6 and 7 are used for block transfers. They belong to the "Process Launch" type, both for read and write operations (see the EppCtrl.vhd header). The HandShakeReqOut activates when regEppAdrlIn(2:0) = 6 or 7. Consequently, at each EPP Data Register read or write cycle for this address, the EppCtrl component activates the ctlEppStart signal.

As a response, the NexysOnBoardMemCtrl:

- performs the actions explained by Note 3 or 4.
- activates the ctlMsmDoneOut signal
- waits for the ctlEppStart signal to go inactive
- inactivates the ctlMsmDoneOut signal
- returns to the idle state (stMsmReady).

Note 1: Writing to "registers" 6 or 7 also updates register 4.

Note 2: "Register 5" is not a physical register. The memory data bus content is read at this address.

Note 3: Writing to "register" 6, when in byte mode:

- updates the register 4 content AND
- launches an automatic RAM write cycle (asynchronous mode):
 - generates the appropriate control signal sequence
- increments the Memory address register by 1.

Reading from address 6, when in byte mode:

- launches an automatic RAM read cycle (asynchronous mode):
 - generates the appropriate control signal sequence
 - loads the AutoReadData register with the content of the addressed RAM location
 - increments the Memory address register by 1.

Writing to "register" 6, when in word mode and even address:

- launches a blind cycle to update the auxiliary register. No RAM cycle is launched.
- increments the Memory address register by 1.

Writing to "register" 6, when in word mode and odd address:

- updates the register 4 content AND
- launches an automatic RAM write cycle (asynchronous mode):
 - generates the appropriate control signal sequence. A word is written to RAM:
 - memory address bus is set to the word address ($A0 = 0$)
 - memory data bus is set with the previous stored value of auxiliary register as lower byte and current value of register 4 as higher byte)
- increments the Memory address register to the next even.

Reading from address 6, when in word mode and even address:

- launches an automatic RAM read cycle (asynchronous mode):
 - generates the appropriate control signal sequence
 - loads the AutoReadData register with the lower byte of the addressed RAM location (sent over the Epp)
 - loads the auxiliary register with the upper byte of the addressed RAM location (not yet sent over the Epp)
- increments the Memory address register by 1.

Reading from address 6, when in word mode and odd address:

- launches a blind cycle to send the previous stored value of the auxiliary register. No RAM cycle is performed.
- increments the Memory address register by 1.

Note 4: Writing to "register" 7, when in byte mode:

- updates the register 4 content AND
- launches an automatic Flash write cycle:
 - generates the appropriate control signal sequence
 - waits for the internal Flash write procedure to complete
- increments the Memory address register by 1.

Reading from address 7, when in byte mode:

- launches an automatic Flash read cycle:
 - generates the appropriate control signal sequence
 - loads the AutoReadData register with the content of the addressed Flash location
- increments the Memory address register by 1.

Writing to "register" 7, when in word mode and even address:

- launches a blind cycle to update the auxiliary register. No Flash cycle is launched.
- increments the Memory address register by 1.

Writing to "register" 7, when in word mode and odd address:

- updates the register 4 content and
- launches an automatic Flash write cycle:
 - generates the appropriate control signal sequence (a word is written to Flash:
 - memory address bus is set to the word address (A0 = 0)
 - memory data bus is set with the previous stored value of auxiliary register as lower byte and current value of register 4 as higher byte)
 - waits for the internal Flash write procedure to complete
- increments the Memory address register to the next even.

Reading from address 7, when in word mode and even address:

- launches an automatic Flash read cycle:
 - generates the appropriate control signal sequence
 - loads the AutoReadData register with the lower byte of the addressed Flash location (sent over the Epp)
 - loads the auxiliary register with the upper byte of the addressed Flash location (not yet sent over the Epp)
- increments the Memory address register by 1.

Reading from address 7, when in word mode and odd address:

- launches a blind cycle to send the previous stored value of the auxiliary register. No Flash cycle is performed.
- increments the Memory address register by 1.

Memory Control Register

Bit	Function
0 (0x01)	Output enable (read strobe)
1 (0x02)	Write enable (write strobe) active LOW signals
2 (0x04)	RAM chip select (not used)
3 (0x08)	Flash chip select
4 (0x10)	Memory module during programming (memory is not available for custom applications)
5 (0x20)	Byte enable ('0') or word enable ('1')

The EPP interfaces can only handle 8-bit data. Word mode (16-bit) is only available for Automatic Read/Write, for both RAM (using register 6) and Flash (using register 7).

A "Word write" operation consists in two EPP (byte) cycles:

- the first one, at even address, launches a "blind cycle", which only stores the lower data byte in an auxiliary register.
- the second one, at odd address, combines the two data bytes to a data word on the memory data bus, and writes it to memory.

A "Word read" operation consists in two EPP (byte) cycles:

- the first one, at even address, reads the data word from memory, sends the lower data byte over the EPP bus and stores the upper byte in an auxiliary register.
- the second one, at odd address, launches a "blind cycle", which only sends the upper data byte over the EPP bus.

Manual mode is only allowed for Flash (Cellular RAM would hold CS active too long, blocking refresh cycles). Manual "Word" mode for flash reads the upper data bus byte for odd addresses and the lower data bus byte for even addresses. Manual "Byte" mode for flash reads the lower data bus byte for both odd and even addresses.

These features are completely transparent to the EPP host, which only needs to set the appropriate value in the Memory Control and memory address registers and read/write the data registers (4...7).

Port Definitions

clk	in, system clock (50MHz)
HandShakeReqOut	out, User Handshake Request
ctlMsmStartIn	in, Automatic process Start
ctlMsmDoneOut	out, Automatic process Done
ctlMsmDwrIn	in, Data Write pulse
ctlEppRdCycleIn	in, Indicates a READ EPP cycle
EppRdDataOut	out, Data Input bus
EppWrDataIn	in, Data Output bus
regEppAdrIn	in, = "00000000"; EPP Address Register content (bits 7:3 ignored)
ComponentSelect	in, active HIGH, selects the current MemCtrl instance.

If a single "client" component (CxMemCfg or other) is connected to a "host" component (EppCtrl or other), ComponentSelect signal can be held permanently active (connected to Vcc). When more "client" components (CxMemCfg or other) are connected to a "host" component (EppCtrl or other), the ComponentSelect input of each client must be synthesized by decoding the higher bits of regEppAdrOut bus, in such a way as to provide a distinct address range for each. The C1MemCfg component requires 8 EPP data registers (address range xxxxx000...xxxxx111)

Memory Bus Signals

MemDB	inout _vector(15 downto 0), -- Memory data bus
MemAdr	out _vector(23 downto 1), Memory Address bus
FlashByte	out, Byte enable('0') or word enable('1')
RamCS	out, RAM CS
FlashCS	out, Flash CS
MemWR	out, memory write
MemOE	out, memory read (Output Enable), also controls the MemDB direction
RamUB	out, RAM Upper byte enable
RamLB	out, RAM Lower byte enable
RamCre	out, Cfg Register enable
RamAdv	out, RAM Address Valid pin
RamClk	out, RAM Clock
RamWait	in, RAM Wait pin
FlashRp	out, Flash RP pin
FlashStSts	in, Flash ST-STs pin
MemCtrlEnabled	out, MemCtrl takes bus control

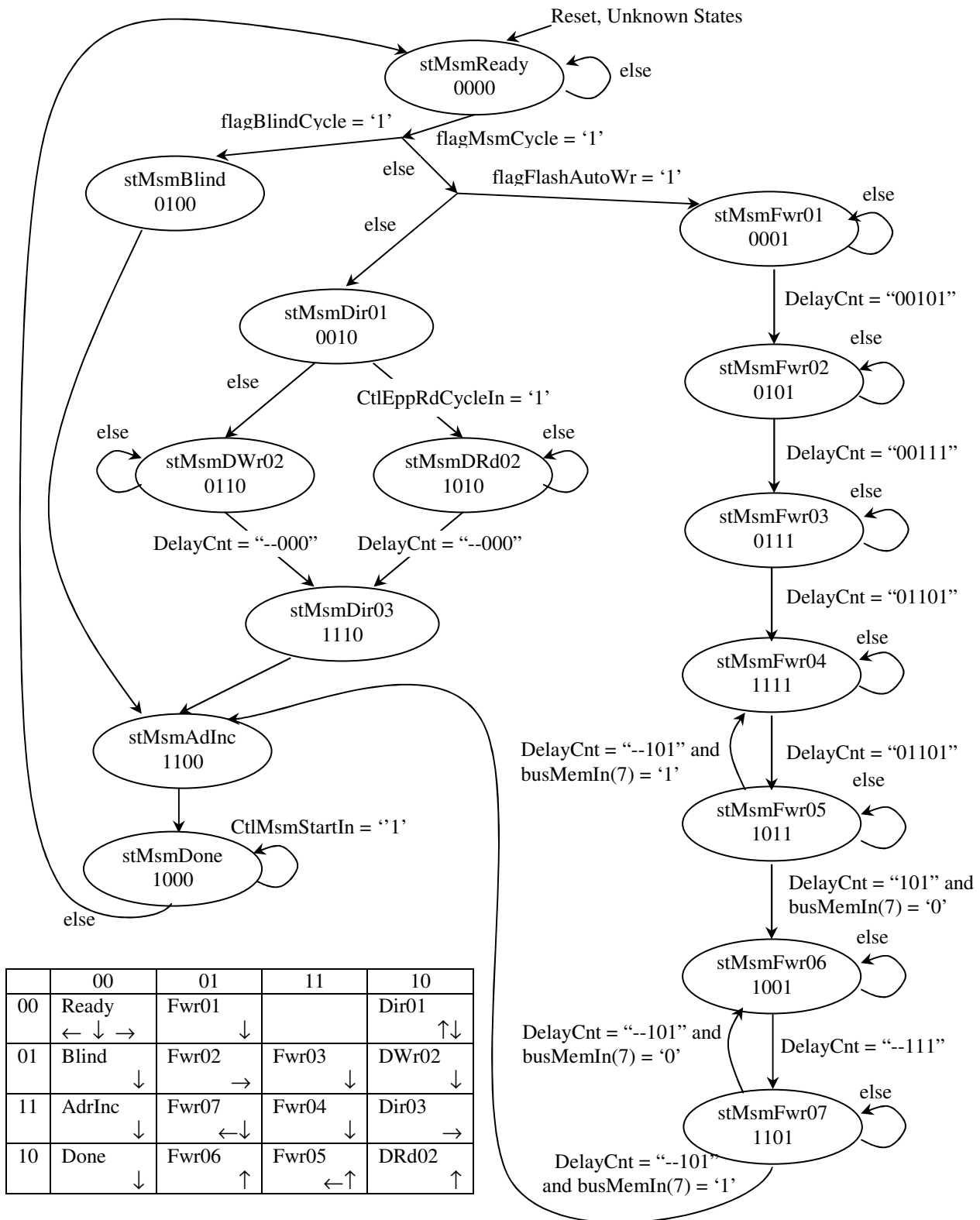


Figure 3 NexysOnBoardMemCtrl State Diagram and State Transition Table

OnBoard MemCfg User's Manual

The OnBoardMemCfg project allows read/write/erase operations on RAM/Flash 16-bit memory chips loaded on Digilent boards (like the Nexys and Nexys 2). This component is used in conjunction with the Digilent Adept suite software and the Digilent on-board USB circuitry to exchange data with an application running on a host PC and the on board memory. The application can be a Digilent program (like MemUtil, ExPort, or similar) or a custom application that uses Digilent .dll's to communicate via the USB port.

Using Digilent MemUtil Software to Configure the Memory Module

Using MemUtil to configure the memory module is very simple and intuitive. You should keep in mind the features provided by the Digilent board you use, like the size of RAM/Flash memory and the memory data bus width.

The OnBoardMemCfg project can work with Digilent boards loaded with 16-bit on-board memory (like the Nexys, Nexys 2, etc.) OnBoardMemCfg can use the on-board RAM (in both 16-bit and 8-bit mode) and on-board Flash memory in 16-bit mode. MemUtil configures OnBoardMemCfg for 16-bit mode both for Flash and RAM. Consequently, MemUtil can handle only block transfers of an even number of bytes with an even RAM or Flash start address.

The OnBoardMemCfg project contains a configuration register which sets the mode (16 or 8-bit) and contains an Enable bit (set to allow access to the component). MemUtil automatically sets the appropriate value in the configuration register for 16-bit mode, and enables the OnBoardMemCfg component.

The software functions are:

Properties. The user sets:

- Connection Properties. In the pull-down menu, the desired connection is selected. Multiple communication modules (Ethernet, USB, Serial) can be connected at the same time to the host PC. A MemUtil instance uses a single connection.
- Configure. The selected connection can be configured. For more information, see the *Adept User Manual*.
- Memory configuration. Selects the Digilent board type.
- Starting Register Address. MemUtil supports multiple memory modules attached to different motherboard connectors. For more details, see the C1memCfgX2 project. For the OnBoardMemCfg project, select *0x00*.

Load RAM. The user specifies:

- the source file
- the File Start Location
- the RAM start address
- the length of transferred data block (even number)
- the Verify option

Load Flash. Same as above, plus:

- the Auto Erase option. The software erases only the Flash blocks which are affected by the Load operation.

Store RAM. The user specifies:

- the RAM start address
- the length of transferred data block (even number)
- the destination file
- the file write option (Append, Replace, Overwrite)
- the File Start Location (only for Overwrite)

Store Flash. Same as above.

Erase Flash. The user specifies:

- the blocks selected for erase operation
- type of operation (Erase All or Erase Selected)

Multiple instances of MemUtil can run at the same time, on the same PC, sharing the same connection and system board. The system board needs to be configured with the OnBoardMemCfg project. Other Digilent applications (such as Digilent TransPort and Digilent ExPort) can also share the same connection.

Memory modules are identified by the starting register address. The OnBoardMemCfg project assigns the starting register address 0x00 to the memory module.

Successive operations can be launched through any active MemUtil instance. The user should allow a command to be fully executed before launching another one for the same communication module.

Multiple MemUtil instances can be used to control different board sets. For example, different USB channels can be connected each to its own set (a USB module and a system board). Each system board needs to be configured with the OnBoardMemCfg project. In this case, MemUtil instances are set to use different connections.

Using Digilent TransPort Software to Configure the Memory Module

The TransPort software allows the user to access the memory module at lower level commands. The routines used are the same as the ones used by MemUtil to implement the high level commands, as described above. TransPort is useful for debugging and for checking the command sequences before implementing them in a custom application.

The OnBoardMemCfg component can work with Digilent boards loaded with 16-bit on-board memory (like the Nexys, Nexys 2, etc.) OnBoardMemCfg can use the on-board RAM both in 16-bit and 8-bit mode and on-board Flash memory in 16-bit mode.

The OnBoardMemCfg component contains a configuration register which sets the mode (16 or 8-bit) and contains an Enable bit (which is set to allow access to the component). When using OnBoardMemCfg with TransPort or a custom application, the configuration register must be set with the appropriate value.

When using OnBoardMemCfg in 16-bit mode (either for Flash or RAM) any block transfer must contain an even number of bytes and start with an even RAM or Flash address.

There are two ways to access the on-board memory using the OnBoardMemCfg component in conjunction with TransPort or a custom application:

- low-level commands, where the memory bus cycle is performed step by step with multiple EPP transfers
- automatic Read/Write, where only the data byte is transmitted through EPP, the memory bus cycle is generated automatically by the OnBoardMemCfg component, and the memory address is auto-incremented after the data is transferred.

Low-level commands are not allowed for the RAM, since the timing would be too slow. The RAM chips loaded on Digilent boards are pseudostatic. The low-level commands' generated memory cycle would exceed the time specification for the internal refresh period.

The software functions are:

Properties. The user sets:

- Connection. In the pull-down menu, the desired connection is selected. Multiple communication modules (Ethernet, USB, Serial, on-board USB), can be connected at the same time to the host PC. A MemUtil instance uses a single connection.
- Configure. The selected connection can be configured. For more information, see the *Adept User Manual*.

Load File. Sends a file (fragment) to a specified EPP Data Register. The user specifies:

- the source file
- the File Start Location
- the Destination Register
- the Load Entire File option
- the length of the transferred data block (only if the Load Entire File option is not checked)
- the Load button launches the transfer.

Store File. Reads a specified EPP Data Register and saves data to a file (fragment). The user specifies:

- the destination file
- the File Output Mode (Append, Replace, Overwrite)
- the File Start Location (only for Overwrite)
- the Source Register
- the length of transferred data block
- the Store button launches the transfer.

Register I/O. Reads or writes a byte or group of bytes to/from a specified EPP Data Register. The user specifies:

- the EPP Register Address (for read and write operations)
- the data (for write operations)
- the read Display Format

There are eight lines, each able to define a register address and data. Their transfer can be performed line by line (by pressing the Read or Write button of the desired line) or as a full sequence (by pressing the Read All or Write All buttons).

Multiple instances of TransPort can run at the same time, on the same PC, sharing the same connection and system board. The system board needs to be configured with the OnBoardMemCfg project. Other Digilent applications (such as Digilent MemUtil and Digilent ExPort) can also share the same connection.

Memory modules are identified by the Register Address range. The OnBoardMemCfg project assigns the Register Address range 0x00 ...0x07 to the memory module.

Successive operations can be launched through any active TransPort instance. The user should allow a command to be fully executed before launching another one for the same communication module.

Multiple TransPort instances can be used to control different board sets. For example, different USB channels can be connected each to its own set (a USB module and a system board). Each system board needs to be configured with the OnBoardMemCfg project. In this case, TransPort instances are set to use different connections.

As described above, the TransPort software can be used to control any EPP interface implemented in the FPGA circuit. The following section shows how to access the specific on-board memory features. The information below is also useful as a guide for software developers, showing the low-level command sequence to be implemented using Adept API's in order to access on-board memory via the OnBoardMemCfg project.

Enabling the OnBoardMemCfg Component and Setting the 16 or 8-Bit Mode

The memory control register, regMemCtl, is an 8-bit read/write register, assigned to the EPP address MemCtrlReg = 0. (Reading these bits doesn't affect behavior, it only helps in debugging).

The meaning of the bits is as follows.

-- Memory control register

ctlMcrOe	<= regMemCtl(0); -- Output enable (read strobe)	– active '0'	initialized '1'
ctlMcrWr	<= regMemCtl(1); -- Write enable (write strobe)	– active '0'	initialized '1'
ctlMcrRAMCs	<= regMemCtl(2); -- RAM chip select	– active '0'	initialized '1'
ctlMcrFlashCs	<= regMemCtl(3); -- Flash chip select	– active '0'	initialized '1'
ctlMcrEnable	<= regMemCtl(4); -- MemCtrl	– enable ('1') or disable ('0')	initialized '0'
ctlMcrWord	<= regMemCtl(5); -- Byte enable	– ('0') or word enable ('1')	initialized '0'
not used	<= regMemCtl(7 downto 6); -- don't care (xx)		initialized '00'

Bits 0...3 directly affect the control lines of the RAM and Flash chips (these control lines are active '0'). Low-level commands (described in the sections below) directly set/reset these bits. When using the Automatic Read/Write registers (described in the sections below), the OnBoardMemCfg takes control over the control lines of the RAM and Flash chips. The internal Oe, Wr, and CS signals are ANDed with the bits 0...3 of the regMemCtl. Consequently, these bits are initialized and should be all '1' at any time when not used in a low-level command. It is important to avoid forbidden combinations of bits. For example, activating ctlMcrOe, ctlMcrWr, and either ctlMcrRAMCs or ctlMcrFlashCs at the same time would generate a data bus conflict between the FPGA and RAM or Flash chips. Activating ctlMcrOe, ctlMcrRAMCs, and ctlMcrFlashCs at the same time would generate a data bus conflict between the RAM and Flash chips.

Bit ctlMcrEnable is initialized '0' (inactive). Before accessing the memory, this bit should be set '1' (active). When not using the OnBoardMemCfg component to control the on-board memory chips,

ctlMcrEnable should be set back '0' (inactive). This way, the OnBoardMemCfg component releases the memory chips signals and the internal FPGA busses for other components in the FPGA project.

Bit ctlMcrWord is initialized '0' (byte enable). The user sets this bit according to the required data bus mode.

Enabling the OnBoardMemCfg component and setting the 16 or 8-bit mode could be done before or when first writing to address MemCtrlReg = 0. Care must be taken to keep the value of these bits with any write cycle at address MemCtrlReg = 0 when using low-level commands (described in the sections below). When using the Automatic Read/Write registers (described in the sections below), no write to address MemCtrlReg = 0 is required to perform the memory bus cycle. However, before using these commands, the user should enable the OnBoardMemCfg component and set the 16 or 8-bit mode:

To enable the OnBoardMemCfg component and set the 16-bit mode, write "xx111111" to the regMemCtl, at EPP address MemCtrlReg = 0).

To enable the OnBoardMemCfg component and set the 8-bit mode, write "xx011111" to the regMemCtl, at EPP address MemCtrlReg = 0).

Reading a Flash Location with Low-Level EPP Commands (8-Bit Mode)

Before attempting to read the Flash memory, make sure the Flash is in the Array Read mode. The Flash chip can be set in Array Read mode by either of the following actions:

- Power up reset.
- Activate the Reset pin (RP) (active LOW).
- Issue the READ ARRAY command.

Once in READ ARRAY mode, the Flash chip holds that mode until a different command is sent.

The user directly controls the memory busses (Data, Address and Control), using the Register I/O function. (If the Flash chip is already in READ ARRAY mode, skip step 1.)

1. Load Read Array command to the Memory Data Bus Register MemDB(7 downto 0)):
 - 1.1. Write the Read Array command (FFh) to EPP Register regMemWrData(7 downto 0) at EPP address MemDataWr = 4
2. Generate the write sequence on the memory control bus:
 - 2.1. Activate FlashCS and MemWr signals - write "xx010101" to the regMemCtl, at EPP address MemCtrlReg = 0)
 - 2.2. Deactivate MemWr signal - write "xx010111" to the regMemCtl, at EPP address MemCtrlReg = 0)
 - 2.3. Deactivate FlashCS signal - write "xx011111" to the regMemCtl, at EPP address MemCtrlReg = 0)
3. Load the Memory Address Bus Register (MemAdr(18 downto 0)):
 - 3.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 3.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
 - 3.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3 (for bytes keeping the previous value, the steps above can be skipped)

4. Generate the read sequence on the memory control bus:
 - 4.1. Activate FlashCS and MemOE signals - write “xx010110” to the regMemCtl, at EPP address MemCtrlReg = 0)
5. Read the Memory Data Bus Register MemDB(7 downto 0)):
 - 5.1. Read the EPP Register regMemRdData(7 downto 0) at EPP address MemDataRd = 5
6. Finish the read sequence on the memory control bus:
 - 6.1. Deactivate FlashCS and MemOE signals - write “xx011111” to the regMemCtl, at EPP address MemCtrlReg = 0)

Reading a Flash Location with Low-Level EPP Commands (16-Bit Mode)

Before attempting to read the Flash memory, make sure the Flash is in the Array Read mode. The Flash chip can be set in Array Read mode by either of the following actions:

- Power up reset.
- Activate the Reset pin (RP) (active LOW).
- Issue the READ ARRAY command.

Once in READ ARRAY mode, the Flash chip holds that mode until a different command is sent. (If the Flash chip is already in READ ARRAY mode, skip step 1.)

1. Load Read Array command to the Memory Data Bus Register MemDB(7 downto 0)):
 - 1.1. Write the Read Array command (FFh) to EPP Register regMemWrData(7 downto 0) at EPP address MemDataWr = 4
2. Generate the write sequence on the memory control bus:
 - 2.1. Activate FlashCS and MemWr signals - write “xx110101” to the regMemCtl, at EPP address MemCtrlReg = 0)
 - 2.2. Deactivate MemWr signal - write “xx110111” to the regMemCtl, at EPP address MemCtrlReg = 0)
 - 2.3. Deactivate FlashCS signal - write “xx111111” to the regMemCtl, at EPP address MemCtrlReg = 0)
3. Load the Memory Address Bus Register (MemAdr(18 downto 0)):
 - 3.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 3.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
 - 3.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3 (for bytes keeping the previous value, the steps above can be skipped)
4. Generate the read sequence on the memory control bus:
 - 4.1. Activate FlashCS and MemOE signals - write “xx110110” to the regMemCtl, at EPP address MemCtrlReg = 0)
5. Read the Memory Data Bus Register MemDB(7 downto 0)):
 - 5.1. Read the EPP Register regMemRdData(7 downto 0) at EPP address MemDataRd = 5
6. Finish the read sequence on the memory control bus:
 - 6.1. Deactivate FlashCS and MemOE signals - write “xx111111” to the regMemCtl, at EPP address MemCtrlReg = 0)

Writing a Flash Location with Low-Level EPP Commands (8-Bit Mode)

Not supported.

Writing a Flash Location with Low-Level EPP Commands (16-Bit Mode)

The user directly controls the memory busses (Data, Address and Control), using the Register I/O function.

1. Load the Memory Address Bus Register (MemAdr(18 downto 0)) with an even value (for lower byte of the word):
 - 1.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 1.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
 - 1.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3
(for bytes keeping the previous value, the steps above can be skipped)
2. Load Write Command to the Memory Data Bus Register MemDB(7 downto 0)):
 - 2.1. Write the Write Command (40h) to EPP Register regMemWrData(7 downto 0) at EPP address MemDataWr = 4
3. Generate the write sequence on the memory control bus:
 - 3.1. Activate FlashCS and MemWr signals - write "xx110101" to the regMemCtl, at EPP address MemCtrlReg = 0)
 - 3.2. Deactivate MemWr signal - write "xx110111" to the regMemCtl, at EPP address MemCtrlReg = 0)
 - 3.3. Deactivate FlashCS signal - write "xx111111" to the regMemCtl, at EPP address MemCtrlReg = 0)
4. Load data to the Memory Data Bus Register MemDB(7 downto 0)):
 - 4.1. Write the data byte to EPP Register regMemWrData(7 downto 0) at EPP address MemDataWr = 4
5. Load the Memory Address Bus Register (MemAdr(18 downto 0)) with an odd value (incremented by 1, for higher byte of the word):
 - 5.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 5.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
 - 5.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3
(for bytes keeping the previous value, the steps above can be skipped)
6. Load data to the Memory Data Bus Register MemDB(15 downto 8)):
 - 6.1. Write the data byte to EPP Register regMemWrData(15 downto 8) at EPP address MemDataWr = 4
7. Generate the write sequence on the memory control bus:
 - 7.1. Activate FlashCS and MemWr signals - write "xx110101" to the regMemCtl, at EPP address MemCtrlReg = 0)
 - 7.2. Deactivate MemWr signal - write "xx110111" to the regMemCtl, at EPP address MemCtrlReg = 0)
 - 7.3. Deactivate FlashCS signal - write "xx111111" to the regMemCtl, at EPP address MemCtrlReg = 0)

When manually writing a data byte using TransPort software, steps 8 to 15 below are not necessary (in fact, it is not possible to perform them fast enough.) They check a status bit of the Flash chip, which pulses low, while the internal Flash logic performs the write operation. Typical pulse length is

2 μ s. The steps are posted here as a reference to be implemented in a software or HDL component, which automatically performs the write sequence.

8. Generate the read sequence on the memory control bus:
 - 8.1. Activate FlashCS and MemOE signals - write "xx110110" to the regMemCtl, at EPP address MemCtrlReg = 0)
9. Read the Flash Status Register (SR) on the Memory Data Bus Register MemDB(7 downto 0)):
 - 9.1. Read the EPP Register regMemRdData(7 downto 0) at EPP address MemDataRd = 5
10. Finish the read sequence on the memory control bus:
 - 10.1. Deactivate FlashCS and MemOE signals - write "xx111111" to the regMemCtl, at EPP address MemCtrlReg = 0)
11. Check the SR7 bit (MSB of SR). Loop back to 8, if SR7 = 1.
12. Generate the read sequence on the memory control bus:
 - 12.1. Activate FlashCS and MemOE signals - write "xx110110" to the regMemCtl, at EPP address MemCtrlReg = 0)
13. Read the Flash Status Register (SR) on the Memory Data Bus Register MemDB(7 downto 0)):
 - 13.1. Read the EPP Register regMemRdData(7 downto 0) at EPP address MemDataRd = 5
14. Finish the read sequence on the memory control bus:
 - 14.1. Deactivate FlashCS and MemOE signals - write "xx111111" to the regMemCtl, at EPP address MemCtrlReg = 0)
15. Check the SR7 bit (MSB of SR). Loop back to 12, if SR7 = 0.

Erasing a Flash Block with Low-Level EPP Commands

The mode (16-bit or 8-bit) does not matter for erase operations. The user directly controls the memory busses (Data, Address and Control), using the Register I/O function.

1. Load the Memory Address Bus Register (MemAdr(18 downto 0)) – any address in the block to be erased:
 - 1.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 1.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
 - 1.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3
(for bytes keeping the previous value, the steps above can be skipped)
2. Load Erase Command to the Memory Data Bus Register MemDB(7 downto 0)):
 - 2.1. Write the Erase Command (20h) to EPP Register regMemWrData(7 downto 0) at EPP address MemDataWr = 4
3. Generate the write sequence on the memory control bus:
 - 3.1. Activate FlashCS and MemWr signals - write "xxx10101" to the regMemCtl, at EPP address MemCtrlReg = 0)
 - 3.2. Deactivate MemWr signal - write "xxx10111" to the regMemCtl, at EPP address MemCtrlReg = 0)
 - 3.3. Deactivate FlashCS signal - write "xxx11111" to the regMemCtl, at EPP address MemCtrlReg = 0)
4. Load Confirm Erase Command to the Memory Data Bus Register MemDB(7 downto 0)):
 - 4.1. Write the Confirm Erase Command (D0h) to EPP Register regMemWrData(7 downto 0) at EPP address MemDataWr = 4
5. Generate the write sequence on the memory control bus:

- 5.1. Activate FlashCS and MemWr signals - write "xxx10101" to the regMemCtl, at EPP address MemCtrlReg = 0)
- 5.2. Deactivate MemWr signal - write "xxx10111" to the regMemCtl, at EPP address MemCtrlReg = 0)
- 5.3. Deactivate FlashCS signal - write "xxx11111" to the regMemCtl, at EPP address MemCtrlReg = 0)

When manually writing a data byte using TransPort software, steps 6 to 13 below are not necessary (in fact, it's not possible to perform them fast enough.) They check a status bit of the Flash chip, which pulses low, while the internal Flash logic performs the write operation. Typical pulse length is 200-500ms. The steps are posted here as a reference to be implemented in a software or HDL component, which automatically performs the erase sequence.

6. Generate the read sequence on the memory control bus:
 - 6.1. Activate FlashCS and MemOE signals - write "xxx10110" to the regMemCtl, at EPP address MemCtrlReg = 0)
7. Read the Flash Status Register (SR) on the Memory Data Bus Register MemDB(7 downto 0)):
 - 7.1. Read the EPP Register regMemRdData(7 downto 0) at EPP address MemDataRd = 5
8. Finish the read sequence on the memory control bus:
 - 8.1. Deactivate FlashCS and MemOE signals - write "xxx11111" to the regMemCtl, at EPP address MemCtrlReg = 0)
9. Check the SR7 bit (MSB of SR). Loop back to 6, if SR7 = 1.
10. Generate the read sequence on the memory control bus:
 - 10.1. Activate FlashCS and MemOE signals - write "xxx10110" to the regMemCtl, at EPP address MemCtrlReg = 0)
11. Read the Flash Status Register (SR) on the Memory Data Bus Register MemDB(7 downto 0)):
 - 11.1. Read the EPP Register regMemRdData(7 downto 0) at EPP address MemDataRd = 5
12. Finish the read sequence on the memory control bus:
 - 12.1. Deactivate FlashCS and MemOE signals - write "xxx11111" to the regMemCtl, at EPP address MemCtrlReg = 0)
13. Check the SR7 bit (MSB of SR). Loop back to 10, if SR7 = 0.

Reading a RAM Location Using the Automatic RAM Read Register (8-Bit Mode)

1. Enable the OnBoardMemCfg component and set the 8-bit mode:
 - 1.1. Write "xx011111" to the regMemCtl, at EPP address MemCtrlReg = 0)
2. Load the Memory Address Bus Register (MemAdr(18 downto 0)):
 - 2.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 2.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
 - 2.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3
(for bytes keeping the previous value, the steps above can be skipped)
3. Read the regMemRdData(7 downto 0) register at address RamAutoRW = 6. C1MemCtrl automatically generates the read sequence and increments the MemAdr register.
4. Repeat step 3 to read successive RAM locations.

Reading a RAM Location Using the Automatic RAM Read Register (16-Bit Mode)

1. Enable the OnBoardMemCfg component and set the 16-bit mode:
 - 1.1. Write “xx111111” to the regMemCtl, at EPP address MemCtrlReg = 0)
2. Load the Memory Address Bus Register (MemAdr(18 downto 0)) with an even value:
 - 2.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 2.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
 - 2.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3
(for bytes keeping the previous value, the steps above can be skipped)
3. Read the regMemRdData(7 downto 0) register at address RamAutoRW = 6. C1MemCtrl automatically generates the read sequence, reads from RAM memory both lower and upper data bytes, sends the lower data byte via EPP, stores the upper data byte in an intermediate register and increments the MemAdr register.
4. Read the regMemRdData(7 downto 0) register at address RamAutoRW = 6. C1MemCtrl sends the upper data byte from the intermediate register and increments the MemAdr register. No RAM read cycle is generated.
5. Loop from step 3 to read successive RAM locations.

Reading a Flash Location Using the Automatic Flash Read Register (8-Bit Mode)

Before attempting to read the Flash memory, make sure the Flash is in the Array Read mode. The Flash chip can be set in Array Read mode by either of the following actions:

- Power up reset.
- Activate the Reset pin (RP) (active LOW).
- Issue the READ ARRAY command.

Once in READ ARRAY mode, the Flash chip holds that mode until a different command is sent. (If the Flash chip is already in READ ARRAY mode, skip step 1.)

1. Load Read Array command to the Memory Data Bus Register MemDB(7 downto 0)):
 - 1.1. Write the Read Array command (FFh) to EPP Register regMemWrData(7 downto 0) at EPP address MemDataWr = 4
2. Enable the OnBoardMemCfg component and set the 8-bit mode:
 - 2.1. Write “xx011111” to the regMemCtl, at EPP address MemCtrlReg = 0)
3. Load the Memory Address Bus Register (MemAdr(18 downto 0)):
 - 3.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 3.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
 - 3.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3 (for bytes keeping the previous value, the steps above can be skipped)
4. Read the regMemRdData(7 downto 0) register at address RamAutoRW = 7. C1MemCtrl automatically generates the read sequence and increments the MemAdr register. Repeat step 4 to read successive Flash locations.

Reading a Flash Location Using the Automatic Flash Read Register (16-Bit Mode)

Before attempting to read the Flash memory, make sure the Flash is in the Array Read mode. The Flash chip can be set in Array Read mode by either of the following actions:

- Power up reset.
- Activate the Reset pin (RP) (active LOW).
- Issue the READ ARRAY command.

Once in READ ARRAY mode, the Flash chip holds that mode until a different command is sent. (If the Flash chip is already in READ ARRAY mode, skip step 1.)

1. Load Read Array command to the Memory Data Bus Register MemDB(7 downto 0)):
 - 1.1. Write the Read Array command (FFh) to EPP Register regMemWrData(7 downto 0) at EPP address MemDataWr = 4
2. Enable the OnBoardMemCfg component and set the 16-bit mode:
 - 2.1. Write "xx111111" to the regMemCtl, at EPP address MemCtrlReg = 0)
3. Load the Memory Address Bus Register (MemAdr(18 downto 0)) with an even value:
 - 3.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 3.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
 - 3.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3 (for bytes keeping the previous value, the steps above can be skipped)
4. Read the regMemRdData(7 downto 0) register at address FlashAutoRW = 7. C1MemCtrl automatically generates the read sequence, reads from Flash memory both lower and upper data bytes, sends the lower data byte via EPP, stores the upper data byte in an intermediate register and increments the MemAdr register.
5. Read the regMemRdData(7 downto 0) register at address FlashAutoRW = 7. C1MemCtrl sends the upper data byte from the intermediate register and increments the MemAdr register. No Flash read cycle is generated.
6. Loop from step 4 to read successive Flash locations.

Writing a RAM Location Using the Automatic RAM Write Register (8-Bit Mode)

1. Enable the OnBoardMemCfg component and set the 8-bit mode:
 - 1.1. Write "xx011111" to the regMemCtl, at EPP address MemCtrlReg = 0)
2. Load the Memory Address Bus Register (MemAdr(18 downto 0)):
 - 2.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 2.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
 - 2.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3
(for bytes keeping the previous value, the steps above can be skipped)
3. Write data byte to the regMemRdData(7 downto 0) register at address RamAutoRW = 6. C1MemCtrl automatically generates the write sequence and increments the MemAdr register.
4. Repeat step 3 to write successive RAM locations.

Writing a RAM Location Using the Automatic RAM Write Register (16-Bit Mode)

1. Enable the OnBoardMemCfg component and set the 16-bit mode:
 - 1.1. Write “xx111111” to the regMemCtl, at EPP address MemCtrlReg = 0)
2. Load the Memory Address Bus Register (MemAdr(18 downto 0)) with an even value:
 - 2.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 2.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
 - 2.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3
(for bytes keeping the previous value, the steps above can be skipped)
3. Write lower data byte to the regMemRdData(7 downto 0) register at address RamAutoRW = 6. C1MemCtrl stores the lower byte in an intermediate register and increments the MemAdr register. No RAM write cycle is generated.
4. Write upper data byte to the regMemRdData(7 downto 0) register at address RamAutoRW = 6. C1MemCtrl automatically generates the write sequence to write both lower and upper data bytes and increments the MemAdr register.
5. Repeat steps 3 and 4 to write successive RAM locations.

Writing a Flash Location Using the Automatic Flash Write Register (8-Bit Mode)

Not supported.

Writing a Flash Location Using the Automatic Flash Write Register (16-Bit Mode)

1. Enable the OnBoardMemCfg component and set the 16-bit mode:
 - 1.1. Write “xx111111” to the regMemCtl, at EPP address MemCtrlReg = 0)
2. Load the Memory Address Bus Register (MemAdr(18 downto 0)) with an even value:
 - 2.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 2.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
 - 2.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3
(for bytes keeping the previous value, the steps above can be skipped)
3. Write the lower data byte to the regMemRdData(7 downto 0) register at address FlashAutoRW = 7. C1MemCtrl stores the lower byte in an intermediate register and increments the MemAdr register. No Flash write cycle is generated.
4. Write higher data byte to the regMemRdData(7 downto 0) register at address FlashAutoRW = 7. C1MemCtrl automatically generates the write sequence to write both lower and upper data bytes and increments the MemAdr register. The Flash chip enters the READ STATUS mode.
5. Repeat steps 3 and 4 to write successive Flash locations.

Storing the RAM Content to a File Using the Store File Function (8-Bit Mode)

1. Enable the OnBoardMemCfg component and set the 8-bit mode:
 - 1.1. Write “xx011111” to the regMemCtl, at EPP address MemCtrlReg = 0)
2. Load the Memory Address Bus Register (MemAdr(18 downto 0)):
 - 2.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1

- 2.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
- 2.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3
(for bytes keeping the previous value, the steps above can be skipped)
3. Select the Store File tab of TransPort software. Fill in the required fields. Select Source Register 6 (RamAutoRW). Press the Store button.

Storing the RAM Content to a File Using the Store File Function (16-Bit Mode)

1. Enable the OnBoardMemCfg component and set the 16-bit mode:
 - 1.1. Write "xx111111" to the regMemCtl, at EPP address MemCtrlReg = 0)
2. Load the Memory Address Bus Register (MemAdr(18 downto 0)) with an even value:
 - 2.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 2.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
 - 2.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3
(for bytes keeping the previous value, the steps above can be skipped)
3. Select the Store File tab of TransPort software. Fill in the required fields. Set an even value for the Length. Select Source Register 6 (RamAutoRW). Press the Store button.

Storing the Flash Content to a File Using the Store File Function (8-Bit Mode)

Before attempting to read the Flash memory, make sure the Flash is in the Array Read mode. The Flash chip can be set in Array Read mode by either of the following actions:

- Power up reset.
- Activate the Reset pin (RP) (active LOW).
- Issue the READ ARRAY command.

Once in READ ARRAY mode, the Flash chip holds that mode until a different command is sent. (If the Flash chip is already in READ ARRAY mode, skip step 1.)

1. Load Read Array command to the Memory Data Bus Register MemDB(7 downto 0)):
 - 1.1. Write the Read Array command (FFh) to EPP Register regMemWrData(7 downto 0) at EPP address MemDataWr = 4
2. Enable the OnBoardMemCfg component and set the 8-bit mode:
 - 2.1. Write "xx011111" to the regMemCtl, at EPP address MemCtrlReg = 0)
3. Load the Memory Address Bus Register (MemAdr(18 downto 0)):
 - 3.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 3.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
 - 3.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3
(for bytes keeping the previous value, the steps above can be skipped)
4. Select the Store File tab of the TransPort software. Fill in the required fields. Select Source Register 7 (FlashAutoRW). Press the Store button.

Storing the Flash Content to a File Using the Store File Function (16-Bit Mode)

Before attempting to read the Flash memory, make sure the Flash is in the Array Read mode. The Flash chip can be set in Array Read mode by either of the following:

- Power up reset.
- Activate the Reset pin (RP) (active LOW).
- Issue the READ ARRAY command.

Once in READ ARRAY mode, the Flash chip holds that mode until a different command is sent. (If the Flash chip is already in READ ARRAY mode, skip step 1.)

1. Load Read Array command to the Memory Data Bus Register MemDB(7 downto 0):
 - 1.1. Write the Read Array command (FFh) to EPP Register regMemWrData(7 downto 0) at EPP address MemDataWr = 4
2. Enable the OnBoardMemCfg component and set the 16-bit mode:
 - 2.1. Write "xx111111" to the regMemCtl, at EPP address MemCtrlReg = 0)
3. Load the Memory Address Bus Register (MemAdr(18 downto 0)) with an even value:
 - 3.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 3.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
 - 3.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3
(for bytes keeping the previous value, the steps above can be skipped)
4. Select the Store File tab of the TransPort software. Fill in the required fields. Set an even value for the Length. Select Source Register 7 (FlashAutoRW). Press the Store button.

Loading a File to the RAM Using the Load File Function (8-Bit Mode)

1. Enable the OnBoardMemCfg component and set the 8-bit mode:
 - 1.1. Write "xx011111" to the regMemCtl, at EPP address MemCtrlReg = 0)
2. Load the Memory Address Bus Register (MemAdr(18 downto 0)):
 - 2.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 2.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
 - 2.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3
(for bytes keeping the previous value, the steps above can be skipped)
3. Select the Load File tab of the TransPort software. Fill in the required fields. Select Source Register 6 (RamAutoRW). Press the Load button.

Loading a File to the RAM Using the Load File Function (16-Bit Mode)

1. Enable the OnBoardMemCfg component and set the 16-bit mode:
 - 1.1. Write "xx111111" to the regMemCtl, at EPP address MemCtrlReg = 0)
2. Load the Memory Address Bus Register (MemAdr(18 downto 0)) with an even value:
 - 2.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 2.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2

- 2.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3
(for bytes keeping the previous value, the steps above can be skipped)
3. Select the Load File tab of the TransPort software. Fill in the required fields. Set an even value for the Length. Select Source Register 6 (RamAutoRW). Press the Load button.

Loading a File to the Flash Using the Load File Function (8-Bit Mode)

Not supported.

Loading a File to the Flash Using the Load File Function (16-Bit Mode)

1. Enable the OnBoardMemCfg component and set the 16-bit mode:
 - 1.1. Write "xx111111" to the regMemCtl, at EPP address MemCtrlReg = 0)
2. Load the Memory Address Bus Register (MemAdr(18 downto 0)) with an even value:
 - 2.1. Write the lowest address byte to EPP Register MemAdr(7 downto 0) at EPP address MemAdrL = 1
 - 2.2. Write the middle address byte to EPP Register MemAdr(15 downto 8) at EPP address MemAdrM = 2
 - 2.3. Write the highest address byte to EPP Register MemAdr(18 downto 16) at EPP address MemAdrH = 3
(for bytes keeping the previous value, the steps above can be skipped)
3. Select the Load File tab of the TransPort software. Fill in the required fields. Set an even value for the Length. Select Source Register 7 (FlashAutoRW). Press the Load button. The Flash chip enters the READ STATUS mode.

Using Digilent Adept SDK API Functions to Configure the Memory Module

Digilent Adept SDK provides API functions to exchange data with a project implemented in the gate-array. The TransPort application calls these API functions to perform the EPP data write and read at different addresses. These APIs are enumerated below for both versions of Adept. A user program can use these API functions to perform the steps described previously.

Adept SDK

Download the Digilent Adept SDK and read the *Digilent Port Communications Programmers Reference Manual*.

Before performing any EPP transfer, open the device with *DpcOpenData* API and after finishing the transfer(s) close the device with *DpcCloseData*.

To replicate the actions described in previously, use the following APIs:

1. To write a value to the EPP register at the EPP address:
use *DpcPutReg* API with the address in the *bAddr* argument and the register value in the *bData* argument.
2. To read the value from an EPP register at the EPP address:
use *DpcGutReg* API with the address in the *bAddr* argument and the pointer to the register value in the *pbData* argument.
3. To load a file:
use *DpcPutRegRepeat* with the address in the *bAddr* argument, the number of data bytes to write in the *cbData* argument, and the buffer of bytes in the *rgbData* argument.
4. To store a file:
use *DpcGutRegRepeat* with the address in the *bAddr* argument, the number of data bytes to read in the *cbData* argument, and the buffer for the bytes to return in the *rgbData* argument.

Adept2 SDK

Download the Digilent Adept 2 SDK and read the *Digilent Adept 2 Programmers Reference Manual*.

Before performing any EPP transfer, open the device with *DmgrOpen* or *DmgrOpenEx* and the EPP protocol enabled with the *DeppEnable* APIs. After finishing the transfer(s), disable the EPP protocol with *DeppDisable* and close the device with *DmgrClose* APIs.

To replicate the actions described previously, use the following APIs:

1. To write a value to the EPP register at the EPP address:
use *DmgrPutReg* API with the address in the *bAddr* argument and register the value in the *bData* argument.
2. To read the value from an EPP register at the EPP address:
use *DmgrGutReg* API with the address in the *bAddr* argument and pointer to register the value in the *pbData* argument.
3. To load a file:
use *DmgrPutRegRepeat* with the address in the *bAddr* argument, the number of data bytes to write in the *cbData* argument, and the buffer of bytes in the *rgbData* argument.
4. To store a file:
use *DmgrGutRegRepeat* with the address in the *bAddr* argument, the number of data bytes to read in the *cbData* argument, and the buffer for the bytes to return in the *rgbData* argument.