PLX PCI9054 Local Bus Master/Target Interface Design

D. W. Hawkins (dwh@ovro.caltech.edu)
Revision: 1.13

June 27, 2010

Contents

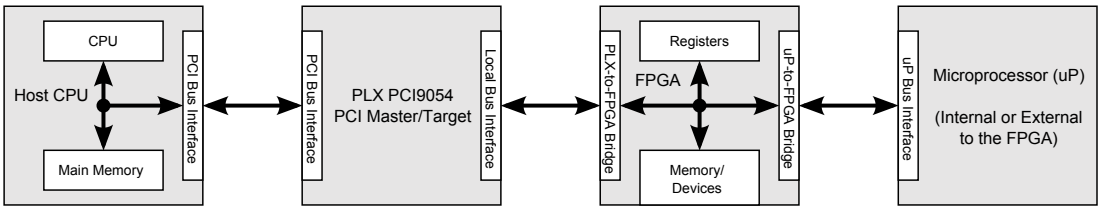


Figure 1: PLX PCI9054 PCI I/O Accelerator example application.

1 Introduction

The PLX Technologies PCI9054 *PCI I/O Accelerator* [3] is a 32-bit, 33MHz, PCI bus master/target interface. The PCI9054 translates the relatively complex PCI bus protocol into a simpler local bus protocol. The PCI9054 contains a DMA controller that supports sustained burst transactions over the PCI bus, and FIFO logic that allows the local bus to operate at a different frequency than the PCI bus (local bus clock frequencies up to 50MHz).

Figure 1 shows a block diagram of an example application of the PCI9054. The PCI9054 provides a PCI interface and DMA controller to an FPGA interfaced to the PCI9054 local bus. The use of the PCI9054 eliminates the need for the PCI interface and DMA logic in the FPGA. This reduces the resource and (potentially) the timing requirements of the FPGA. The FPGA can optionally implement a processor, or interface to an external processor. The PCI9054 provides PCI bus mastering capabilities to this processor, and provides a DMA engine for offloading data transfer tasks.

This document provides details on interfacing an FPGA to the PCI9054 local bus. VHDL code is provided for;

- Local bus master *bus functional model* (BFM);
  - To generate local bus transactions.
- FPGA local bus targets
  - Provide a host CPU or the PCI9054 DMA controller access to FPGA registers or RAM.
- FPGA local bus master/target interface (FPGA bridge)
  - Provide a host CPU or the PCI9054 DMA controller access to FPGA registers or RAM.
  - Provide an FPGA master access to the PCI9054 registers or the PCI bus.

Table 1: PLX PCI9054 Master/Target transactions.

Transaction	PCI Bus		Local Bus	
	Host CPU	PCI9054	PCI9054	FPGA bridge
Host CPU access to PCI9054 registers	Master	Target		
Host CPU access to FPGA registers	Master	Target	Master	Target
PCI9054 DMA	Target	Master	Master	Target
FPGA PCI access	Target	Master	Target	Master
FPGA access to PCI9054 registers			Target	Master

2 Local bus interfacing

2.1 Master/Target transactions

The PCI bus defines two board types; a system or host board (the source of PCI bus clocks and arbitration) and peripheral boards. PCI bus transactions are started by an initiator or *master* and are directed to a slave or *target*. Host CPU boards incorporate bus master/target interfaces, whereas peripheral boards may support either target-only, or master/target interfaces. A board design containing a PCI9054 can support PCI bus master/target operation, with a maximum transfer rate over a 32-bit, 33MHz, PCI bus of $4\,\text{bytes} \times 33 \times 10^6/2^{20} = 125.9\text{MB/s}$.

Figure 1 shows a block diagram of a system containing a board with a PCI9054 interfaced to an FPGA. Table 1 shows the master/target transactions possible on the PCI and local buses. The PCI9054 provides master/target capability on the PCI bus, however, master/target support on the local bus is optional. If the FPGA does not contain a local processor, or has no need to access the PCI9054 registers or the PCI bus, then local bus mastering is not required, hence a local bus target-only interface can be used. Regardless of whether the FPGA local bus interface is a master/target or target-only interface, high-performance PCI bus master transactions are supported via the use of the PCI9054 DMA controller.

An example application of the local bus master/target capabilities are;

- The PCI9054 DMA controller is used to transfer data between the host CPU main memory and on-board memory.
- The FPGA implements a burst capable target interface to support high-performance DMA transactions.
- (Optional) The FPGA implements a master interface to provide access to the PCI9054 internal registers, or access to the PCI bus.

  If the FPGA master interface is only used to access to the PCI9054 registers, then the bus master logic can be simplified by only implementing single transaction support.

2.2 Interface signals (J-mode)

The PCI9054 local bus supports three interface modes; M, J, and C. The modes are provided to support direct-connection to local processor buses. For an FPGA application, the J-mode, with multiplexed address/data bus, results in the lowest number of interface signals. Table 2 shows the signals required by an FPGA interface to the PCI9054 local bus. The J-mode of the PCI9054 is selected by using a mode select pin setting of `MSEL[1:0] = 01b` (p1-13 [3]). In J-mode, the `ALE` pin is not required for use by the FPGA interface, however, the pin requires a pull-down for the device to function correctly (see the note on p12-1 [3]). The PCI9054 pin functions for J-Mode are detailed in Section 12 of the data book [3].

2.3 Bus arbitration

The PCI9054 defaults to being a target on the local bus. In J-mode, the PCI9054 local bus master interface arbitrates for the local bus by asserting the hold request signal, `LHOLD`, and waiting for assertion of the hold acknowledge signal, `LHOLDA` (p4-2 and p5-44 [3]). If the FPGA implements a target-only interface, the PCI9054 is the only local bus master, so the arbiter can be implemented via a pull-up on `HOLDA`, or by having the FPGA register the `HOLD` input to generate the `HOLDA` output (to match the signal timing on p5-44 [3]). If the FPGA implements a master/target interface, then the FPGA master must also arbitrate for the local bus, so a two device arbiter is required internal to the FPGA.

If the PCI9054 local bus master interface is used to configure the FPGA, a pull-up is required on the `HOLDA` signal, so that local bus accesses succeed when the FPGA is not configured. Once the FPGA is configured, arbiter logic internal to the FPGA can control `HOLDA`.

Table 2: FPGA interface to the PLX PCI9054 local bus.

FPGA name	PCI9054 name	FPGA direction		Description
		Master	Target	
Bus arbitration				
plx_hold	LHOLD	I	I	Bus hold (request)
plx_holda	LHOLDA	O	O	Bus hold acknowledge
plx_breqo	BREQo	I	I	PCI9054 requires the local bus
plx_breqi	BREQi	O	O	Local bus master bus requires the bus
Data transfer				
plx_clk	LCLK	I	I	Local bus clock
plx_ccsN	LCCS#	O	O	Configuration register select
plx_adsN	LADS#	O	I	Address strobe
plx_wr_rdN	W/R#	O	I	Write/read
plx_lastN	BLAST#	O	I	Last data phase
plx_waitN	LWAIT#	O	I	Master wait-state
plx_rdyN	LREADY#	I	O	Target wait-state
plx_termN	BTERM#	I	O	Target burst terminate request
plx_dp[3:0]	DP	O	I	Parity
plx_beN[3:0]	LBE#	O	I	Byte-enables
plx_ad[31:0]	LAD	I/O	I/O	Multiplexed address/data bus
Interrupts				
plx_intN	LINT#	I/O	I/O	Interrupt request
plx_serrN	SERR#	I/O	I/O	System error
User/DMA (multifunction signals)				
plx_deN	DMAPAF	I	I	PCI initiator write FIFO almost full
	EOT#	O	O	Terminate the current DMA transfer
plx_udloN	USERo	I	I	User output
	DREQ#	O	O	Demand-mode DMA request
	LLOCKo#	I	I	Local bus lock output
plx_udliN	USERi	O	O	User input
	DACK#	I	I	Demand-mode DMA acknowledge
	LLOCKi#	O	O	Local bus lock input

Table 3: PLX PCI9054 local bus timing parameters (J-mode).

PCI9054 name	PCI9054 direction		Timing for			
	Master	Target	Input (ns)		Output (ns)	
			t_{su}	t_h	$t_{co(min)}$	$t_{co(max)}$
Bus arbitration						
LHOLD	O	O			5.0	10.0
LHOLDA	I	I	7.0	1.0		
BREQo	O	O			5.0	8.5
BREQi	I	I	5.0	1.0		
Data transfer						
LCCS#		I	1.5	1.0		
LADS#	O	I	5.0	1.0	5.0	10.0
W/R#	O	I	8.5	1.0	5.0	12.0
BLAST#	O	I	6.5	1.0	5.0	12.5
LWAIT#	O	I	7.0	1.0	5.0	10.5
LREADY#	I	O	9.5	1.0	5.0	9.5
BTERM#	I	O	9.5	1.0	5.0	10.0
DP	O	I	3.0	1.0	5.0	10.0
LBE#	O	I	9.0	1.0	5.0	10.0
LAD	I/O	I/O	6.5	1.0	5.0	11.0
User/DMA (multifunction signals)						
DMAPAF	O	O			5.0	13.0
EOT#	I	I	8.5	1.0		
USERo	O	O			5.0	9.5
DREQ#	I	I	7.0	1.0		
LLOCKo#	O	O			5.0	9.5
USERi	I	I	7.0	1.0		
DACK#	O	O			5.0	10.5
LLOCKi#	I	I	7.0	1.0		

2.4 Timing parameters

Table 3 contains the local bus timing parameters in J-mode. These timing parameters are used to constrain FPGA timing during place-and-route.

2.5 PCI9054 configuration

The local bus transaction signal operation depends on the following PCI9054 configuration register settings (in addition to the device being configured in J-mode);

- `LREADY#`
  - This signal is the target wait-state control.
  - For PCI9054 Master/FPGA Target transfers the signal is deasserted by the FPGA to insert wait-states.
  - For FPGA Master/PCI9054 Target transfers the signal is deasserted by the PCI9054 to insert wait-states.
  - For PCI9054 master transactions, the signal must be enabled as a PCI9054 input via the registers `LBRD0[22,6]`, `LBRD1[6]`, `DMAMODE0[6]`, and `DMAMODE1[6]` (p11-24, p11-28, p11-34, p11-36 [3]).
- `LWAIT#`
  - This signal is the master wait-state control.
  - For PCI9054 Master/FPGA Target transfers the signal is asserted by the PCI9054 to insert wait-states.
  - For FPGA Master/PCI9054 Target transfers the signal is asserted by the FPGA to insert wait-states.
  - For PCI9054 master transactions, the signal remaines deasserted unless wait-states are programmed via the registers `LBRD0[21:18,5:2]`, `LBRD1[5:2]`, `DMAMODE0[5:2]`, and `DMAMODE1[5:2]` (p11-24, p11-28, p11-34, p11-36 [3]). The wait-states are a fixed number of clock cycles (1 to 15), and are inserted between the address and first data phase, and between data phases.
- `BTERM#`
  - Target devices can assert this signal to request the generation of another address cycle. This feature can be used by burst-capable target devices to trigger a new address cycle when a burst traverses over a decode region (bursting from one device region to another), or by non-burst-capable devices to break a burst into single transactions.
  - For PCI9054 master transactions, the signal must be enabled as a PCI9054 input via the registers `LBRD0[23,7]`, `LBRD1[7]`, `DMAMODE0[7]`, and `DMAMODE1[7]` (p11-24, p11-28, p11-34, p11-36 [3]).
- Continuous burst mode
  - For PCI9054 master transactions, continuous bursting (see Section 4.2.5 p4-3 [3]) is enabled by enabling the `BTERM#` input and via the registers `LBRD0[26,24]`, `LBRD1[8]`, `DMAMODE0[8]`, and `DMAMODE1[8]` (p11-24, p11-28, p11-34, p11-36 [3]).

Section 4 provides the register settings used during hardware testing of the example designs.

2.6 PCI9054 Master/FPGA Target transactions

This section contains local bus timing diagrams for the PCI9054 operating as the local bus master and the FPGA as the local bus target;

- Figure 2 shows read/write single/burst transactions.
- Figure 3 shows read/write single/burst transactions with master wait-states (`plx_waitN` assertion).
- Figure 4 shows read/write burst transactions with target termination (`plx_termN` assertion).

Notes on the transaction sequences are;

- The PCI9054 acquires the local bus (asserts `plx_hold` and receives `plx_holda`) before driving any bus signals. The PCI9054 tristates all signals before releasing the bus.
- The PCI9054 starts a transaction by generating an address strobe (`plx_adsN` driven low for one clock), asserting the write/read indicator (`plx_wr_rdN`), and driving the transaction byte-enables (`plx_beN[3:0]`) and address (`plx_ad[31:0]`).
- If the transaction is a write, the first write-data phase is driven onto the bus on the clock after the address strobe.
- Data transfers occur in data phases where `plx_waitN` is deasserted and `plx_rdyN` is asserted. The final data phase (transaction end) occurs when both `plx_lastN` and `plx_rdyN` are asserted (`plx_waitN` never asserts when `plx_lastN` asserts).
- If the transaction is a single read/write, and there are no wait-states programmed, the last indicator (`plx_lastN`) is asserted from the clock after the address strobe until the end of the transaction.

  If the transaction is a burst read/write, the last indicator is driven during the last data phase (which can last for multiple clocks, if `plx_rdyN` is not asserted when `plx_lastN` is first asserted).
- There is a minimum of one clock turn-around between back-to-back transfers by the PCI9054 (Section 4.2.6 p5-4 [3]).

2.7 PCI9054 Target/FPGA Master transactions

An FPGA local bus master interface provides access to the PCI9054 internal configuration registers and access to the PCI bus. Because the PCI9054 contains a DMA controller that can be used to provide high-performance transfers over the PCI bus, burst-capability is not really required for the FPGA local bus master interface. Providing a local bus master with read/write single transactions support is sufficient for most applications.

The FPGA local bus master transactions are virtually identical to the PCI9054 local bus master transactions, with the exceptions;

- The FPGA bus master arbitration signals are internal to the FPGA.
- Accesses to the PCI9054 internal registers are initiated with the assertion of both `plx_ccsN` and `plx_adsN` (p5-36 [3])
- Accesses to the PCI bus are initiated with the assertion of only `plx_adsN`.

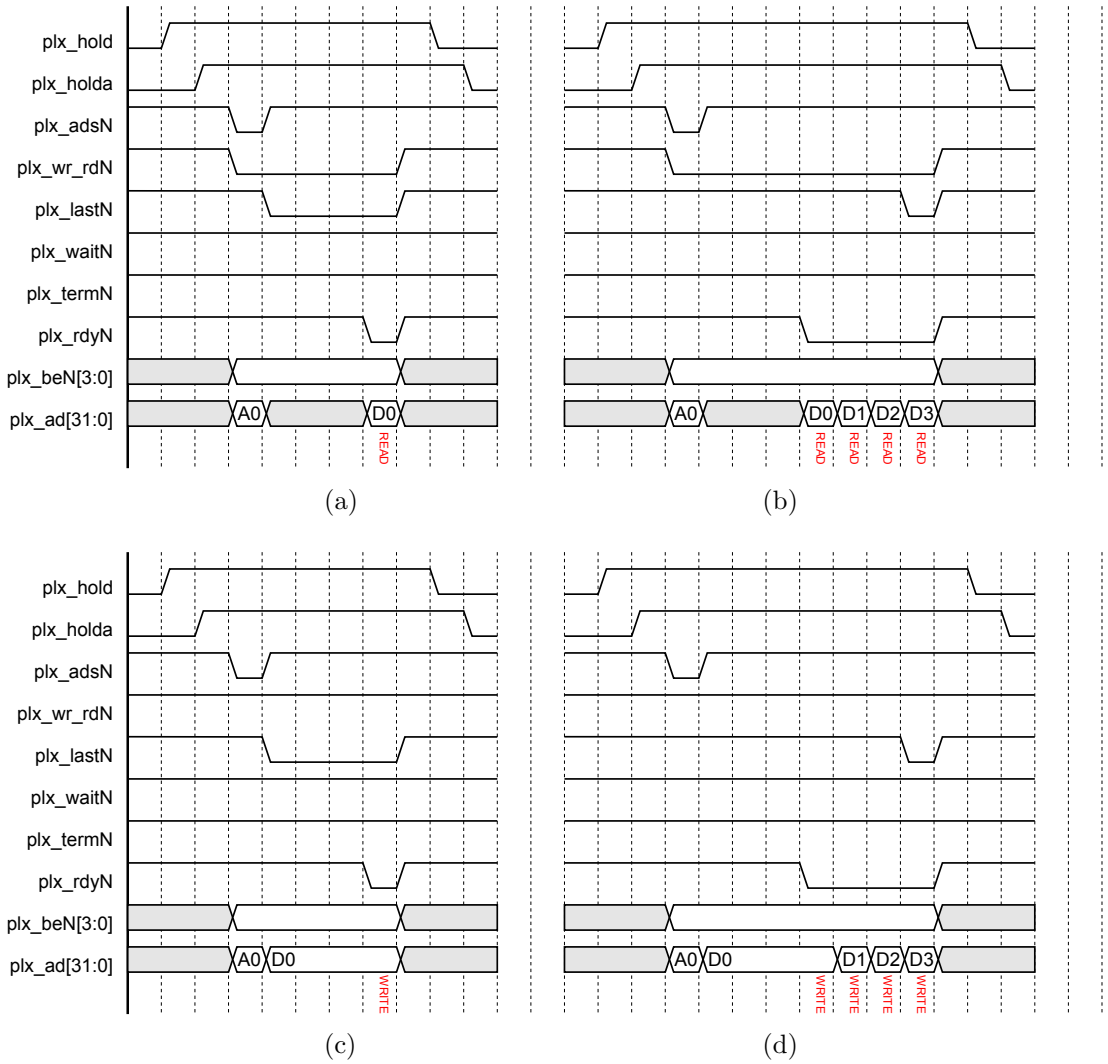


Figure 2: PLX PCI9054 local bus target timing; (a) read-single, (b) read-burst, (c) write-single, and (d) write-burst.

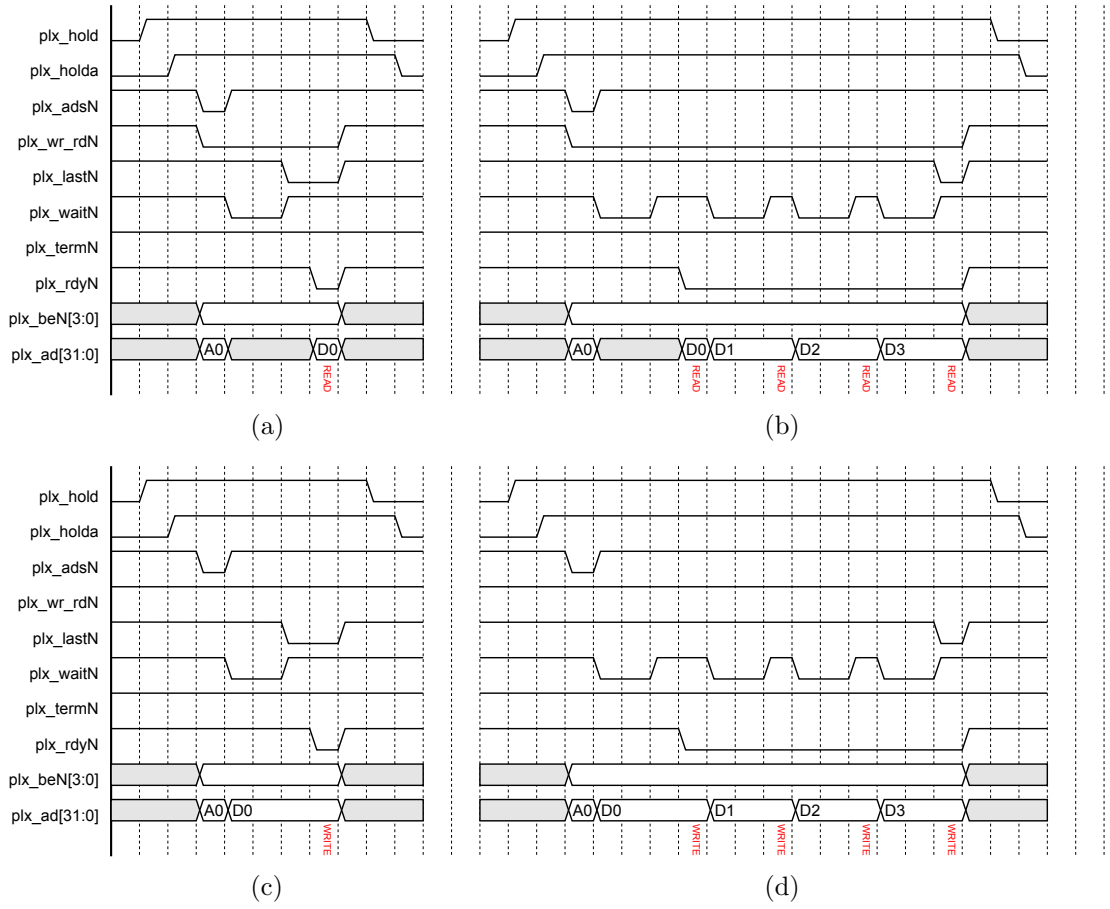
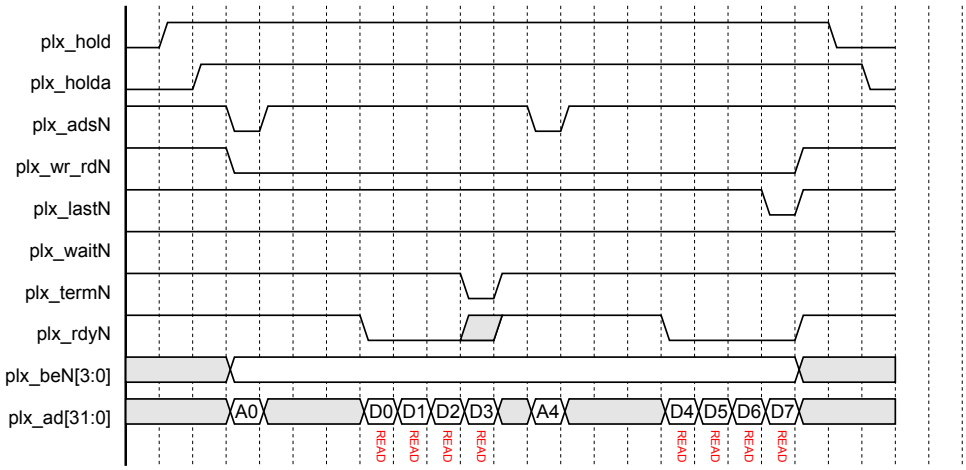
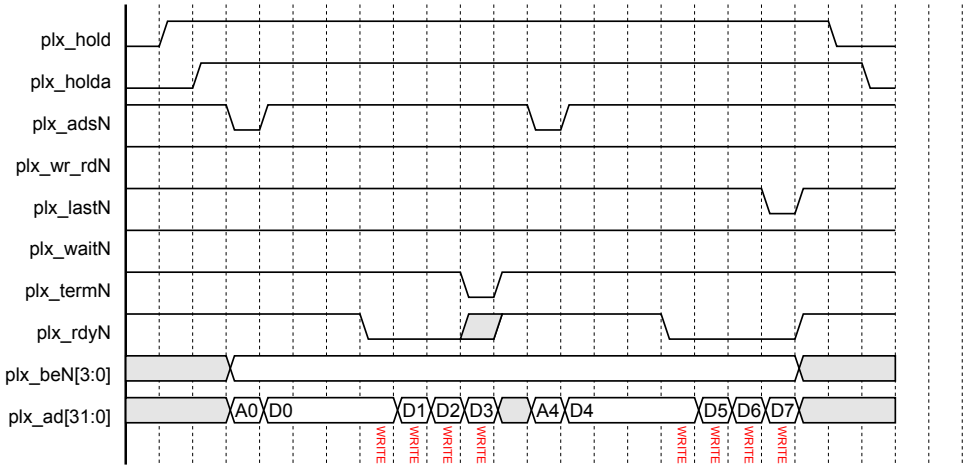


Figure 3: PLX PCI9054 local bus target timing with two wait-states; (a) read-single, (b) read-burst, (c) write-single, and (d) write-burst.

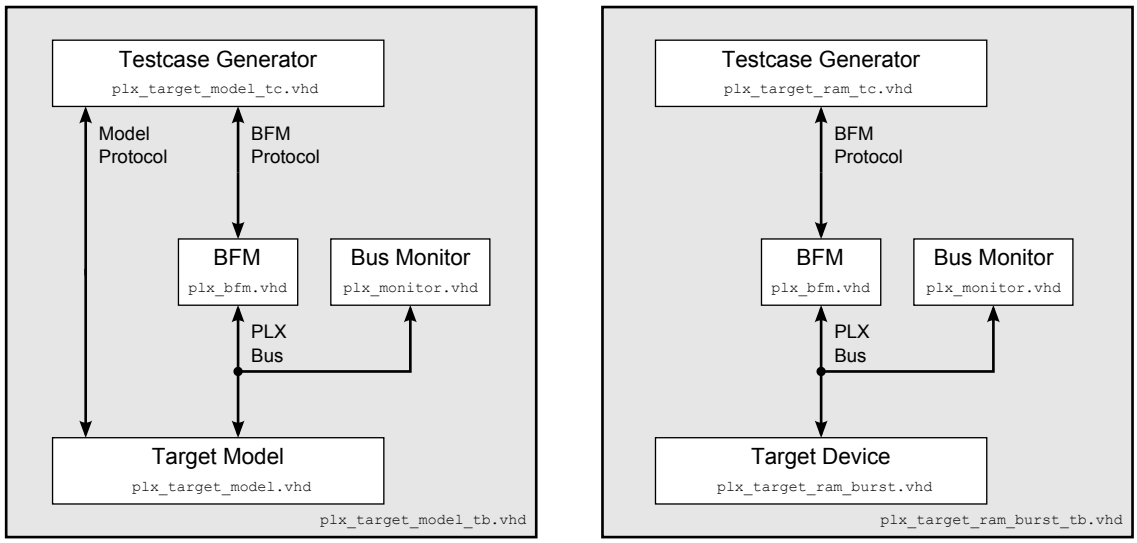


(a)



(b)

Figure 4: PLX PCI9054 local bus target burst terminate timing; (a) read-burst, and (b) write-burst.



(a) (b)

Figure 5: PLX PCI9054 local bus simulation; (a) simulation components testbench, and (b) synthesizeable component testbench.

3 Interface Simulation

A critical part of the bus interface design process, is to ensure that components meet the requirements of the bus protocol. Figure 5 shows a block diagram of how the PLX bus components developed in this document were tested. The PLX bus functional model (BFM) is a component that generates PLX PCI9054 local bus transactions, i.e., the bus signal waveforms that correspond to read-single, read-burst, write-single, and write-burst transactions, transactions with wait-states, and burst-terminated transactions. The PLX bus monitor observes the signals on the PLX bus and ensures that each transaction adheres to the PLX bus protocol; any violation results in simulation termination. A user-defined testcase generator consists of transaction sequences, i.e., reads and writes. The testcase generator communicates with the BFM using a procedural interface, eg. reads and writes are initiated with functions like `read_single()`, and `write_burst()`.

Figure 5(a) shows the testbench for the simulation components. A PLX target model was developed to test the BFM and bus monitor. The testcase generator communicates with the target model using a procedural interface, much the same as used to communicate with the BFM. The testcase first communicates to the model the transaction (or sequence of transactions) about to be initiated on the PLX bus. The transactions are then initiated with the BFM. The target model then uses the previously communicated transaction information to check the transaction. For reads, the model is passed the read data, which it returns to the testcase generator via the read on the PLX bus, and the testcase generator checks the read data matches what was expected. For writes, the model checks the write data obtained from the PLX bus against what was expected. Burst transactions are compared against their expected lengths. The BFM can insert master wait-states (which are fixed values, eg. two wait-states between each transaction), whereas the target model can insert an arbitrary number of wait-states, eg. deassertions of `plx_rdyN` between data phases. The target model communications protocol provides a mechanism for passing an array of wait-state data-phase pairs. The first wait-state entry is the address-to-first data phase latency, followed by the number of valid data phases to allow before inserting target wait-states. The target model testbench

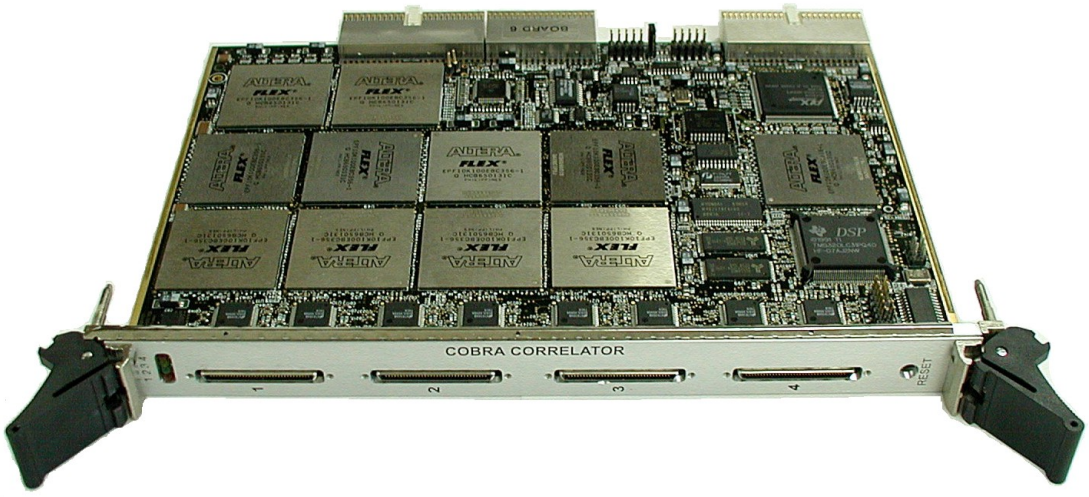


Figure 6: COBRA Correlator Board; PLX PCI9054 PCI interface, Altera FLEX10KE system control FPGA, Texas Instruments TMS320C30 DSP, and 10 FLEX10KE data processing FPGAs.

testcase generator uses a random number generator to test various master and target wait-state combinations, thus ensuring the correctness of the simulation components.

Figure 5(b) shows the testbench for the synthesizeable burst-transaction RAM component (which is described in the next section). Since this component has fixed properties, there is no target model communications path shown (though the testbench for a synthesizeable component with external I/O could have such a path). The testbench for the RAM generates read and write, single and burst transactions. All components designed in the next section are verified using a testbench based on the test components shown in Figure 5.

4 Interface Examples

Figure 6 shows a compact PCI (cPCI) board design that incorporates the PLX PCI9054 PCI to local bus bridge[1]. The interface examples in this section implement PCI9054 local bus target and local bus master/target interfaces of increasing complexity, with timing analysis based on the use of the COBRA board FLEX10KE system controller FPGA. The COBRA board local bus operates at 33MHz (30ns clock period), however, the FPGA timing constraints are set to achieve 50MHz operation.

Altera FLEX10KE place-and-route (P&R) and post-P&R simulation[2] requires the use of Quartus II 9.0SP2 and Modelsim-Altera Edition 6.4a (download from the Quartus archive). Support for the FLEX10KE FPGAs is not available in newer versions of the tools. The VHDL developed in this document can be simulated in newer versions of ModelSim, and can target newer devices, eg. Stratix or Cyclone series devices.

[1]The PCI9054 was used in the board design, rather than using a PCI core in the Altera FLEX10KE system controller FPGA, as the FPGA I/O were not PCI compliant and the PCI9054 was cheaper than the FPGA resources required to implement a PCI bridge

[2]Building output netlists, `.vho`, and using SDF timing netlists `.sdo` files

```
# lspci -s 01:0d.0 -v
01:0d.0 Bridge: PLX Technology, Inc. PCI9054 32-bit 33MHz PCI <-> IOBus Bridge (rev 0a)
        Subsystem: PLX Technology, Inc. PCI9054 32-bit 33MHz PCI <-> IOBus Bridge
        Flags: bus master, medium devsel, latency 64, IRQ 169
        Memory at febff800 (32-bit, non-prefetchable) [size=256]
        I/O ports at e800 [size=256]
        Memory at fea00000 (32-bit, non-prefetchable) [size=1M]
        Memory at fe900000 (32-bit, non-prefetchable) [size=1M]
        Expansion ROM at febe0000 [disabled] [size=64K]
        Capabilities: [40] Power Management version 1
        Capabilities: [48] #06 [0080]
        Capabilities: [4c] Vital Product Data
```

Figure 7: PLX PCI9054 PCI configuration after power-up with a blank configuration EEPROM.

4.1 PCI9054 configuration

The operation of the PLX PCI9054 is dependent on its configuration register settings, which can be loaded from an EEPROM at power-on. To ensure that the results of this document can be reproduced on other hardware incorporating a PCI9054, the COBRA board PCI9054 configuration EEPROM was erased, so that the board powers-on with default register settings as given in Chapter 11 of the device data book [3].

Figure 7 shows the PCI configuration space settings (as reported by the Linux tool `lspci`). The nominal PCI configuration space is four base address regions (BARs) and an expansion ROM region, i.e.,

- BAR0: 256-bytes 32-bit non-prefetchable memory (PCI9054 registers)
- BAR1: 256-bytes I/O ports (PCI9054 registers)
- BAR2: 1MB 32-bit non-prefetchable memory (local address space 0)
- BAR3: 1MB 32-bit non-prefetchable memory (local address space 1)
- Expansion ROM 64kB

The 256-bytes of PCI9054 configuration registers are accessible as memory or I/O ports via the first two BARs. Figure 8(a) shows the power-on defaults for these registers (read from BAR0).

The hardware tests in this document use the 1MB BAR3 region which is referred to in the PCI9054 data book as Local Address Space 0 (LAS0). The configuration of LAS0 is controlled by the registers highlighted in blue in Figure 8(c);

- `LAS0RR = FFF00000` configures the device as a 1MB non-prefetchable memory region.
- `LAS0BA = 00000000` disables access to the region. The region must be enabled after power-on by writing 1 to this register (see Section 5).
- `LBRD0 = 40430043h`; configures LAS0 as 32-bits, with the ready (`plx_rdyN`) input enabled, the termination (`plx_termN`) input disabled, bursting disabled, and zero wait-states (so `plx_waitN` will not assert).

Figure 9 shows the configuration EEPROM settings used on a COBRA board, while Figure 8(b) shows the BAR0 power-on register values after configuring from the EEPROM. Comparison of the register values in Figures 8(a) and 8(b) shows how the EEPROM changes the power-on values.

```
00: FFF00000 00000000 00200000 00300500
10: FFFF0000 00000000 40430043 00000000
20: 00000000 00000000 00000000 00000000
30: 00000000 00000008 00000000 00000000
40: 00000000 00000000 00000000 00000000
50: 00000000 00000000 00000000 00000000
60: 00000000 00000000 0F010180 080F767E
70: 905410B5 0000000A 00000000 00000000
80: 00000043 00000000 00000000 00000000
90: 00000000 00000043 00000000 00000000
A0: 00000000 00000000 00001010 00200000
B0: 00000000 00000000 00000000 00000000
C0: 00000002 00000000 00000000 00000000
D0: 00000000 00000000 00000000 00000000
E0: 00000000 00000000 00000050 00000000
F0: FFF00000 00000000 00000043 00000000
```

(a)

```
00: FF800008 00000001 10200000 00300600
10: 00000000 00000000 4B4300C3 00000000
20: 00000000 00000000 00000000 00000000
30: 00000000 00000008 00000000 00000000
40: 00000000 00000000 00000000 00000000
50: 00000000 00000000 00000000 00000000
60: 00000000 00000000 0F010100 180F767E
70: 905410B5 0000000A 00000000 00000000
80: 00000043 00000000 00000000 00000000
90: 00000000 00000043 00000000 00000000
A0: 00000000 00000000 00001010 10200000
B0: 00000000 00000000 00000000 00000000
C0: 00000002 00000000 00000000 00000000
D0: 00000000 00000000 00000000 00000000
E0: 00000000 00000000 00000050 00000000
F0: 00000000 00000000 00000000 00000000
```

(b)

00:	LAS0RR	LAS0BA	MARBR	PROT_AREA
10:	EROMRR	EROMBA	LBRD0	DMRR
20:	DMLBAM	DMLBAI	DMPBAM	DMCFGA
30:	OPQIS	OPQIM		
40:	MBOX0	MBOX1	MBOX2	MBOX3
50:	MBOX4	MBOX5	MBOX6	MBOX7
60:	P2LDBELL	L2PDBELL	INTCSR	CNTRL
70:	PCIHIDR	PCIHREV	LAS0RR	LAS0RR
80:	DMAMODE0	DMAPADR0	DMALADR0	DMASIZ0
90:	DMADPR0	DMAMODE1	DMAPADR1	DMALADR1
A0:	DMASIZ1	DMADPR1	DMACSR	DMAARB
B0:	DMATHR	DMADAC0	DMADAC1	
C0:	MQCR	QBAR	IFHPR	IFTPR
D0:	IPHPR	IPTPR	OFHPR	OFTPR
E0:	OPHPR	OPTPR	QSR	
F0:	LAS1RR	LAS1BA	LBRD1	DMDAC

(c)

Figure 8: PLX PCI9054 registers; (a) register values after power-up with a blank configuration EEPROM, (b) register values after power-up with the configuration EEPROM settings in Figure 9, and (c) register names. The register names for Local Address Space 0 (LAS0) are highlighted in blue, and the register names for DMA channel 0 are highlighted in red.

00:	905410B5	PCIIDR
04:	0680000A	PCICCR
08:	00000100	PCIMLR
0C:	00000000	MBOX0
10:	00000000	MBOX1
14:	FF800008	LAS0RR
18:	00000001	LAS0BR
1C:	10200000	MARBR
20:	00300600	PROT_AREA
24:	00000000	EROMRR
28:	00000000	EROMBA
2C:	4B4300C3	LBRD0
30:	00000000	DMRR
34:	00000000	DMLBAM
38:	00000000	DMLBAI
3C:	00000000	DMPBAM
40:	00000000	DMCFGA
44:	905410B5	PCISID
48:	00000000	LAS1RR
4C:	00000000	LAS1BA
50:	00000000	LBRD1
54:	00004C06	HS_NEXT

Figure 9: COBRA board PLX PCI9054 configuration EEPROM settings; Local Address Space 0 is configured as an 8MB prefetchable PCI region and burst transfers are enabled on the local bus.

4.2 Target-only single-transaction register interface

The simplest PCI9054 local bus target is a single 32-bit register that supports single-access read/write transactions, i.e., no support for bursts (`plx_lastN`) or master wait-states (`plx_waitN`). Figure 10 shows a block diagram for the design implemented in `plx_target_register.vhd`[3], while Figure 11 shows the read/write timing for the design.

Figure 11(a) shows a single clock bus turn-around phase between the address and read-data. This turn-around phase is required, since in J-mode, the `plx_ad` bus is driven by the PCI9054 during the address phase, and then driven by the FPGA during the read-data phase. The timing parameters in Table 3 show that the PCI9054 clock-to-output delay for the `plx_ad` bus is between 5.0ns and 11.0ns. Timing analysis of the FPGA design indicates a clock-to-output delay of between 6.0 and 9.0ns. If the bus turn-around cycle was not present, then the FPGA would start driving read data onto the `plx_ad` bus after 6.0ns (best-case), whereas the PCI9054 would still be driving the address until 11.0ns (worst-case), thus there would be a driver conflict on the bus for 5.0ns every read cycle.

The `plx_target_register.vhd` design shows how a basic PLX target interface can be implemented, but more importantly, it provides a hardware implementation that can be tested to confirm the bus signal operations. For example, Figure 12 shows the read and write waveforms captured from a COBRA board configured as described in Section 4.1, i.e., default register settings, with access to LAS0 enabled. In Figure 12, note how during reads the write/read control, `plx_wr_rdN`, is driven low for one extra clock after the end of the transaction relative to the timing shown in Figures 2 and 11. A review of the PLX PCI9054 data book [3] shows many of the timing diagrams have ambiguous timing for the write/read control and the byte-enables. The component designs in this document do not care what the state of these signals is during the clock phases after a transaction has completed, so the waveforms in Figure 2 were retained for use in verification simulations. The testbench `plx_target_register_tb.vhd` contains test-cases for various combinations of single-transaction reads and writes.

The hardware setup used to capture the timing waveforms in Figure 12 was also used to investigate the clock-to-output timing; the PCI9054 timing is given in Table 3, and the timing reported for the FPGA `plx_rdyN` output was $t_{\mathrm{CO(max)}} = 6.6$ns. The signals driven by the PCI9054 all had clock-to-output delays of around 5.5ns, and the FPGA clock-to-output for the ready signal was around 6.0ns; this can be seen in Figure 12 by noting how the edges of `plx_rdyN` signal align fairly well with the timing of the other control signals. The PCI9054 `plx_rdyN` setup/hold times from Table 3 are 9.5ns/1.0ns, so with an FPGA clock-to-output maximum of 6.6ns, the minimum local bus period with ideal clocks is 16.1ns (62MHz), so there is 3.9ns margin for a 50MHz (20ns) clock and 13.9ns margin for a 33MHz (30ns) clock.

[3]The block diagram in Figure 10 can be compared to the post place-and-route RTL view generated by Quartus, eg., see Tools→Netlist Viewers→RTL Viewer

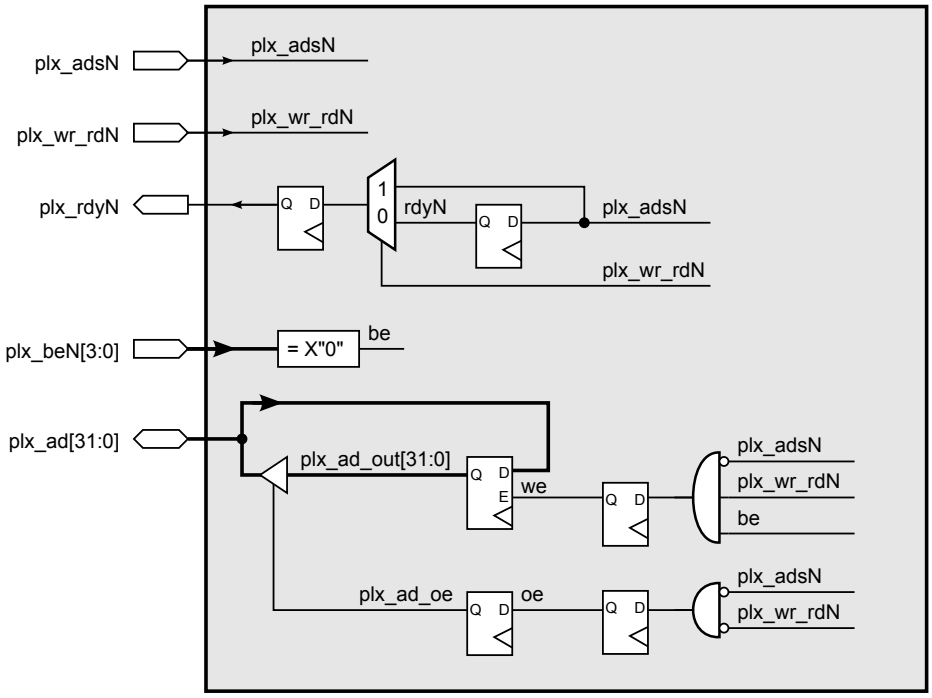


Figure 10: PLX PCI9054 local bus target register block diagram.

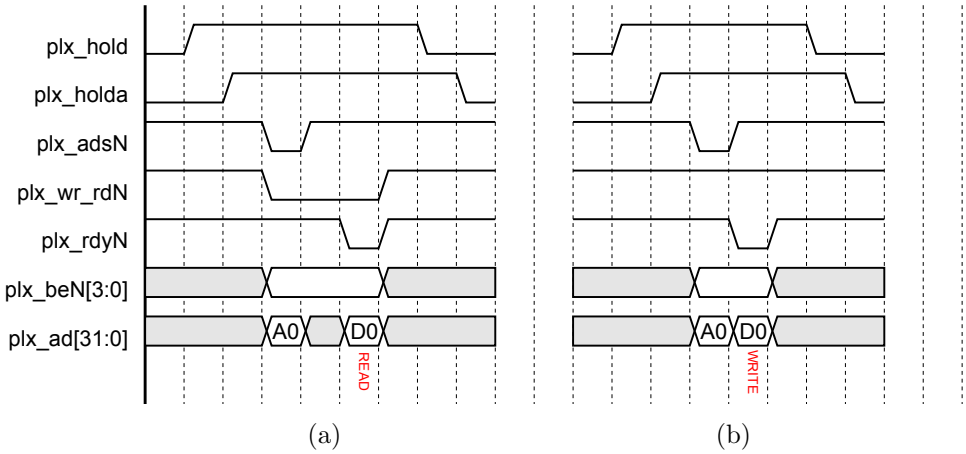
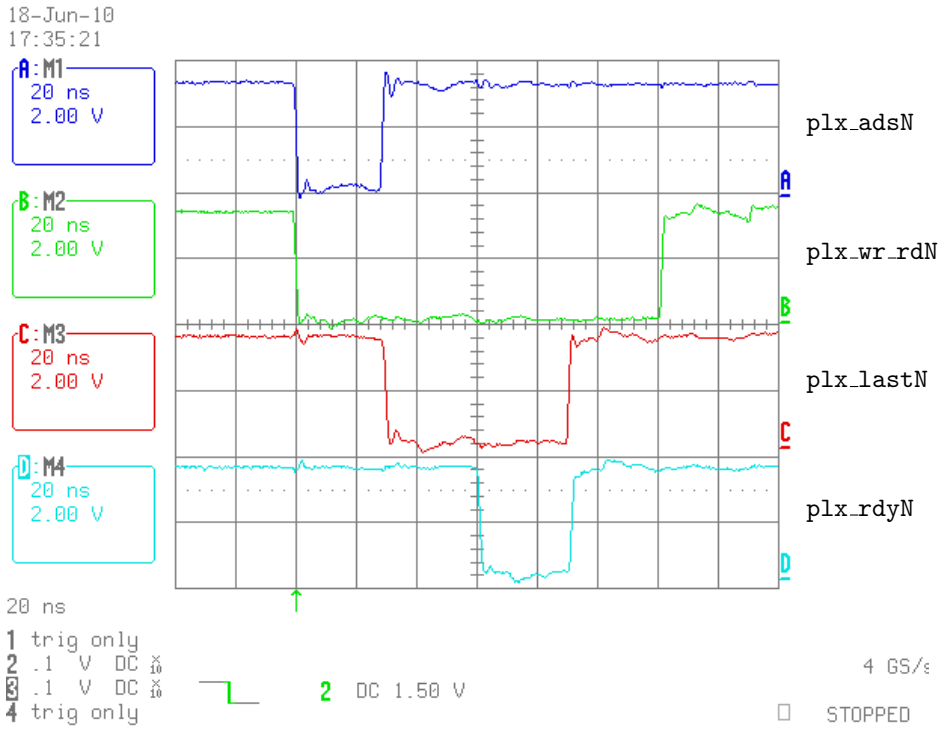
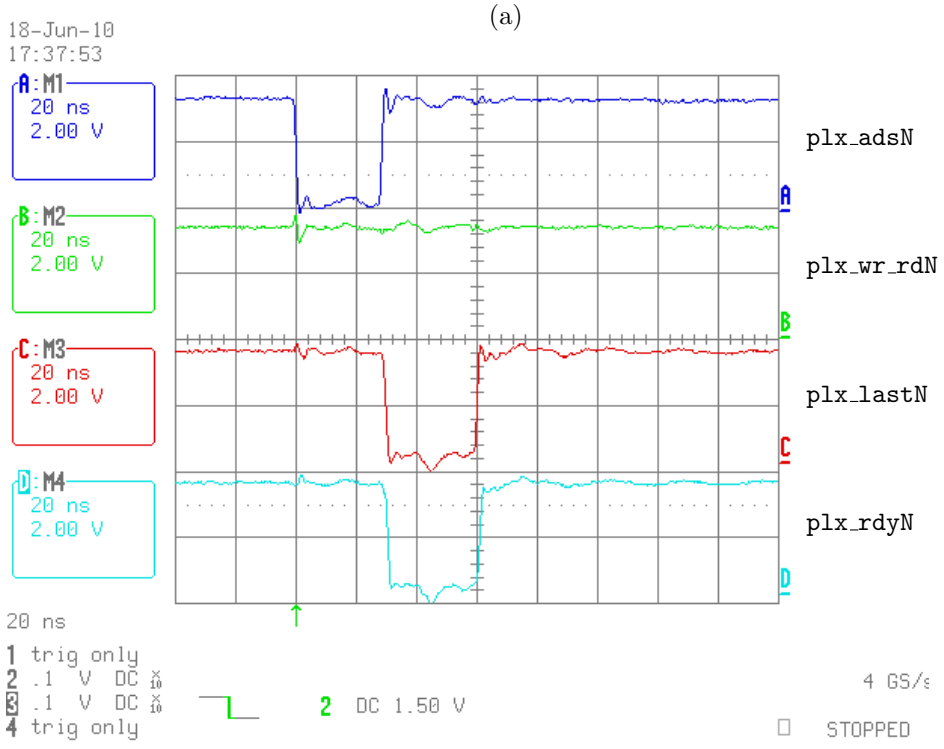


Figure 11: PLX PCI9054 local bus target register timing; (a) read-single, and (b) write-single.



(a)



(b)

Figure 12: PLX PCI9054 local bus target register access waveforms; (a) read-single, and (b) write-single.

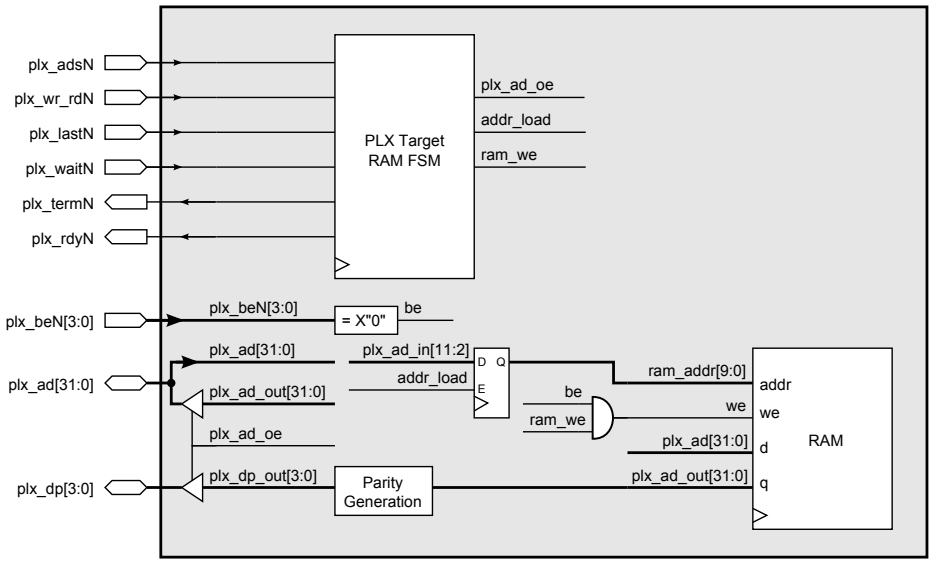


Figure 13: PLX PCI9054 local bus target single-transaction RAM interface block diagram.

4.3 Target-only single-transaction RAM interface

Figure 13 shows a block diagram for RAM target that supports single-access read/write transactions. This design is used to investigate burst-termination and master wait-state timing. The target converts burst-accesses (transactions with `plx_lastN` high) into single-access transactions by asserting the `plx_termN` signal. The target supports master wait-states by delaying the assertion of `px_rdyN` until the wait-states complete (`plx_waitN` deasserts). The design shown in Figure 13 is implemented in `plx_target_ram_single.vhd`, which contains the finite-state machine `plx_target_ram_single_fsm.vhd` shown by the algorithmic state-machine (ASM) chart in Figure 14. Figures 15 through 18 show read and write, single and burst transactions, with and without wait-states. The testbench `plx_target_ram_single_tb.vhd` reproduces all of these waveforms.

The design deliberately implements a *poor* target interface that will cause timing problems, and should *not* be copied for any real-world design—see the hardware measurements for further discussion on this.

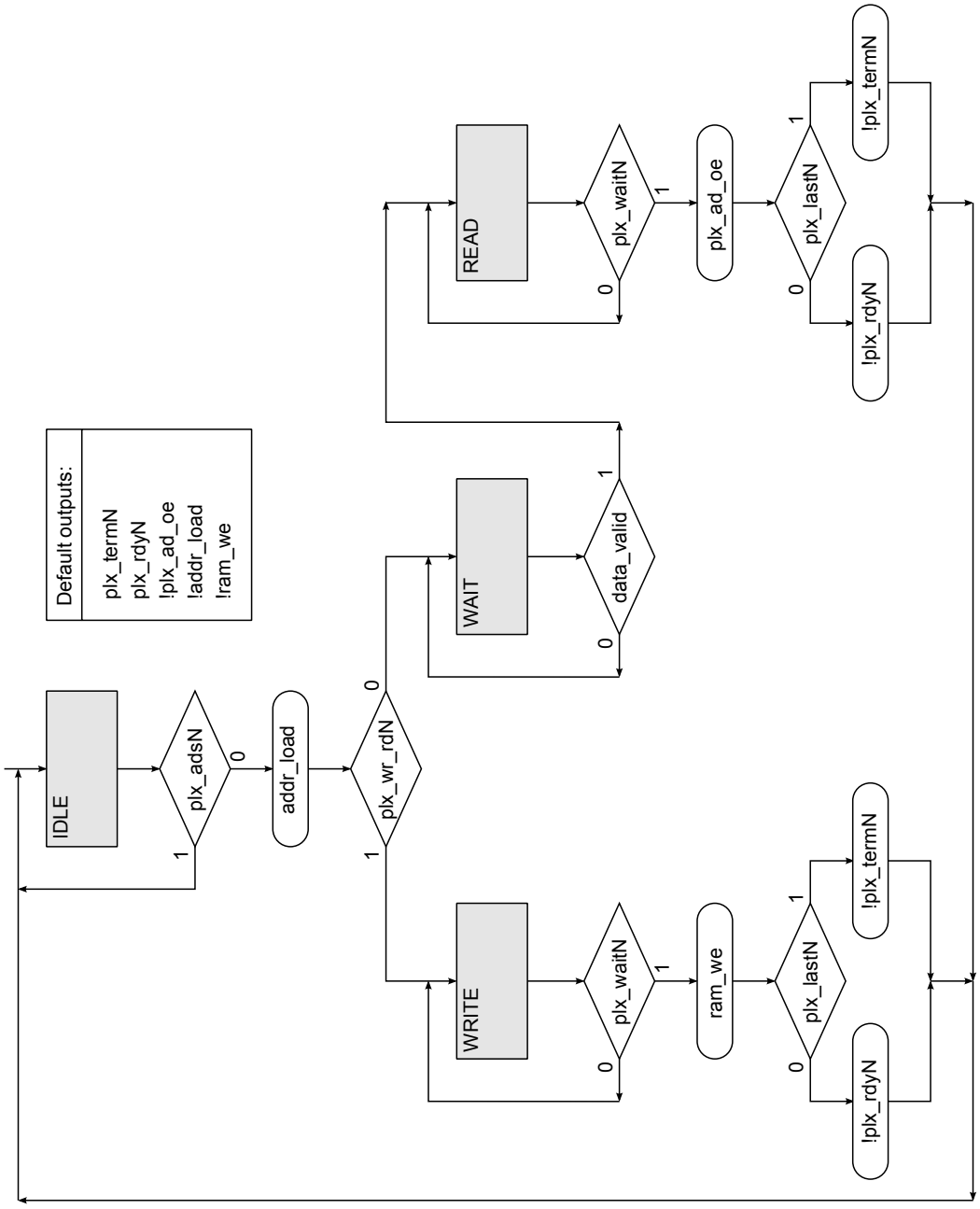


Figure 14: PLX PCI9054 local bus target single-transaction RAM interface finite-state machine (FSM).

Read transactions

With reference to the FSM in Figure 14, and the read timing diagrams in Figures 15 and 16;

- **Read-single**

  Figure 15(a) shows a read-single transaction. The assertion of `plx_adsN` and `plx_wr_rdN` low causes the FSM to transition to the WAIT state and to generate an address load pulse (`addr_load`). The FLEX10KE RAM has an address-to-data read latency of two clocks due to the presence of input and output registers. The address load pulse is pipelined two clocks to generate a pulse on `data_valid`. The FSM uses that pulse to transition to the READ state where read data is driven onto the `plx_ad` bus, and since the input `plx_lastN` is asserted, the output `plx_rdyN` is asserted low to complete the read transaction.

- **Read-burst**

  Figure 15(b) shows a read-burst transaction. The transaction proceeds identically to a read-single transaction up until the point where read data is driven onto the `plx_ad` bus. At that point, since the target input `plx_lastN` is high, the target output `plx_termN` is asserted low to terminate the burst. This causes the PCI9054 to generate another address phase and to continue the read-burst. Each read phase with `plx_lastN` high is terminated by the target. The last phase of a read-burst is identical to a read-single transaction. The repeated assertion of `plx_termN` converts the read-burst into a series of back-to-back read-single transactions.

- **Read-single with wait-states**

  Figure 16(a) shows a read-single transaction with three wait-states. The FSM transitions to the READ state once the read-data from the RAM is valid. However, since the master wait-state (target input) `plx_waitN` is asserted low, the target does not drive valid data onto the bus, and does not assert `plx_rdyN`. It is only once the wait-state signal deasserts that the read-data is driven onto the bus and the ready signal asserted. Three wait-states were chosen for this example, so that the FSM would remain in the READ state for an extra clock relative to the read-single transaction. The PCI9054 wait-state generator can be programmed for 0 to 15 clocks. Transactions with wait-states of 0, 1, or 2 will look identical to read-single transactions (due to the read-data latency), whereas wait-states of 3 to 15 will cause an increase in the read transaction time.

- **Read-burst with wait-states**

  Figure 16(b) shows a read-burst transaction with three wait-states between every read. The read-burst is converted to a series of back-to-back read-single with wait-states transactions.

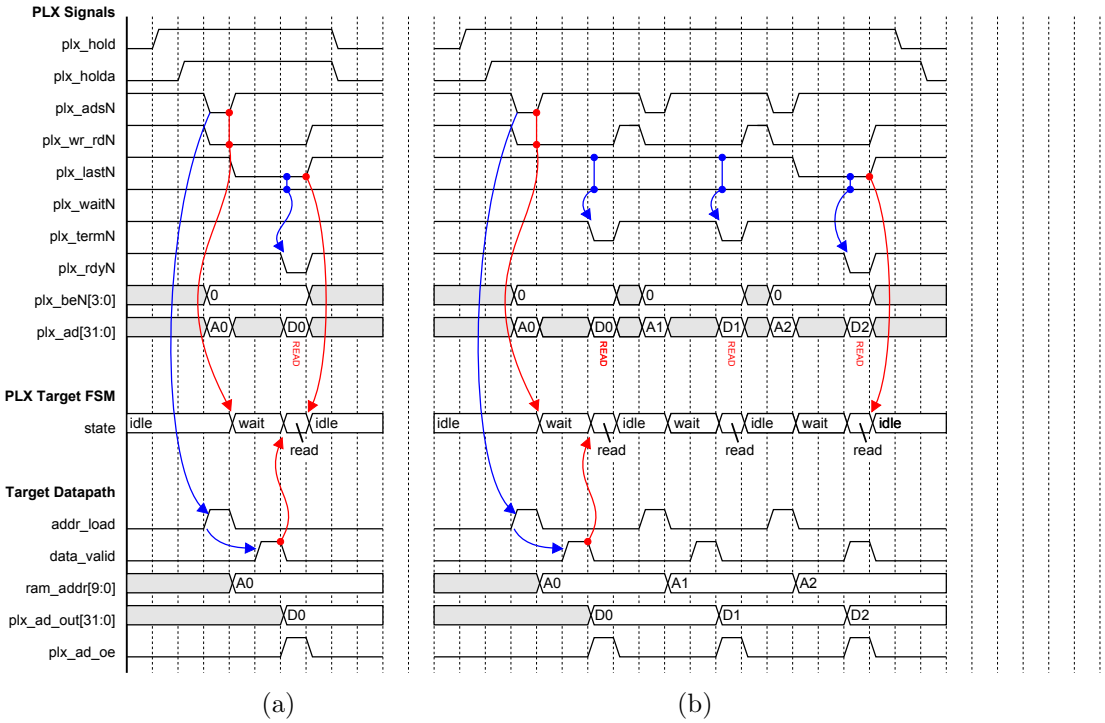


Figure 15: PLX PCI9054 local bus target single-transaction RAM read timing; (a) read-single, and (b) read-burst.

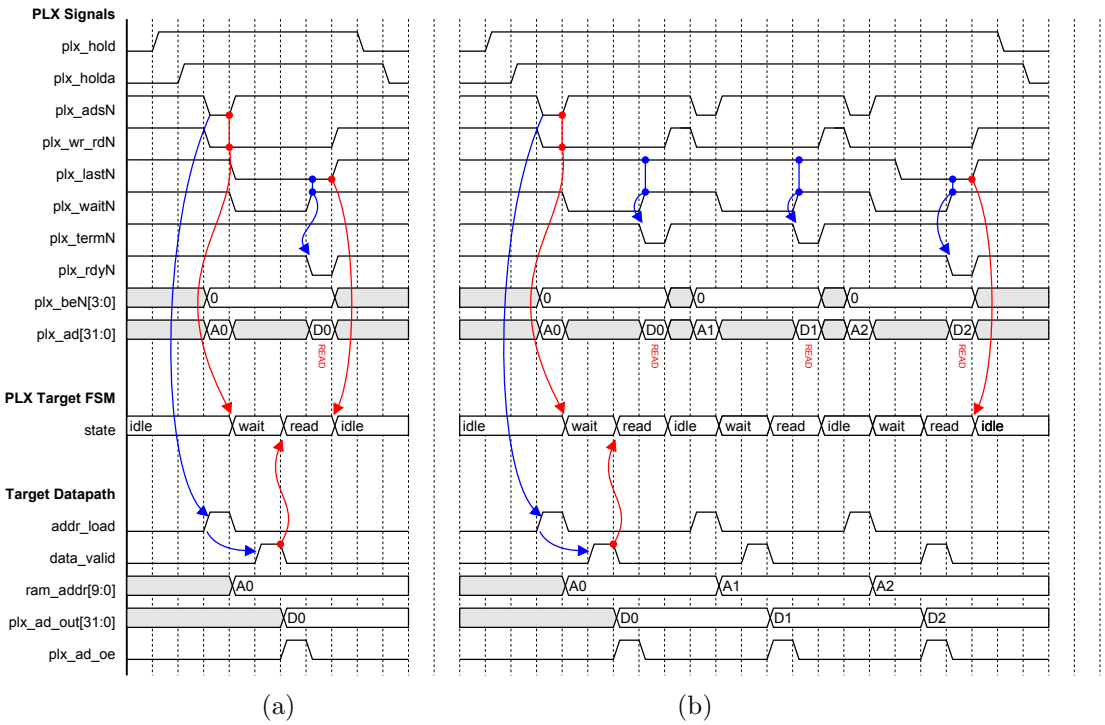


Figure 16: PLX PCI9054 local bus target single-transaction RAM read timing with three wait-states; (a) read-single, and (b) read-burst.

Write transactions

With reference to the FSM in Figure 14, and the write timing diagrams in Figures 17 and 18;

- **Write-single**

  Figure 15(a) shows a write-single transaction. The assertion of `plx_adsN` and `plx_wr_rdN` high causes the FSM to transition to the WRITE state and to generate an address load pulse (`addr_load`). In the WRITE state, the FSM generates a RAM write-enable (`ram_we`) pulse to write to the RAM. The FLEX10KE RAM does not have individual byte-enables, so Figure 13 shows that the byte-enables are checked to allow only 32-bit writes to the RAM. In the WRITE state, since the input `plx_lastN` is asserted, the output `plx_rdyN` is asserted low to complete the write transaction.

- **Write-burst**

  Figure 17(b) shows a write-burst transaction. As with the read-burst transaction, the `plx_termN` signal is asserted to break the burst into a series of back-to-back write-single transactions.

- **Write-single with wait-states**

  Figure 18(a) shows a read-single transaction with two wait-states. The FSM transitions to the WRITE state after the address strobe. However, since the master wait-state (target input) `plx_waitN` is asserted low, the target does not generate a RAM write-enable pulse, and does not assert `plx_rdyN`. It is only once the wait-state signal deasserts that the write-data is written to the RAM and the ready signal asserted.

- **Write-burst with wait-states**

  Figure 18(b) shows a write-burst transaction with two wait-states between every write. The write-burst is converted to a series of back-to-back write-single with wait-states transactions.

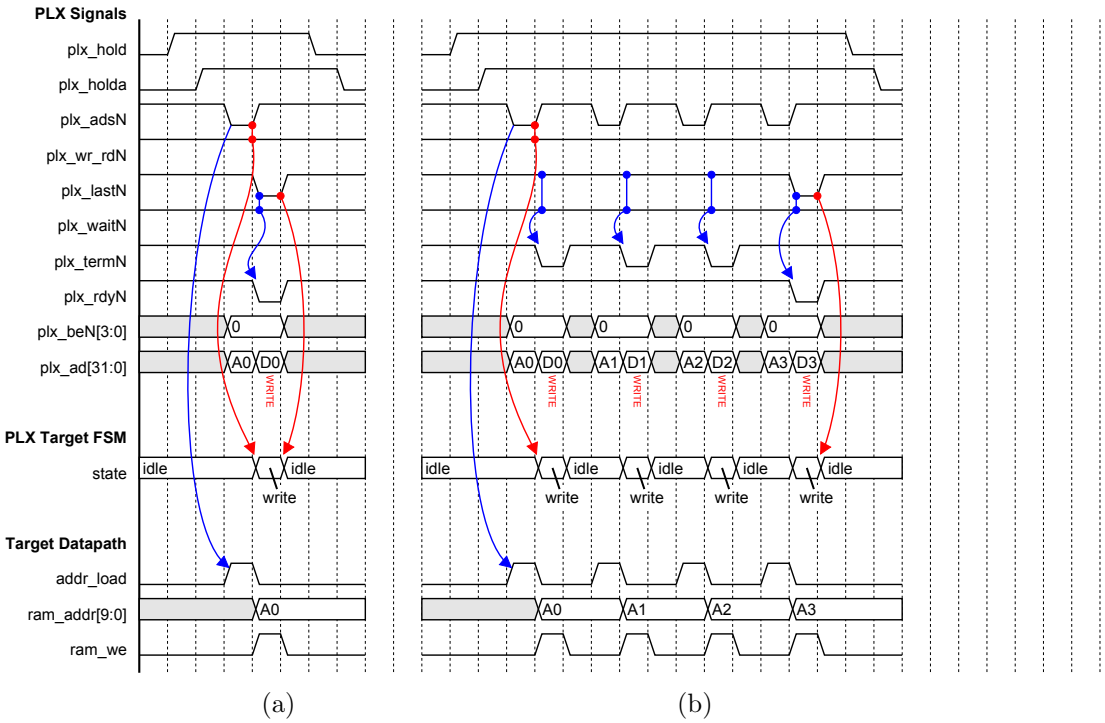


Figure 17: PLX PCI9054 local bus target single-transaction RAM write timing; (a) write-single, and (b) write-burst.

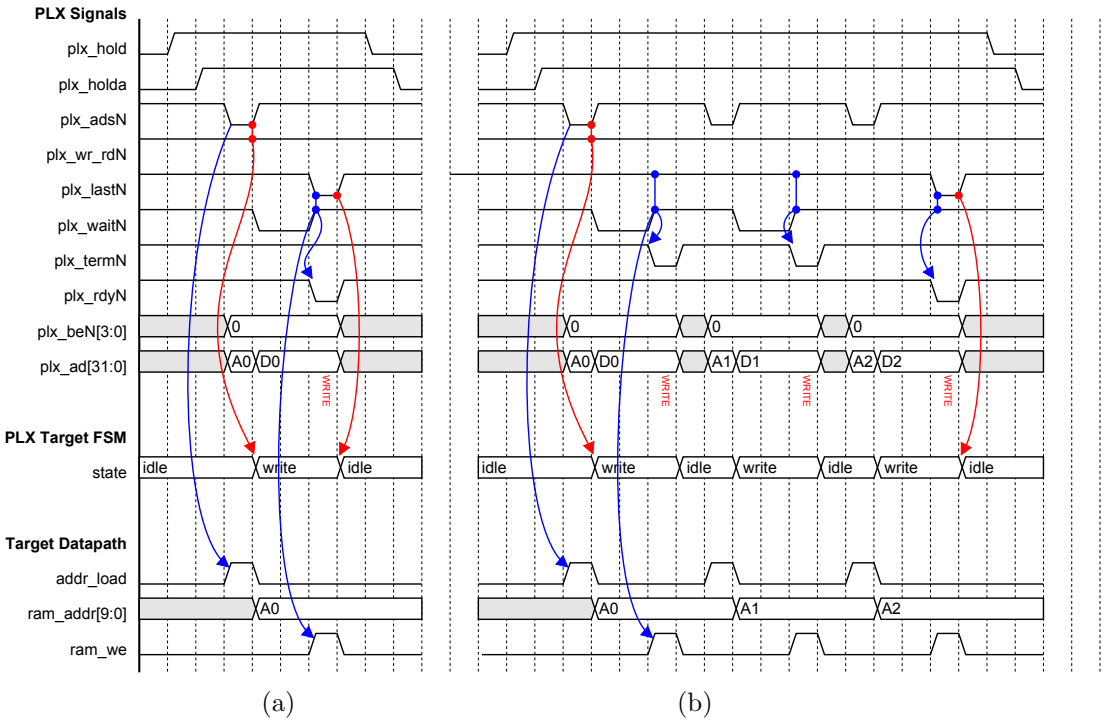


Figure 18: PLX PCI9054 local bus target single-transaction RAM write timing with two wait-states; (a) write-single, and (b) write-burst.

Hardware testing

Figure 19 show single-transaction waveforms for the target RAM interface, while Figure 20 shows the same transactions with two wait-states. Relative to the target register waveforms shown in Figure 12, the `plx_wr_rdN` signal has been replaced with `plx_waitN`, to show the effect of the wait-states. If you carefully compare Figure 19 and Figure 20 (by toggling between the figures in the electronic version of this document), you will notice the following;

- **Read wait-state timing**:
  - Compare Figures 19(a) and 20(a).
  - The transactions start (`plx_adsN` low pulse) and end (`plx_rdyN` low pulse) occur at around the same time for either transaction, i.e., two wait-states has no effect on the read timing, since the RAM data is not ready for read during that time.
  - With no wait-states `plx_lastN` is asserted from the deassertion of `plx_adsN` through to the assertion of `plx_rdyN`. With wait-states, `plx_lastN` does not assert during wait-states (when `plx_waitN` is asserted).
  - Careful comparison of the `plx_rdyN` signal timing between the two figures, shows that falling-edge timing occurs slightly later when wait-states are used, i.e., the effective clock-to-output *changes* depending on the inputs. This occurs due to the FSM generating a combinatorial ready output based on last and wait-state inputs. The Quartus Timing Analyzer report indicates this type of timing issue by reporting a propagation delay (t_{pd}) between the inputs `plx_lastN` and `plx_waitN` to the output `plx_rdyN` (paths are also reported for the `plx_termN` and `plx_ad` outputs). Any such reports should be interpreted as a warning that the design implementation needs to be improved.
  - Figure 20(b) shows the read timing with three wait-states. The read data is ready after two clocks, but the wait-states hold off the assertion of ready.
- **Write wait-state timing**:
  - Compare Figures 19(b), and 20(a) and (b).
  - The two and three wait-state examples in Figure 20 result in identical timing for both reads and writes. The wait-states delay the assertion of `plx_lastN` by the master, and the assertion of `plx_rdyN` by the target.
- **Read burst-termination timing**:
  - Compare Figures 19(a), 21(a), and Figure 22(a).
  - The burst transactions are 8-bytes, i.e., two 32-bit transactions.
  - The first data phase of the read burst in Figure 21(a) shows the two transactions in the burst (two assertions of address strobe). The first transaction starts with `plx_lastN` high, indicating a burst. This transaction is terminated by the target via the assertion of `plx_termN`. The second transaction starts with `plx_lastN` asserted, indicating the final phase of the burst (or a single transaction), and this transaction ends with the assertion of `plx_rdyN`.
  - The read-burst setup used in Figure 21(a) was used to examine the timing of `plx_wr_rdN` (not shown in the figures); the signal was observed to stay low for the duration of the two burst transaction and then tri-state (the rising-edge was not driven high, but had an RC-risetime characteristic). The signal remains asserted for longer bursts. This assertion of `plx_wr_rdN` is consistent with the extra clock low seen Figure 12(a) at the end of the transaction.

  - The timing in Figure 21(a) indicates that the target *can not* drive the `plx_ad` bus after the end of a transaction, as within one clock, the PCI9054 could drive a new address.

- **Write burst-termination timing**:
  - Compare Figures 19(b), 21(b), and Figure 22(b).
  - The burst transactions are 8-bytes, i.e., two 32-bit transactions.
  - The first data phase of the write burst in Figure 21(b) shows the two transactions in the burst (two assertions of address strobe). The first transaction starts with `plx_lastN` high, indicating a burst. This transaction is terminated by the target via the assertion of `plx_termN`. The second transaction starts with `plx_lastN` asserted, indicating the final phase of the burst (or a single transaction), and this transaction ends with the assertion of `plx_rdyN`.
  - Figure 21(b) demonstrates another issue with the cominatorial use of `plx_lastN` and `plx_waitN` to generate the `plx_termN` output. The difference in propagation delays internal to the FPGA cause the glitch on `plx_termN` around the time `plx_rdyN` asserts. The testbench waveforms in Figure 22 show that the simulation reproduces this problem.

The variation in timing of the `plx_rdyN`, `plx_termN`, and the output data and parity (not shown) can be eliminated by using output registers on those signals. The design in the next section demonstrates this technique.

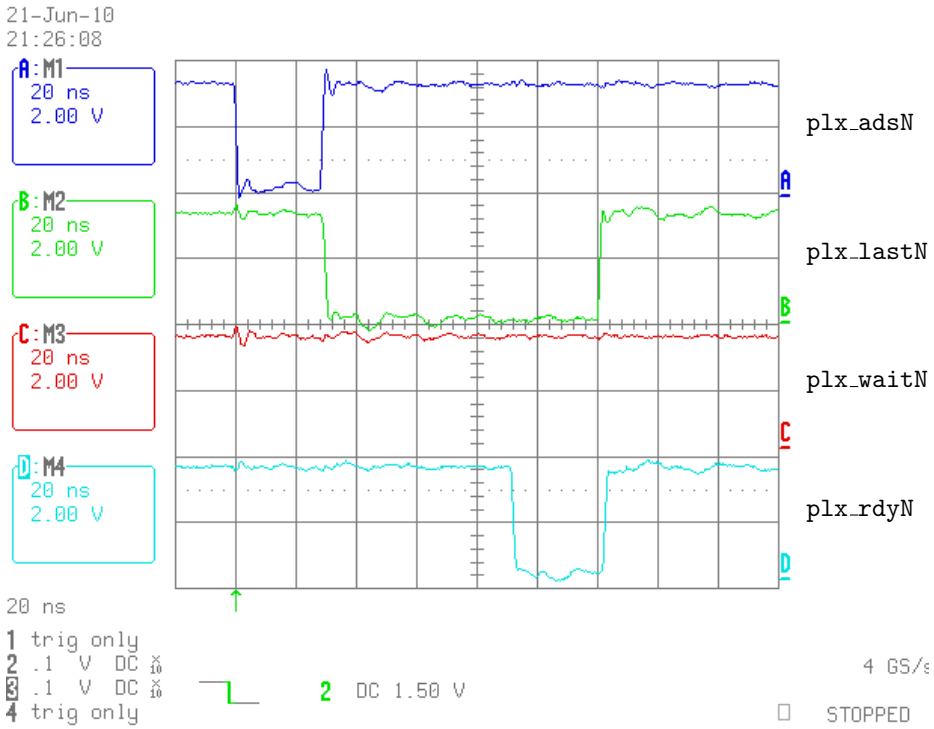
The single and burst transactions used to generate the figures in this and the following sections were setup as follows;

- The COBRA board was powered-up with a blank EEPROM.
- Local address space 0 was enabled, eg. using the PCI debug tool `c 4 1` (see Section 5).
- Read single; `d 0 4`
- Write single; `c 0 12345678`
- `LBRD0 = 40430043` (default) for no wait-states, `4043004B` for two wait-states, `4043004F` for three wait-states.
- Burst transactions were setup using a second COBRA board on the same PCI segment as a data source or destination, and using the DMA controller on the board being probed to move data between COBRA boards. The DMA controller programming sequence was

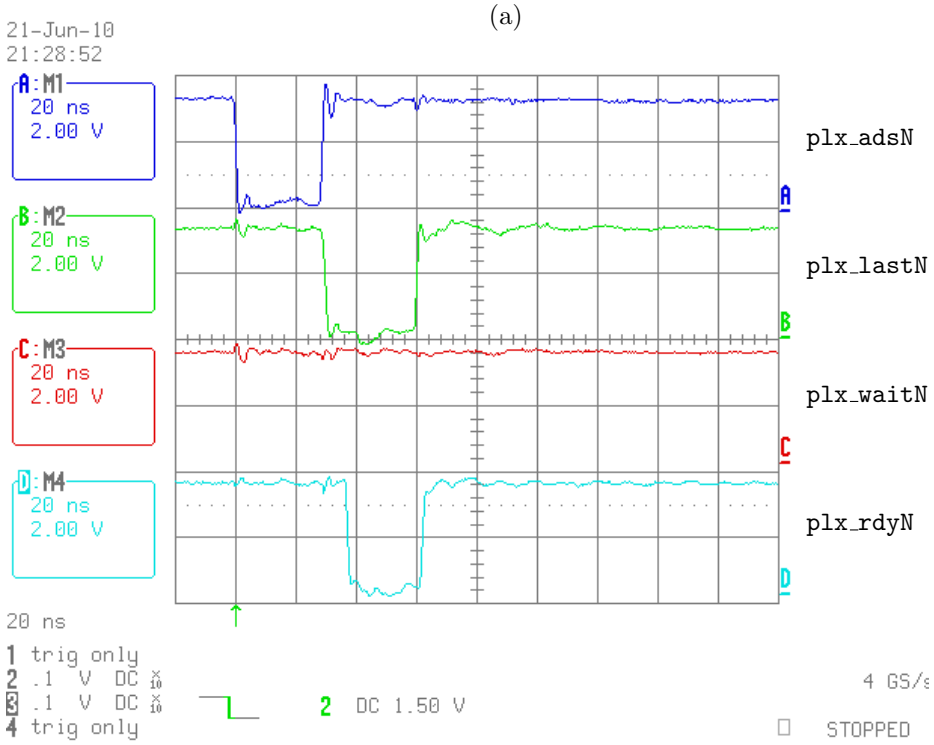
```
c 80 1C3       # DMAMODE0: Bursting enabled
c 84 FB000000  # DMAPADR0: PCI address (the other board)
c 88 0         # DMALADR0: Local bus address
c 8C 8         # DMASZI0:  Number of bytes to transfer
c 90 0         # DMADPR0:  0 = PCI->LBC write burst, or
               #           8 = LBC->PCI read burst
c A8 3         # DMACSR:   Perform the DMA
```

  where Figure 8(b) shows the DMA channel 0 registers highlighted in red. The setting for `DMADPR0` was changed to trigger write or read bursts. The `DMAMODE0` register could be set to `43` to generate two back-to-back single transactions (or `DMASIZ0` could be changed to `4` to generate the single transactions).

- Note that the local bus generates burst transfers even though the PCI space is marked as non-prefetchable (due to the blank EEPROM).
- Back-to-back local bus transactions can be triggered by performing non-32-bit accesses, eg. `d 2 4` or `c 2 11112222`.

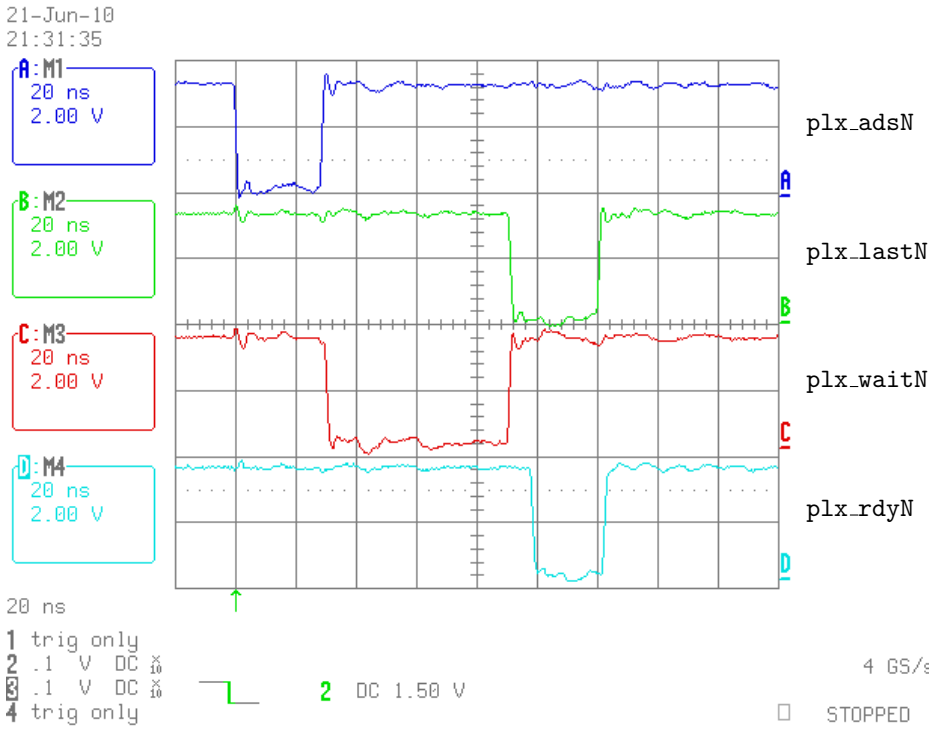


(a)

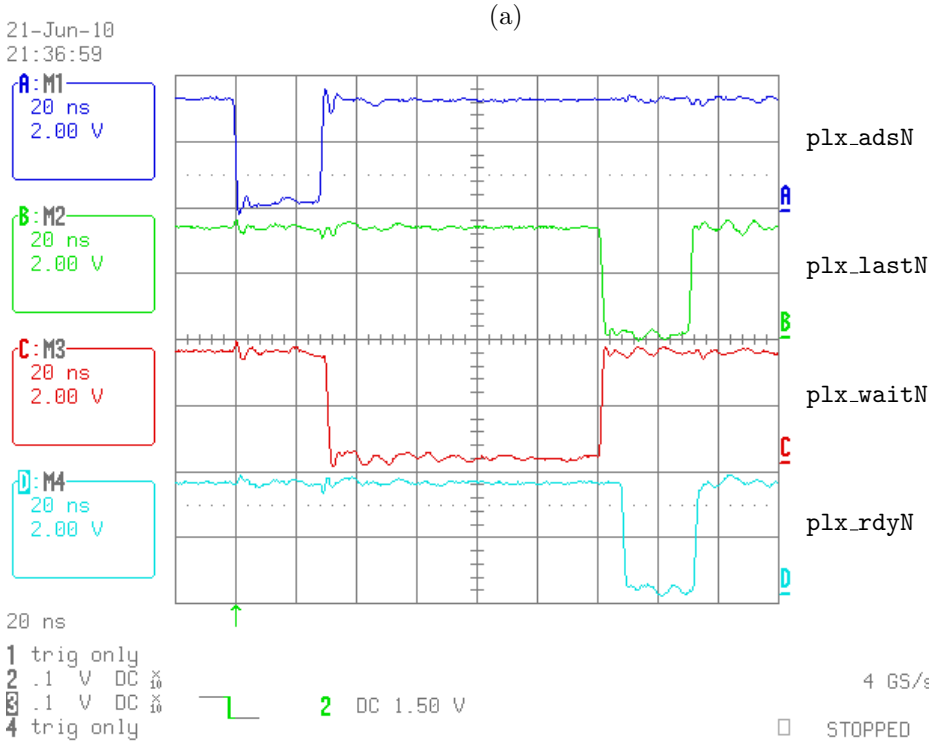


(b)

Figure 19: PLX PCI9054 local bus target single-transaction RAM access waveforms; (a) read-single, and (b) write-single.

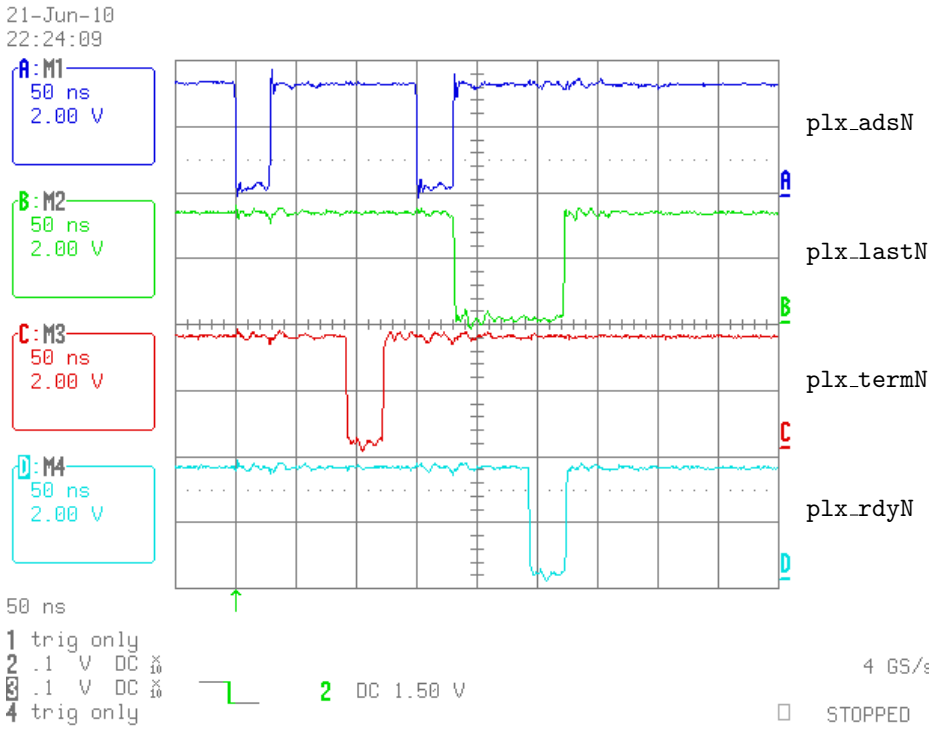


(a)

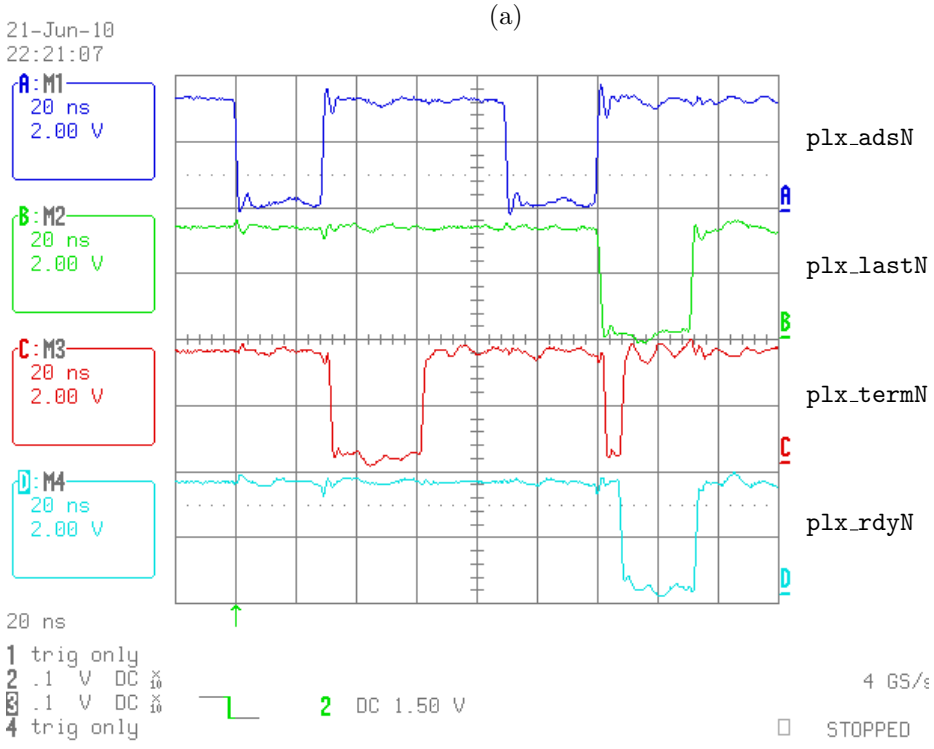


(b)

Figure 20: PLX PCI9054 local bus target single-transaction RAM access waveforms; read or write with (a) 2 wait-states, and (b) 3 wait-states.

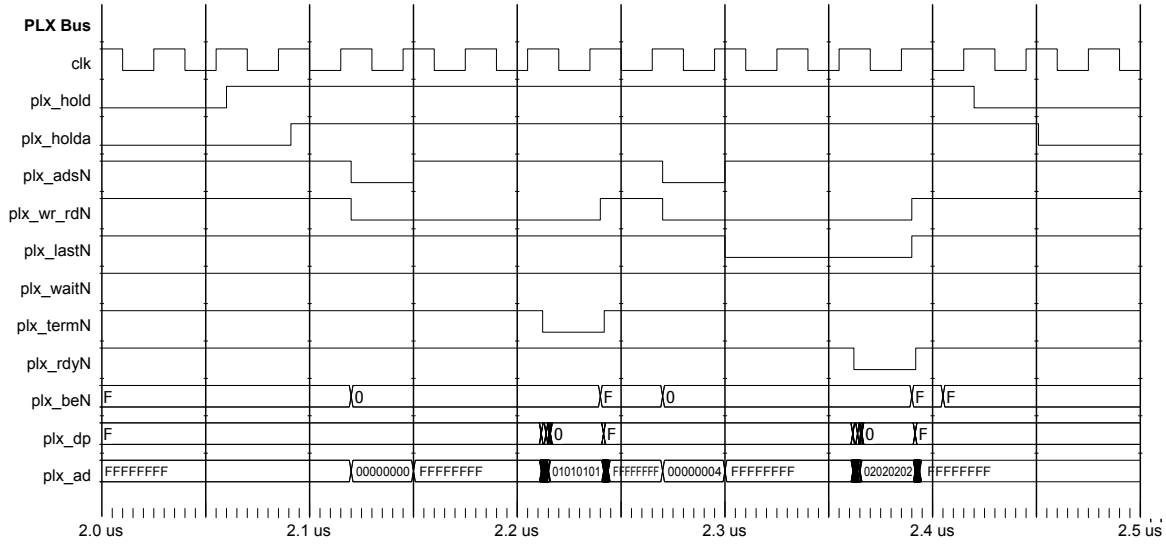


(a)

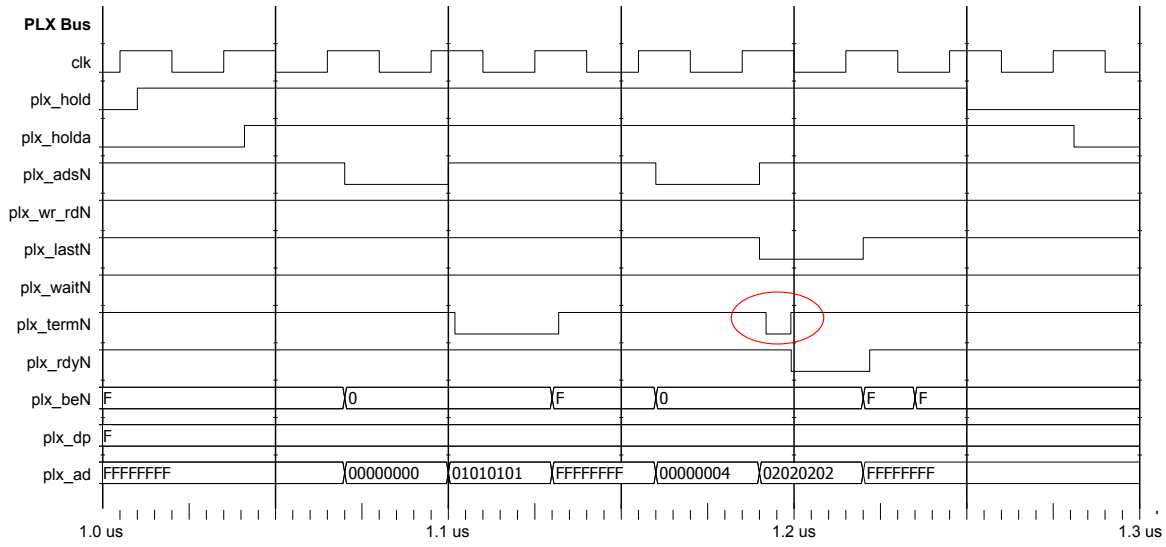


(b)

Figure 21: PLX PCI9054 local bus target single-transaction RAM burst access waveforms; (a) read-burst, and (b) write-burst.



(a)



(b)

Figure 22: PLX PCI9054 local bus target single-transaction RAM burst access simulation waveforms; (a) read-burst, and (b) write-burst. Note that the glitch observed on `plx_termN` in Figure 21(b) is also present in the write-burst simulation (red circle).

4.4 Target-only single/burst transaction RAM interface

The design in the previous section was used to demonstrate the operation of the `plx_waitN` and `plx_termN` signals and implemented only single-access transactions. The designs in this section implement single- and burst-access transactions. The designs do not support wait-states, since the bus master will not generate them by default (and there is no need to program them). The designs do not assert `plx_termN`. There are three variations on the target RAM interface implemented in the design `plx_target_ram_burst.vhd`;

A) Target RAM with no input or output registers.

- Figure 23 shows the block diagram.
- Figure 24 shows the FSM; `plx_target_ram_burst_a_fsm.vhd`.
- Figure 25 shows the read timing.
- Figure 26 shows the write timing.
- This target RAM interface is a *poor* design due to the lack of input or output registers. The same problems as discussed with the design in the previous section can be expected. The design is included in this section, as it allows the improvement in timing due to the addition of input and output registers to be analyzed.

B) Target RAM with output registers.

- Figure 27 shows the block diagram.
- Figure 28 shows the FSM; `plx_target_ram_burst_b_fsm.vhd`.
- Figure 29 shows the read timing.
- Figure 30 shows the write timing.
- The addition of output registers cuts the timing paths from the control inputs to outputs, and cuts the paths from the internal RAM to the output, and internal RAM through the parity logic, to the output.

C) Target RAM with input and output registers.

- Figure 31 shows the block diagram.
- Figure 32 shows the FSM; `plx_target_ram_burst_c_fsm.vhd`.
- Figure 33 shows the read timing.
- Figure 34 shows the write timing.
- The addition of both input and output registers cuts the majority of timing paths for signals coming onto the FPGA. The requirement that the PLX bus not be driven after the end of a transaction does require direct use of the `plx_lastN` signal to determine when to deassert `plx_rdyN` and to disable `plx_ad_oe` (and hence the `plx_ad` drivers).

The boolean generics `ENABLE_INPUT_REGISTERS` and `ENABLE_OUTPUT_REGISTERS`. select the design to test (a design with only input registers is not supported). The addition of registers to the input or output changes the timing of the control signals and data in the design, so each design instantiates a different FSM.

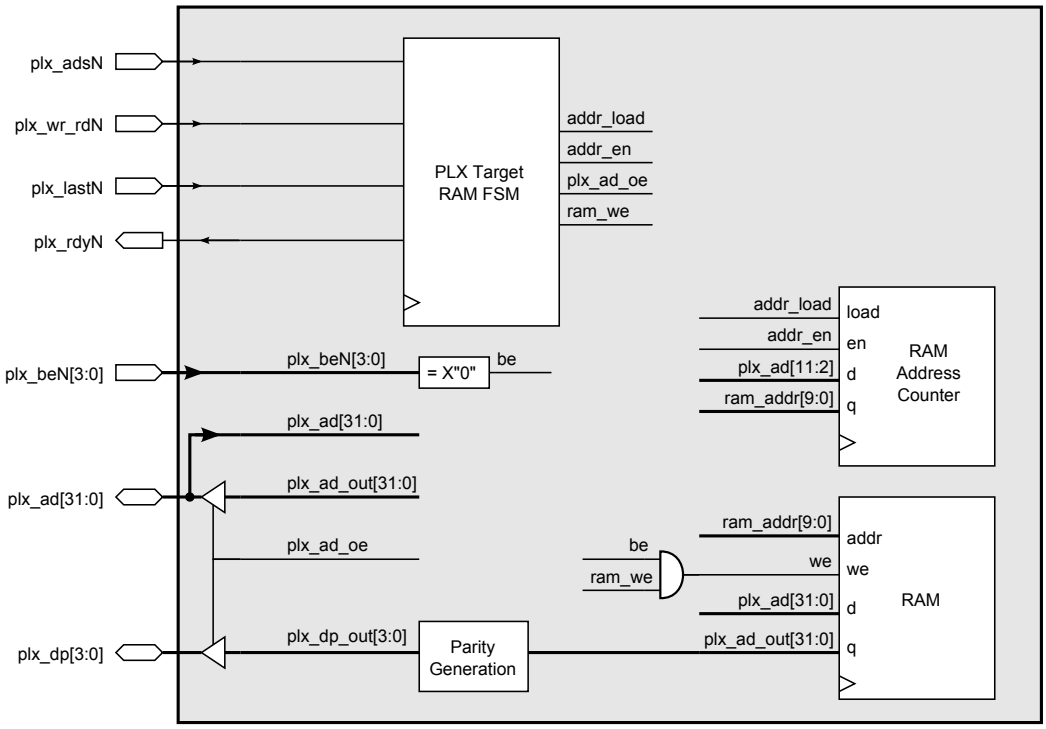


Figure 23: PLX PCI9054 local bus target burst-transaction RAM (with no input and output registers) interface block diagram.

Target RAM with no input or output registers

- The single-transaction read and write timing in Figures 25(a) and 26(a) produce identical PLX bus signal timing as shown for the previous design in Figures 15(a) and 17(a). However, the internal FSMs differ in their operation.
- The FSM in Figure 24 deals with single/burst transactions using the same read or write states.
- Figure 25 shows how all read transactions are initiated as a burst, with the internal read pipeline prefetching data from the RAM; the RAM address counter is loaded and enabled, with valid data ready two clocks later. For a read-single transaction, only one data phase is enabled onto the `plx_ad` bus.
- Figure 26 shows how write-single and write-burst transactions are dealt with. Since the master does not generate wait-states, any transaction phase in which the target asserts `plx_rdyN` is a transaction phase in which data can be written to RAM.
- The end of a transaction occurs when `plx_lastN` and `plx_rdyN` are detected asserted.

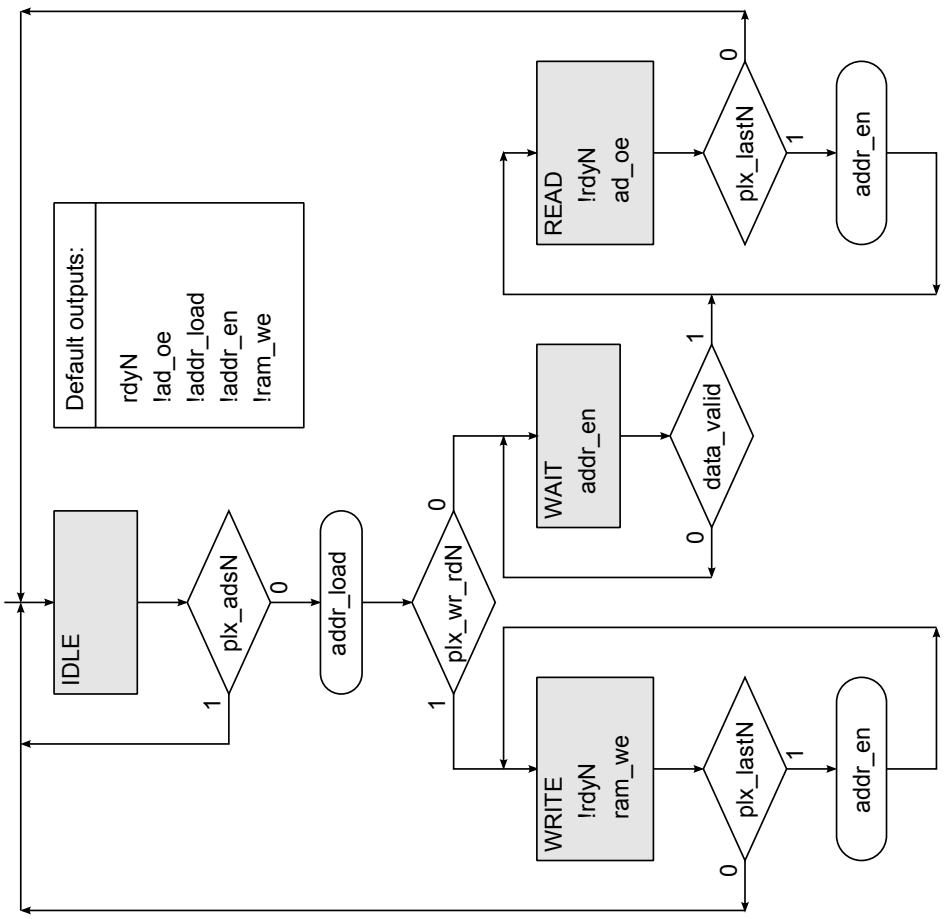


Figure 24: PLX PCI9054 local bus target burst-transaction RAM (with no input and output registers) interface finite-state machine (FSM).

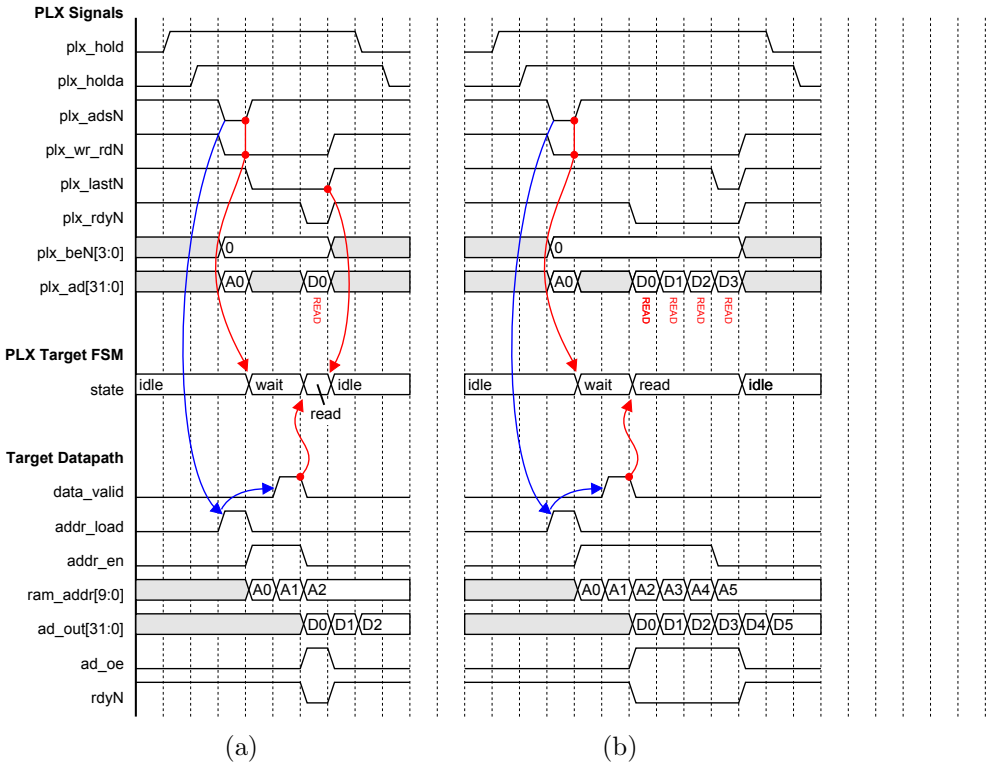


Figure 25: PLX PCI9054 local bus target burst-transaction RAM (with no input and output registers) read timing; (a) read-single, and (b) read-burst.

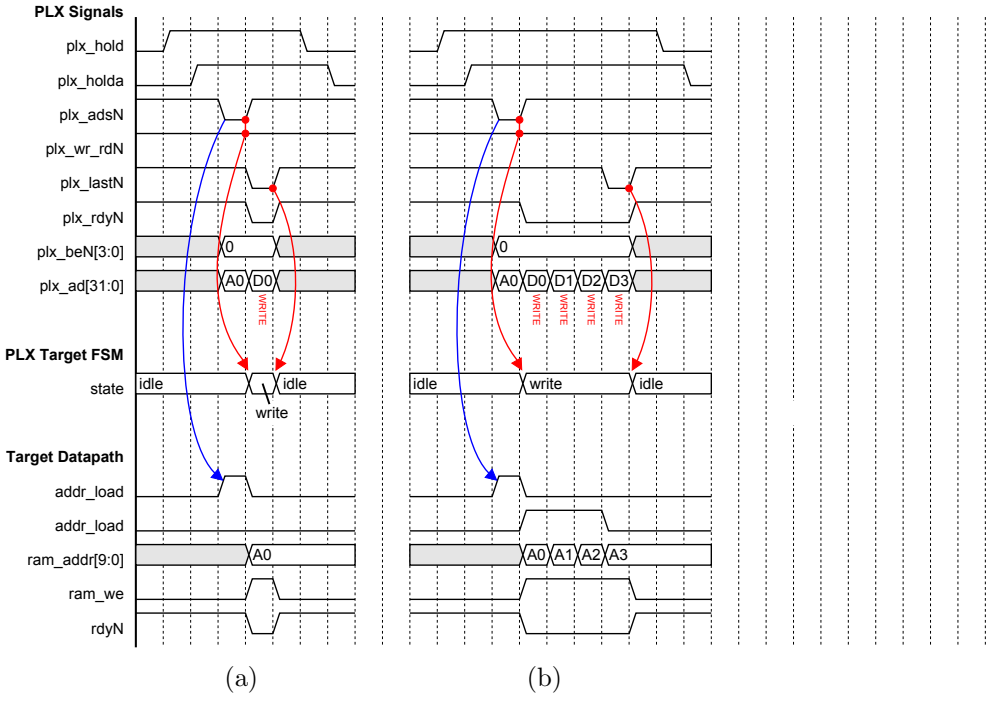


Figure 26: PLX PCI9054 local bus target burst-transaction RAM (with no input and output registers) write timing; (a) write-single, and (b) write-burst.

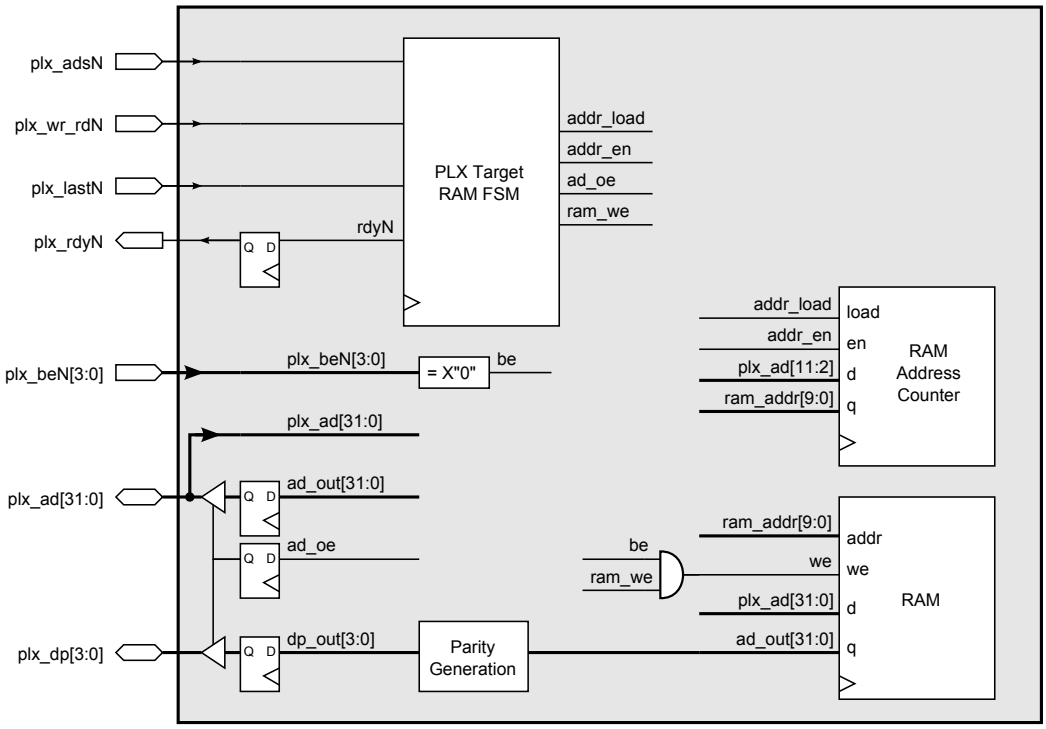


Figure 27: PLX PCI9054 local bus target burst-transaction RAM (with output registers) interface block diagram.

Target RAM with output registers

- The read timing in Figure 29 shows how the addition of the output registers delays the read data by one clock relative to the read timing for the design without the output registers (shown in Figure 25). Because the data is valid one clock later, the ready output needs to also be delayed by one clock. The output register on the `plx_rdyN` signal ensures that requirement is met; the FSM output `rdyN` is now registered as the PLX bus output `plx_rdyN`.

- Because the `plx_rdyN` signal is now delayed by a clock, the FSM write state sequence needs to assert `rdyN` a clock earlier (to get the same PLX bus write timing). A comparison of the write timing in Figure 30 with Figure 26 shows the `rdyN` signal asserted one clock earlier in the design with output registers.

- The end of a transaction occurs when `plx_lastN` and `plx_rdyN` are detected asserted. The FSM outputs the signal `rdyN` *not* `plx_rdyN`. To ensure the FSM correctly detects the transaction end, the FSM internally registers `rdyN` to create a signal with the same timing as `plx_rdyN`. The FSM in Figure 28 uses that internal ready signal to detect the transaction end. The use of an internal copy of the ready signal ensures that a critical timing path from the FSM output to the PLX bus and then back to the FSM is *not* created.

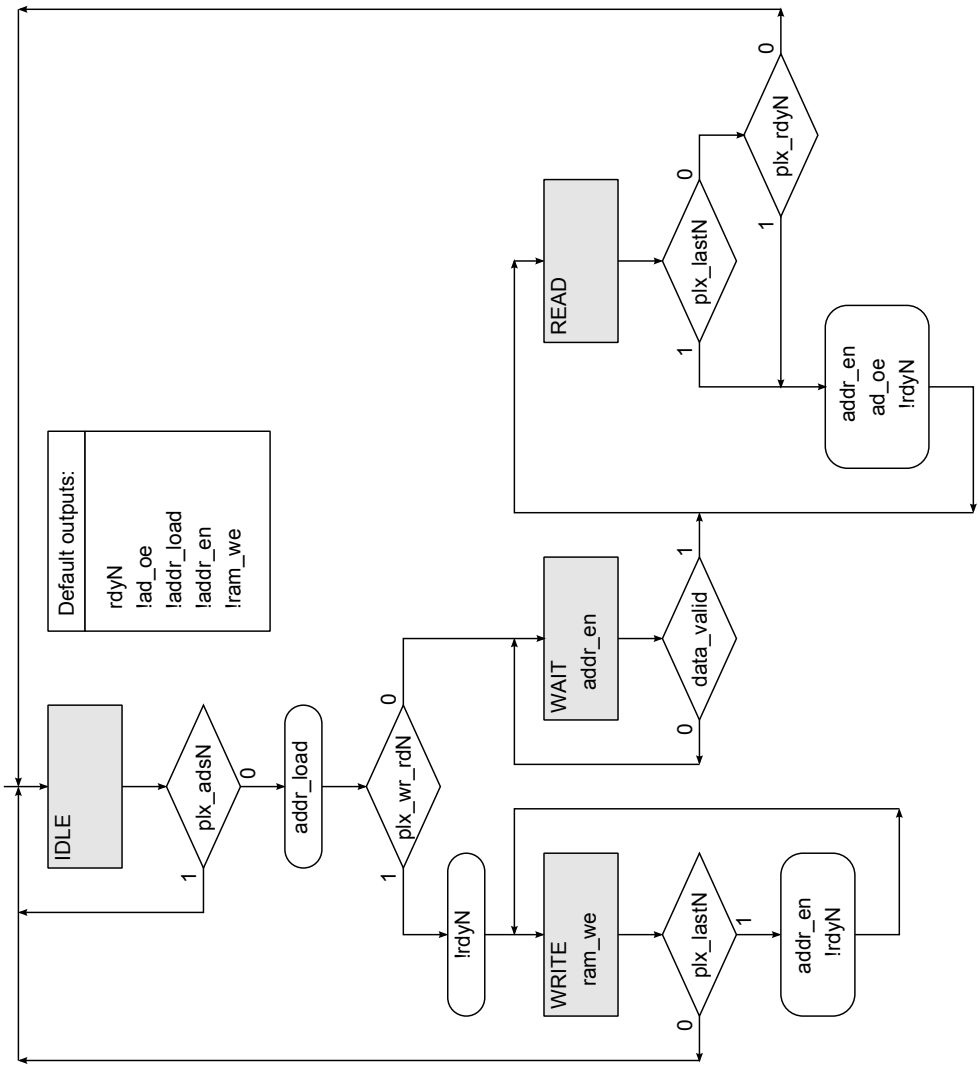


Figure 28: PLX PCI9054 local bus target single-transaction RAM (with output registers) interface finite-state machine (FSM).

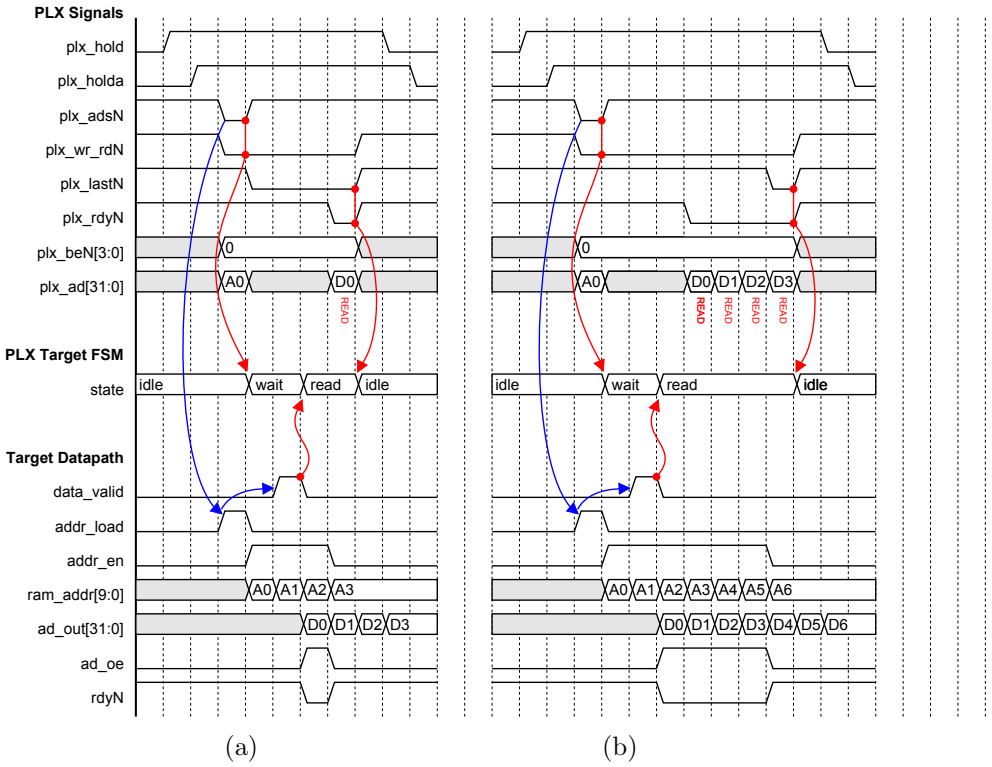


Figure 29: PLX PCI9054 local bus target burst-transaction RAM (with output registers) read timing; (a) read-single, and (b) read-burst.

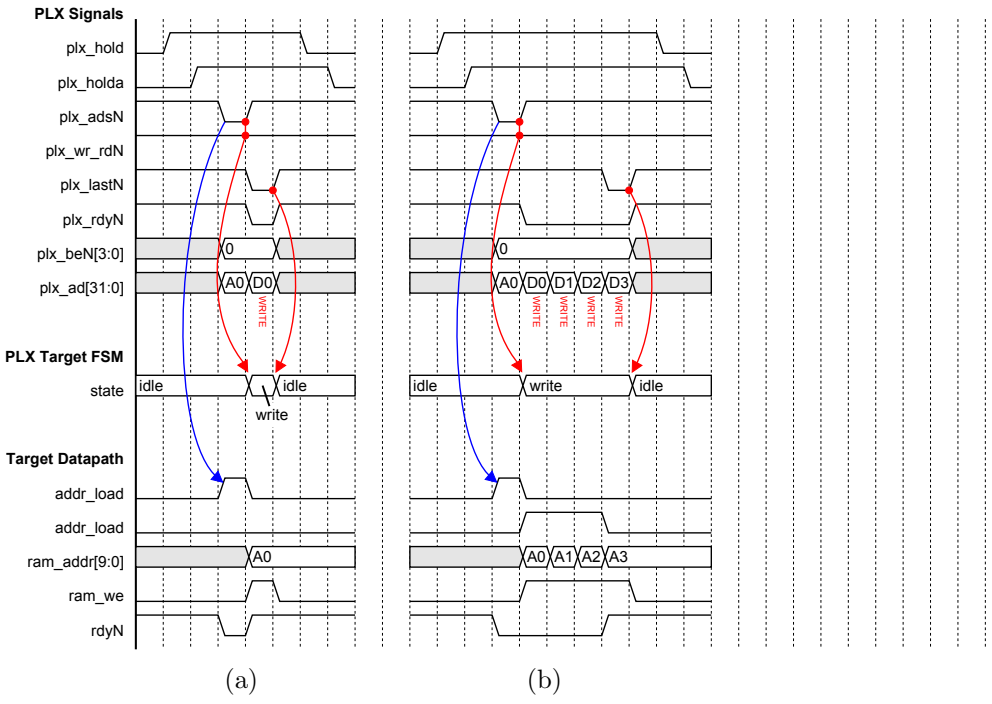


Figure 30: PLX PCI9054 local bus target burst-transaction RAM (with output registers) write timing; (a) write-single, and (b) write-burst.

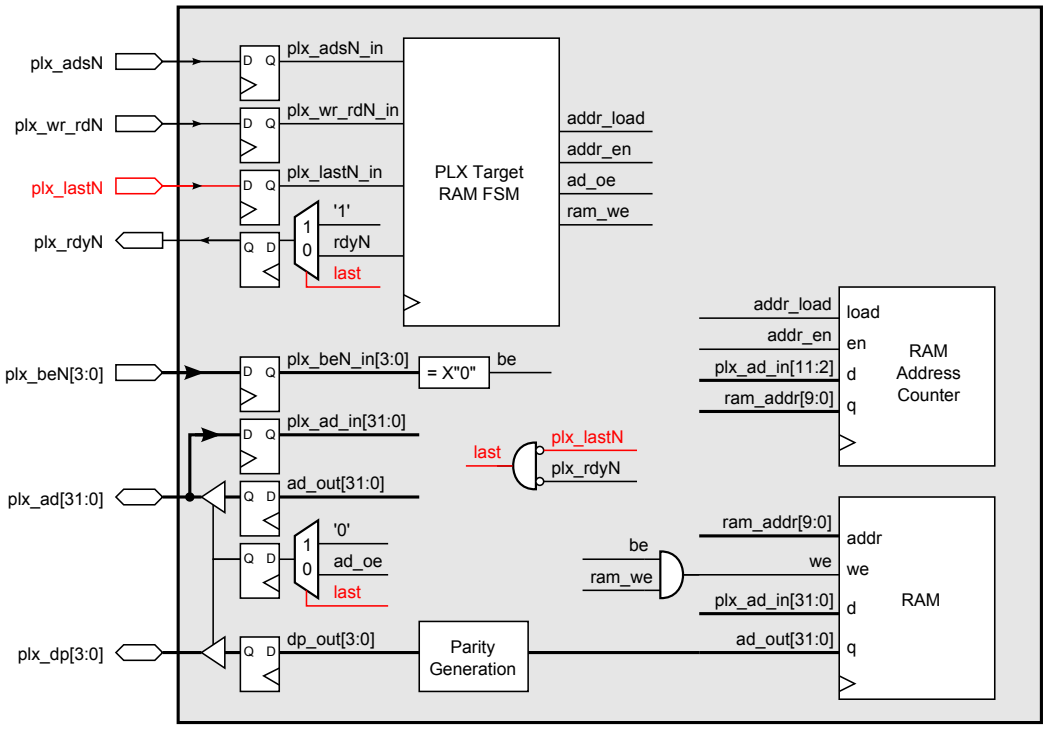


Figure 31: PLX PCI9054 local bus target burst-transaction RAM (with input and output registers) interface block diagram. The `last` signal (in red) is used to detect the transaction end and deassert outputs.

Target RAM with input and output registers

- The read and write timing in Figures 33 and 34 shows how the addition of input registers delays the start of the FSM sequence by a clock. The additional input clock delay causes the read data to be ready, and the first write phase to occur, one clock later than in the previous design.

- The end of a transaction occurs when `plx_lastN` and `plx_rdyN` are detected asserted *on the PLX bus*. The FSM uses the input signal `plx_lastN_in` which is delayed by one clock relative to `plx_lastN`. The FSM outputs the signal `rdyN`, which is output as `plx_rdyN`. To ensure the FSM correctly detects the transaction end, the FSM internally registers `rdyN` twice to create a signal with the same timing as `plx_rdyN_in`. The FSM in Figure 32 uses that internal ready signal to detect the transaction end. The use of an internal copy of the ready signal ensures that a critical timing path from the FSM output to the PLX bus and then back to the FSM is *not* created.

- The write FSM state uses the internal copy of the `plx_rdyN_in` signal to determine when to enable the RAM address counter and to assert RAM write-enable, eg. see the timing of these signals in Figure 34.

- The use of the delayed `plx_lastN_in` and `plx_rdyN_in` signals causes the FSM to detect the transaction end one clock *after* it has occurred on the PLX bus. The `last` signal is created using `plx_lastN` directly and another delayed copy of `rdyN` to generate a pulse at the transaction end. This pulse is used to deassert the `plx_rdyN` and `plx_ad_oe` signals at the end of a transaction, i.e., the FSM outputs are ignored during the last clock phase of a transaction.

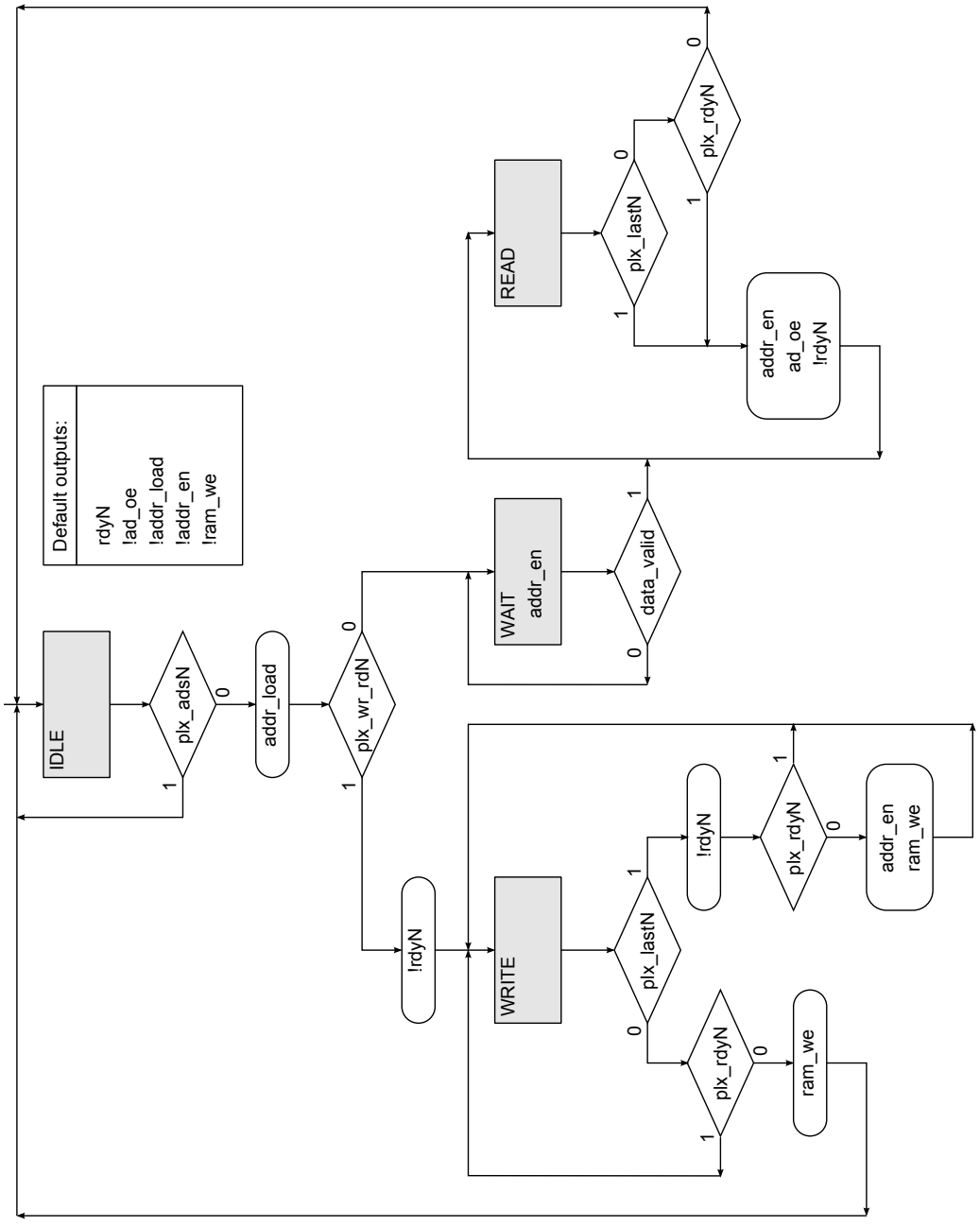


Figure 32: PLX PCI9054 local bus target single-transaction RAM (with input and output registers) interface finite-state machine (FSM).

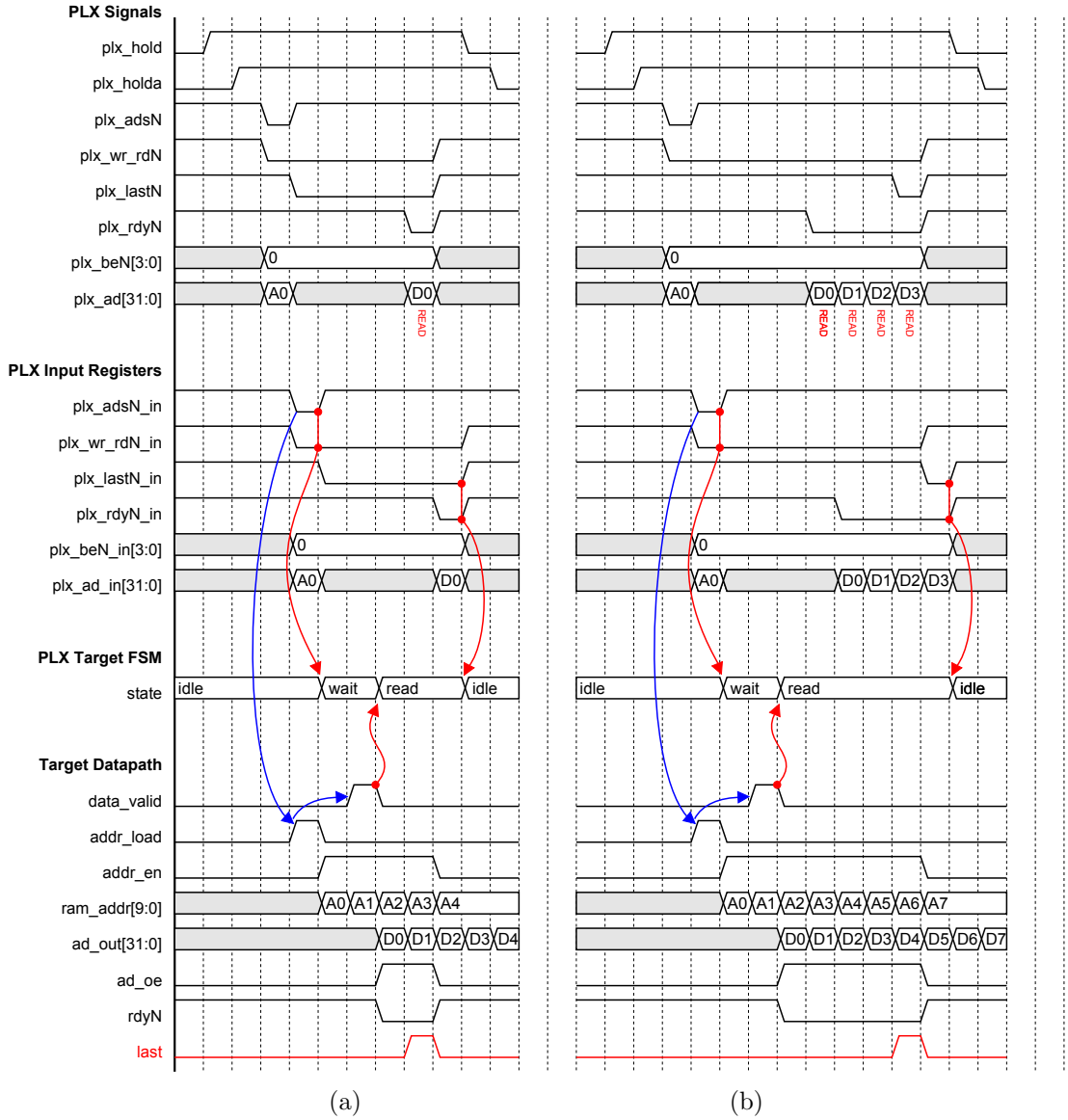


Figure 33: PLX PCI9054 local bus target burst-transaction RAM (with input and output registers) read timing; (a) read-single, and (b) read-burst. The `last` signal (in red) is used in the top-level design to detect the transaction end and deassert outputs.

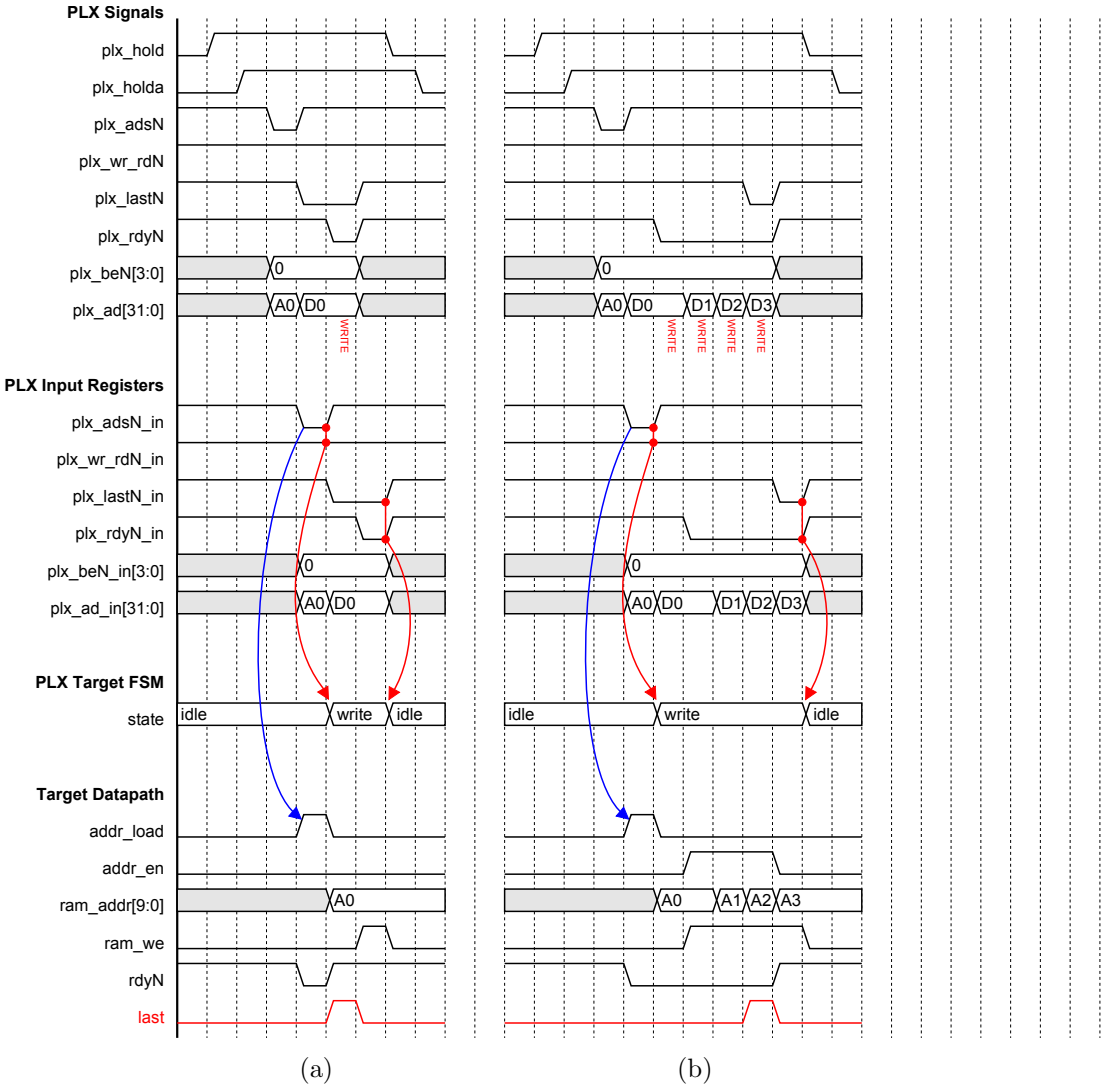


Figure 34: PLX PCI9054 local bus target burst-transaction RAM (with input and output registers) write timing; (a) write-single, and (b) write-burst. The `last` signal (in red) is used in the top-level design to detect the transaction end and deassert outputs.

Table 4: PLX PCI9054 local bus target burst-transaction RAM timing report.

Description	IOE Register		Worst-case value		
	Input	Output	t_{su} (ns)	t_h (ns)	t_{co} (ns)
Constraint			6.0	1.0	9.0
Design A	OFF	OFF	8.3	-1.3	11.8
Design B	OFF	OFF	9.3	-1.3	6.9
	OFF	ON	8.5	-1.3	8.7
Design C	OFF	OFF	4.1	-1.3	8.7
	OFF	ON	5.3	-1.3	8.7
	ON	OFF	5.9	-0.5	7.9
	ON	ON	5.3	-1.3	8.7

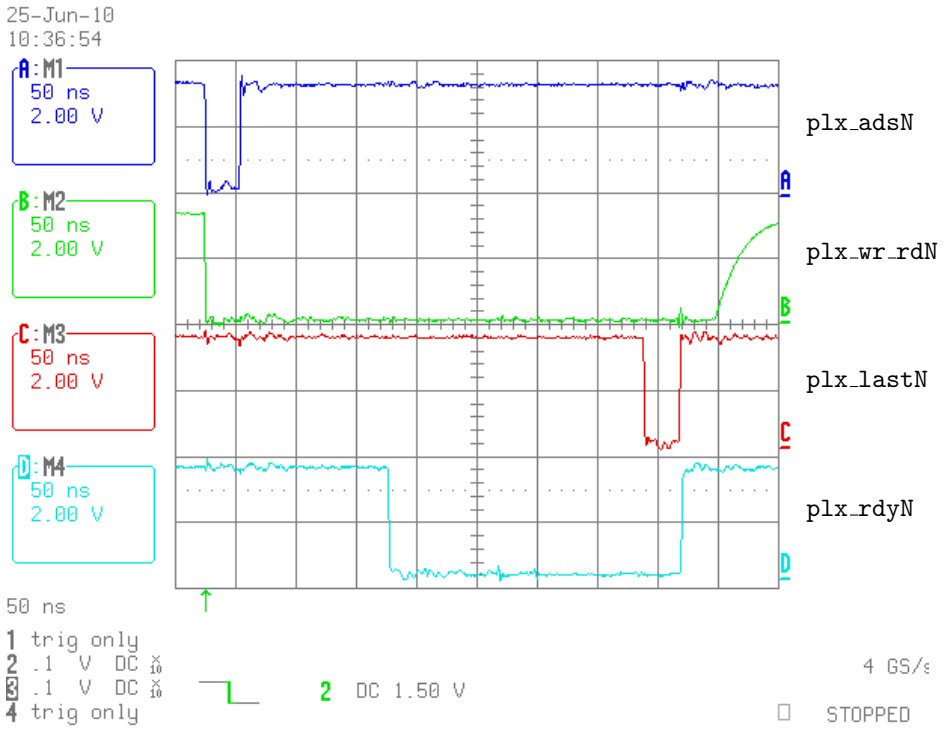
Hardware testing

Table 4 shows the timing report for the three target RAM designs (the values are the worst-case timing parameters reported by the Quartus Timing Analyzer tool). The FPGA setup constraint was determined based on a 50MHz clock (20ns period), worst-case PCI9054 clock-to-output delay of 12.5ns, and a desire to have 1.5ns margin, i.e., the constraint was 20.0ns - 12.5ns - 1.5ns = 6.0ns. The hold constraint was set to 1.0ns (since the FPGA had no problem meeting that). The clock-to-output constraint was based on a 50MHz clock, worst-case PCI9054 setup requirement of 9.5ns, and a desire to have 1.5ns margin, i.e., the constaint was 20.0ns - 9.5ns - 1.5ns = 9.0ns.

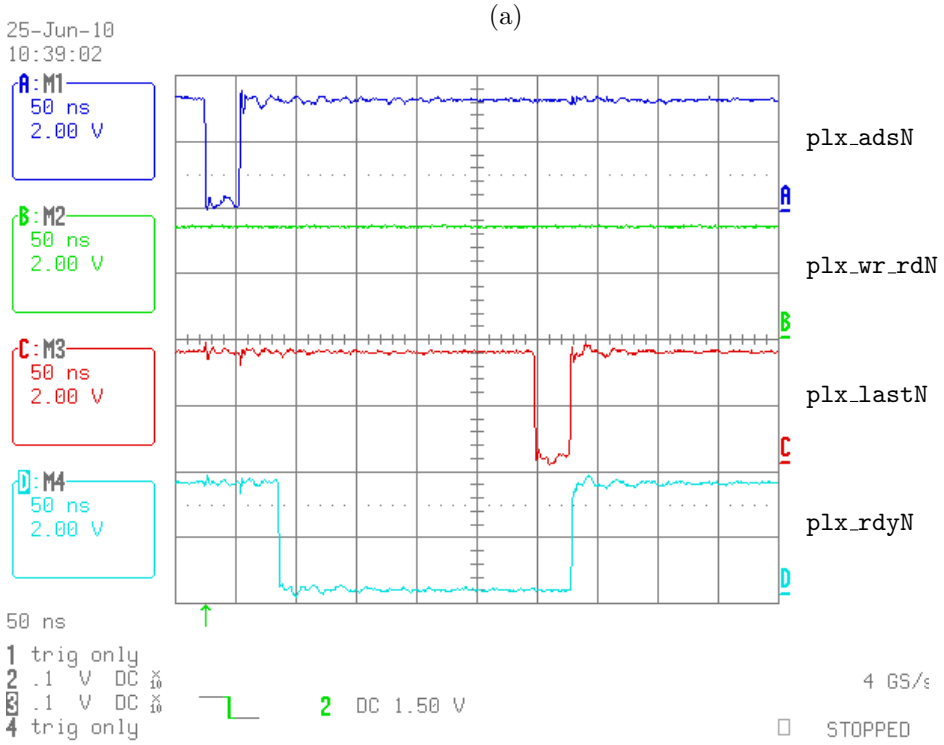
Table 4 shows that the only design that meets the timing requirements at 50MHz is the design with input and output registers (Design C). The FLEX10KE devices can have the input and output registers forced into the I/O elements (IOE) on the periphery of the FPGA by setting the `FAST_INPUT_REGISTER` and `FAST_OUTPUT_REGISTER` constraints to `ON`. The first two columns in Table 4 indicate the settings for the register constraints; they did not make much difference to any of the timing results.

The COBRA boards operate with a 33MHz clock (30ns) period, which provides an additional 10ns of timing margin. All the designs functioned correctly when tested on a COBRA board.

Figure 35 shows transaction waveforms for read and write bursts of 32-bytes. The burst transfers were between the board and a 4kB page in host memory. Burst transfers of 4kB were also tested; the transfers were generally completed with a single address strobe (one burst to transfer all data), i.e., ignoring the overhead of initial access, the transfer bursts at the maximum transfer rate of the local bus and hence PCI bus. A PCI bus analyzer was used to confirm that the transfers were occurring as burst transactions on the PCI bus; the 4kB transfers took around 33μs, i.e., achieved a PCI bus bandwidth of 4kB/(33μs$\times 2^{20}$) = 118MB/s. The PCI9054 configuration EEPROM was tested blank and programmed; similar performance was observed in either case. The 4kB transfers were initiated by the PCI9054 DMA controller, so the most critical setting is to use a `DMAMODE0` setting of `1C3` (bursting enabled).



(a)



(b)

Figure 35: PLX PCI9054 local bus target burst-transaction RAM access waveforms; (a) read-burst, and (b) write-burst (32-bytes, 8×32-bit words).

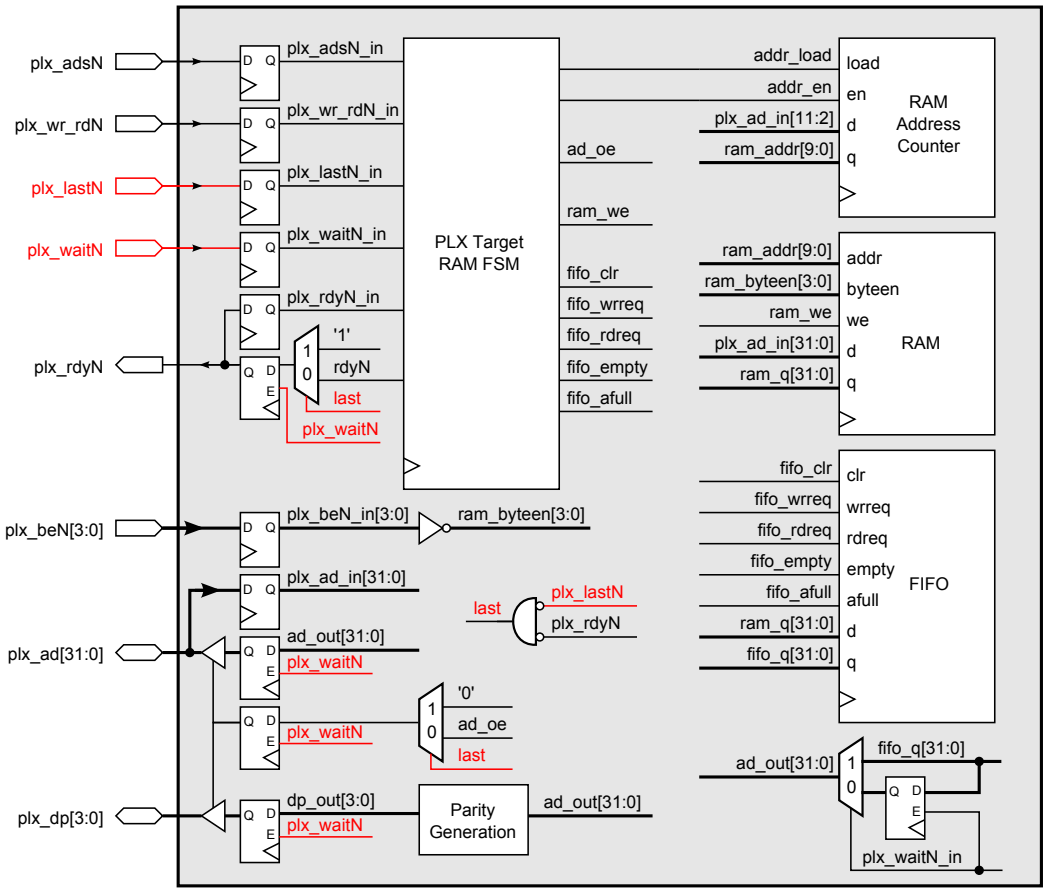


Figure 36: PLX PCI9054 local bus target burst-transaction RAM with master wait-state support interface block diagram.

4.5 Target-only single/burst transaction RAM interface with master wait-state support

The designs in the previous section demonstrated how to implement burst-transaction support, and showed how input and output registers can be used to improve the design timing. This section shows the impact of supporting master wait-states. Master wait-states *do not* need to be supported in a target interface, as the PCI9054 will not generate them if it is not programmed to. The purpose of the design in this section is to show how master wait-states complicate the pipelining of read-burst data[4]. The design does not terminate bursts, so `plx_termN` is never asserted.

Figure 36 shows the block diagram of the target RAM design that supports burst transactions and master wait-states[5]. Relative to the design that does not handle wait-states shown in Figure 31, the changes are;

- The wait-state signal `plx_waitN` has been added, and is used as an enable control on the output registers.
- The read data-path now has a FIFO on the output of the RAM, and a multiplexed register on the output of the FIFO.

[4]The design also demonstrates how to handle master wait-states for bus protocols where they can not be disabled.
[5]Timing report; $t_{su} = 5.2$ns, $t_h = -1.3$ns, $t_{co} = 7.9$ns.

- The `plx_rdyN` signal is fed-back to the control FSM, via `plx_rdyN_in`, so that the effect of wait-states on the ready handshake can be accounted for in the control FSM. The design actually internally duplicates the path shown in the figure, so that a critical timing path is not created.
- The RAM is shown with byte-enables. This is not actually a design change, but a diagram change. Generics in the design control whether FLEX10KE-style RAM or Cyclone/Stratix RAM is instantiated.

Figure 37 shows the control FSM, Figures 38 and 39 show the read timing without and with master wait-states, and Figures 40 and 41 show the write timing without and with master wait-states. The timing of all of these figures is a result of the logic required to implement support for master wait-states during read-bursts, i.e., the timing in Figure 39(b).

During a read-burst, the control FSM prefetches data from the RAM and outputs it onto the PLX bus, eg, given a burst start address `A0`, a short while later the read-data `D0` is driven onto the PLX bus followed a clock later by `D1`, and so on. Figure 39(b) shows that after the `D0` transaction on the PLX bus, the wait-state control `plx_waitN` asserts low, requiring that the data `D1` must be *held* on the PLX bus until the wait-state signal deasserts. The FSM can not implement this data holding (wait-state) logic, as it can not react fast enough, due to the use of the PLX bus input and output registers. The hold logic is implementing using the `plx_waitN` control signal directly. The wait-state signal is used as an enable control on the `plx_ad` bus output registers so that when the signal is low (wait-states inserted), the output data (eg. `D1`) is held on the PLX bus until the wait-state ends (`plx_waitN` goes high, and the output registers are enabled). Because the data is driven onto the PLX bus during reads by the target, and the ready signal indicates when read data is valid, the output registers for `plx_ad_out`, `plx_dp_out`, `plx_ad_oe` and `plx_rdyN` must all be enabled using `plx_waitN` as shown in Figure 36 (the figure shows the register inputs `ad_out`, `dp_out`, and `ad_oe`, which once registered become the signals `plx_ad_out`, `plx_dp_out`, and `plx_ad_oe`, which go to the output tri-states).

The use of `plx_waitN` to hold data on the PLX bus during a read-burst wait-state meets the protocol requirement of the PLX bus. Once the wait-state signal deasserts, the next data phase must be valid on the PLX bus, or the `plx_rdyN` signal needs to deassert (to indicate a target wait-state). The bottom traces in Figure 39(b) show the signal required on `ad_out` to meet the requirement of having the next data phase valid on the bus, and `plx_rdyN` asserted. The figure also shows the FIFO output, `fifo_q`, along with FIFO read controls derived from the registered input PLX bus control signals; the `plx_waitN` input register causes the output of the FIFO to change before it is driven out onto the PLX bus (compare the `fifo_q` output to what is required on the `plx_ad` bus on the next clock). The multiplexed register (or *hold* register) on the output of the FIFO acts like a one-entry deep FIFO; it captures the output data from the FIFO, in case it is needed on the next clock (or clocks) due to a wait-state (or wait-states). Figure 39(b) shows how the `ad_out` multiplexer controlled by `plx_waitN_in` appropriately selects the FIFO output, `fifo_q`, or the hold register, `fifo_r`, to deliver the appropriate read-burst data onto the `plx_ad` bus.

Figures 39(a) and 41(a) show how the use of the `plx_waitN` enable control on the `plx_rdyN` output delays the first ready phase during read or write single transactions; the clock phases in which `plx_rdyN` is shown with a green ellipse can not change due to the fact that `plx_waitN` is low, i.e., the `plx_rdyN` output register is held (high), its only once `plx_waitN` deasserts that the FSM signal `rdyN` actually makes it onto the PLX bus as `plx_rdyN` (low).

The design shown in Figure 36 is implemented in `plx_target_ram_burst_wait.vhd`, which contains the finite-state machine `plx_target_ram_burst_wait_fsm.vhd` shown in Figure 37. The testbench `plx_target_ram_burst_tb.vhd` is used to test both this design, and the designs from the previous section; generics control which design to select, and whether to test master wait-states. The testbench tests the design using linearly increasing master wait-states and randomly generated wait-states. Figure 42 shows transaction waveforms for burst transactions with master wait-states.

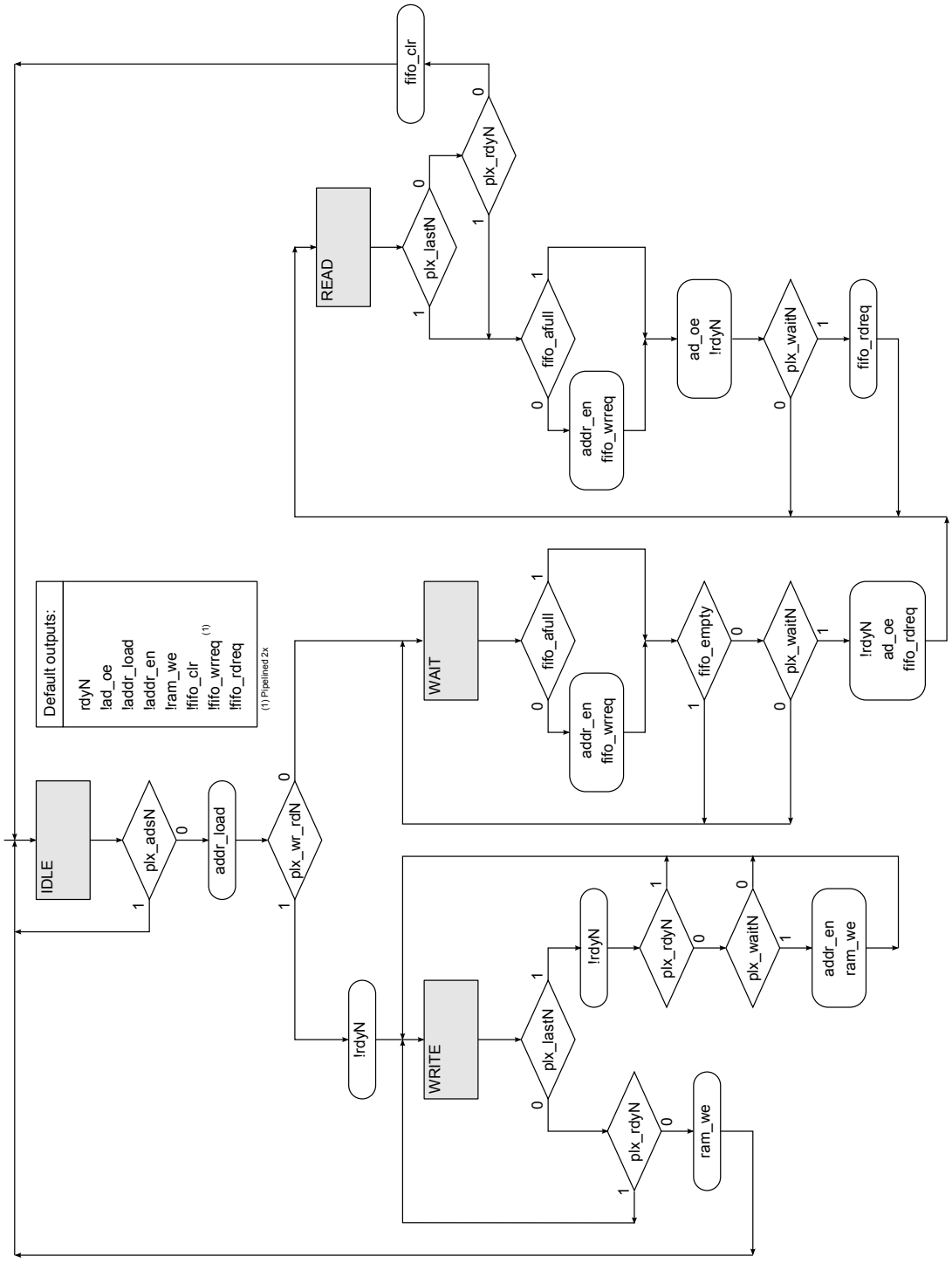


Figure 37: PLX PCI9054 local bus target burst-transaction RAM with master wait-state support interface finite-state machine (FSM).

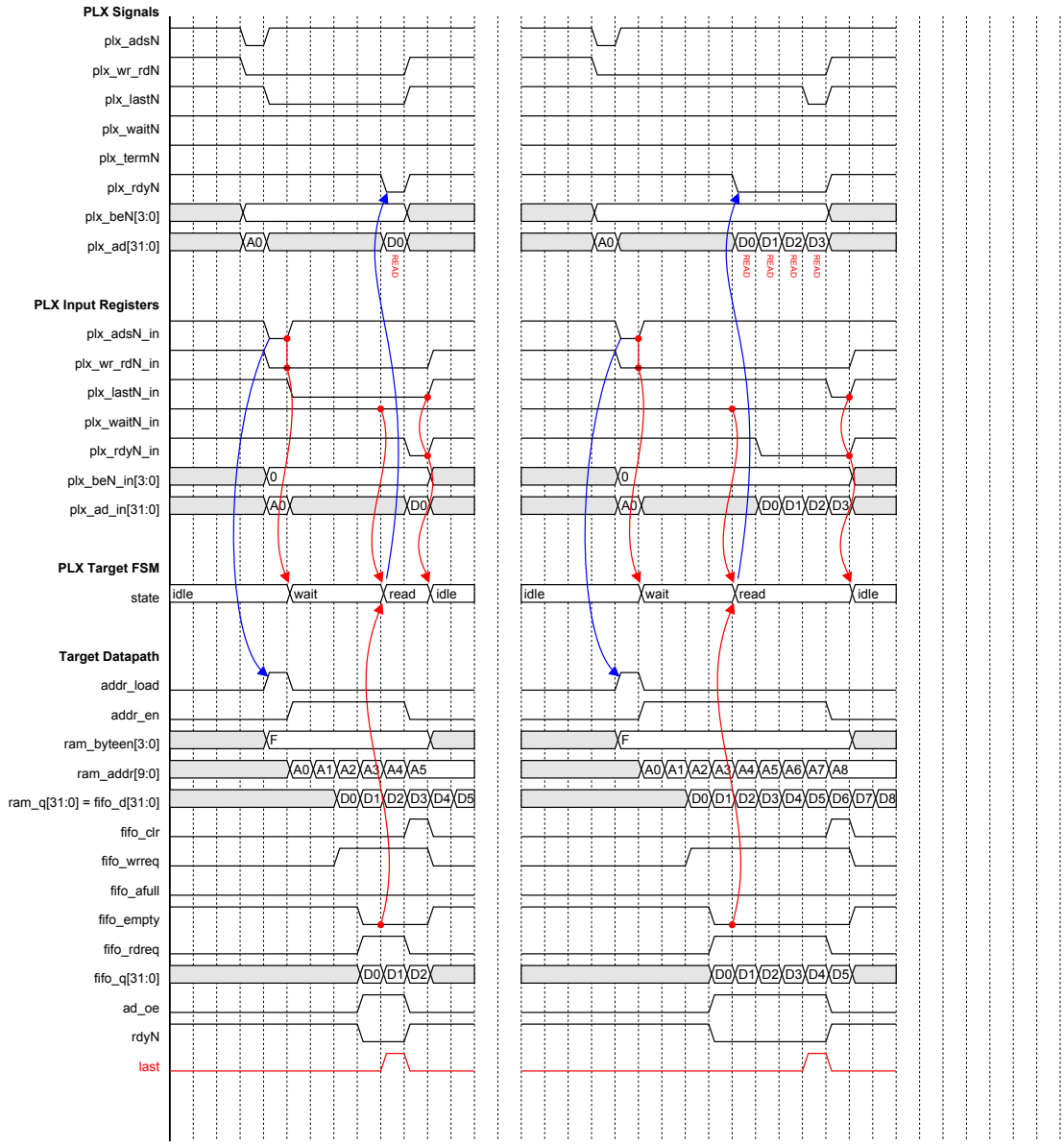


Figure 38: PLX PCI9054 local bus target burst-transaction RAM with master wait-state support read timing; (a) read-single, and (b) read-burst. The `last` signal (in red) is used in the top-level design to detect the transaction end and deassert outputs.

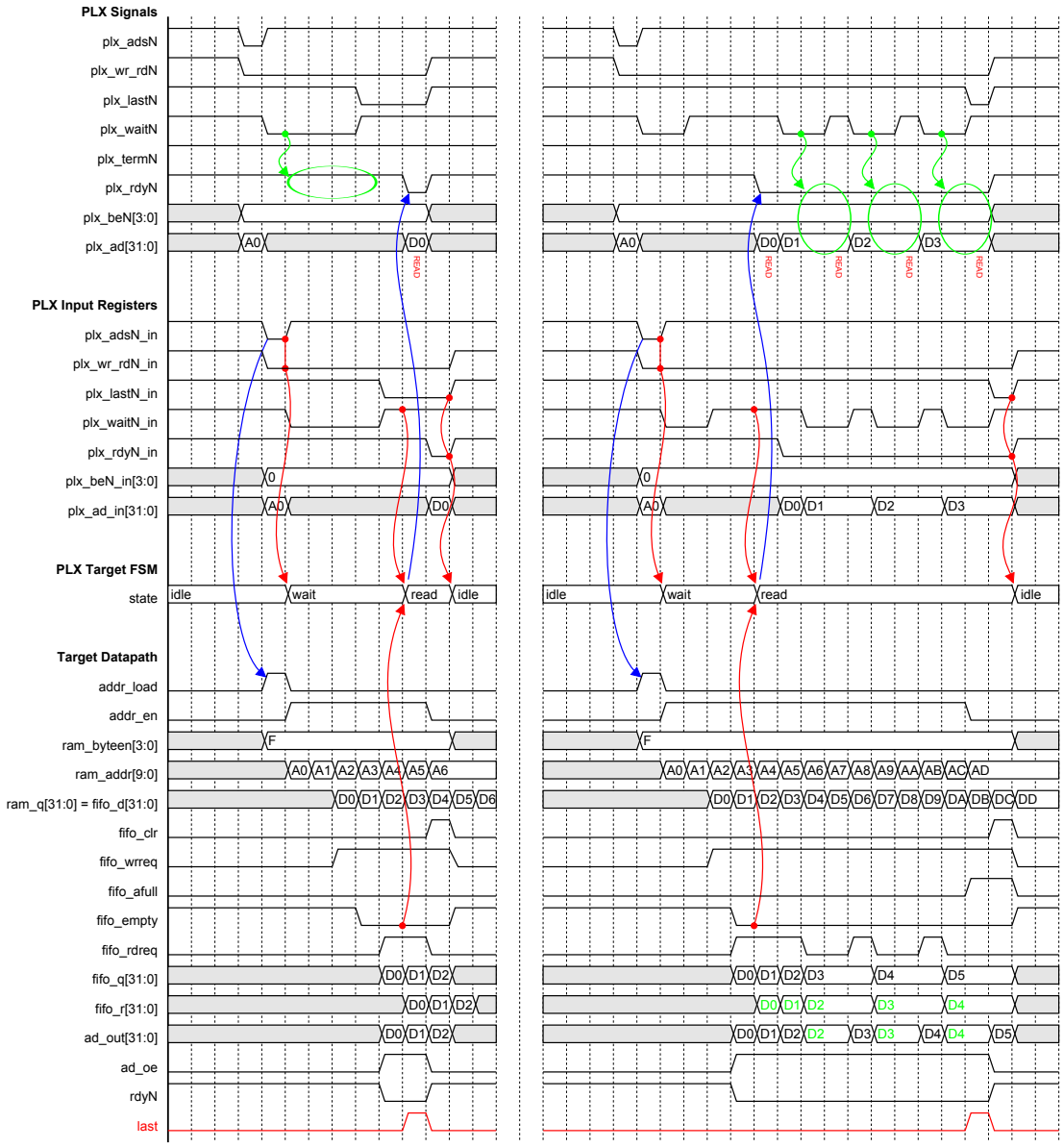
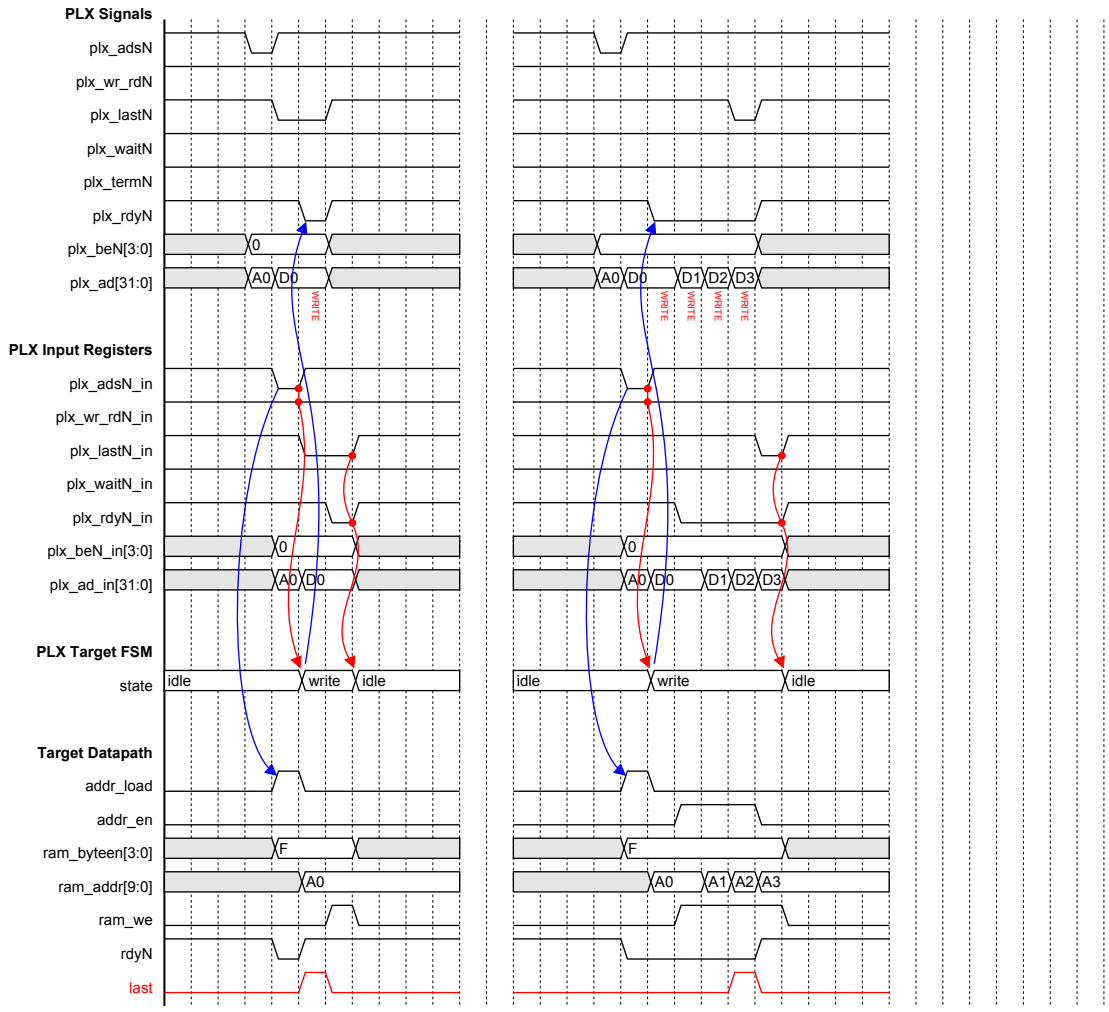


Figure 39: PLX PCI9054 local bus target burst-transaction RAM with master wait-state support read timing; (a) read-single with four wait-states, and (b) read-burst with two wait-states. The `last` signal (in red) is used in the top-level design to detect the transaction end and deassert outputs.



(a) (b)

Figure 40: PLX PCI9054 local bus target burst-transaction RAM with master wait-state support write timing; (a) write-single, and (b) write-burst. The `last` signal (in red) is used in the top-level design to detect the transaction end and deassert outputs.

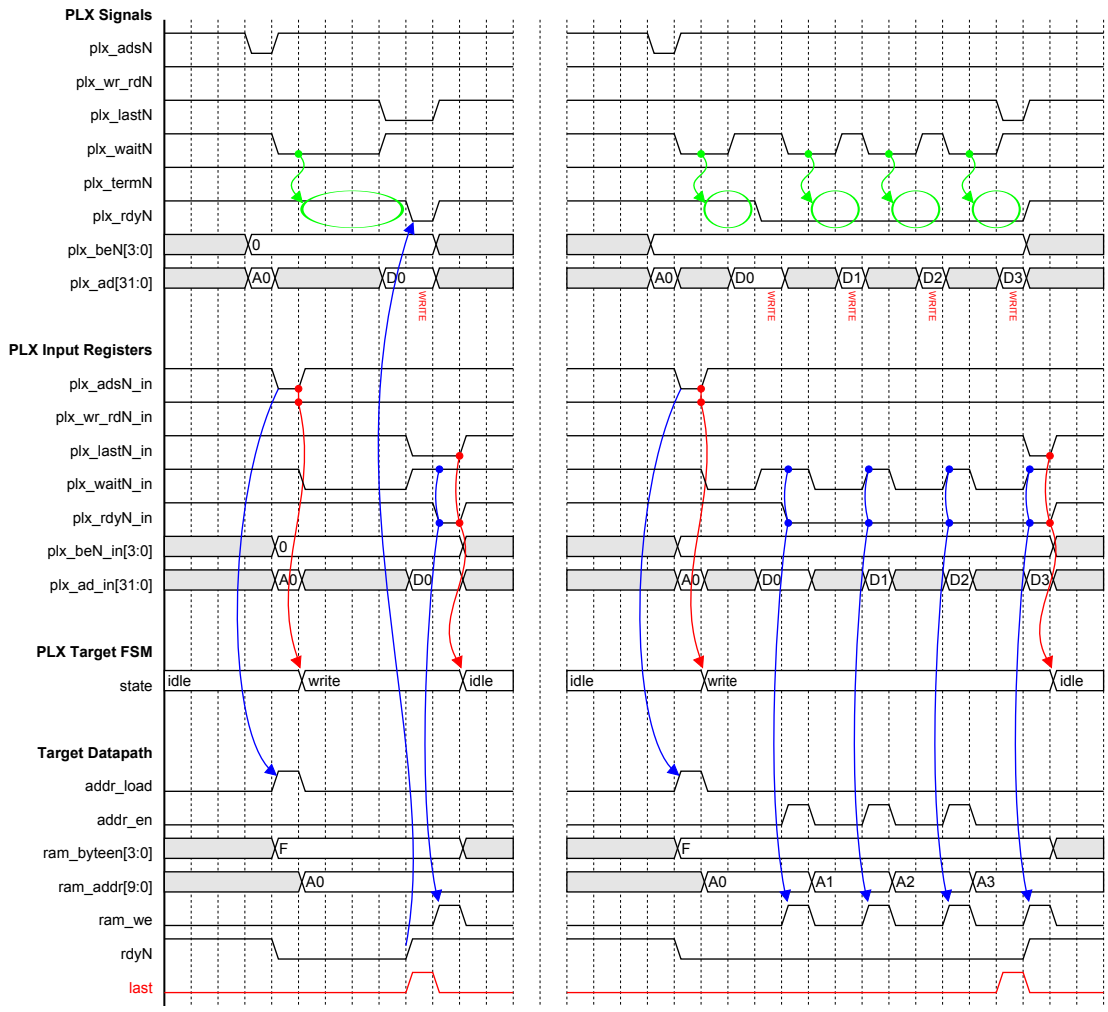
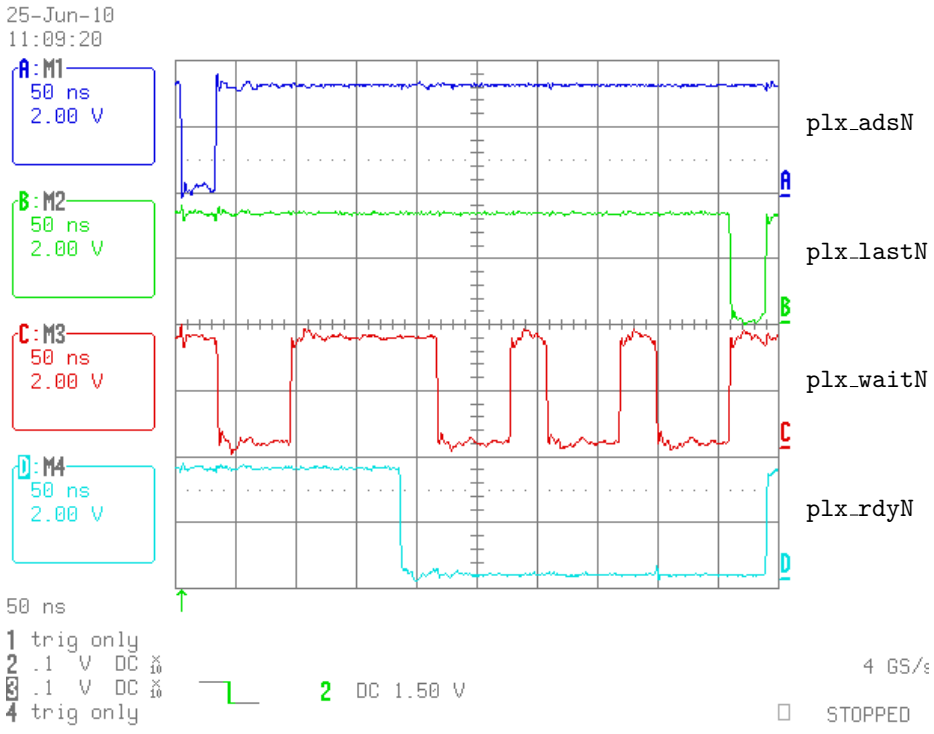
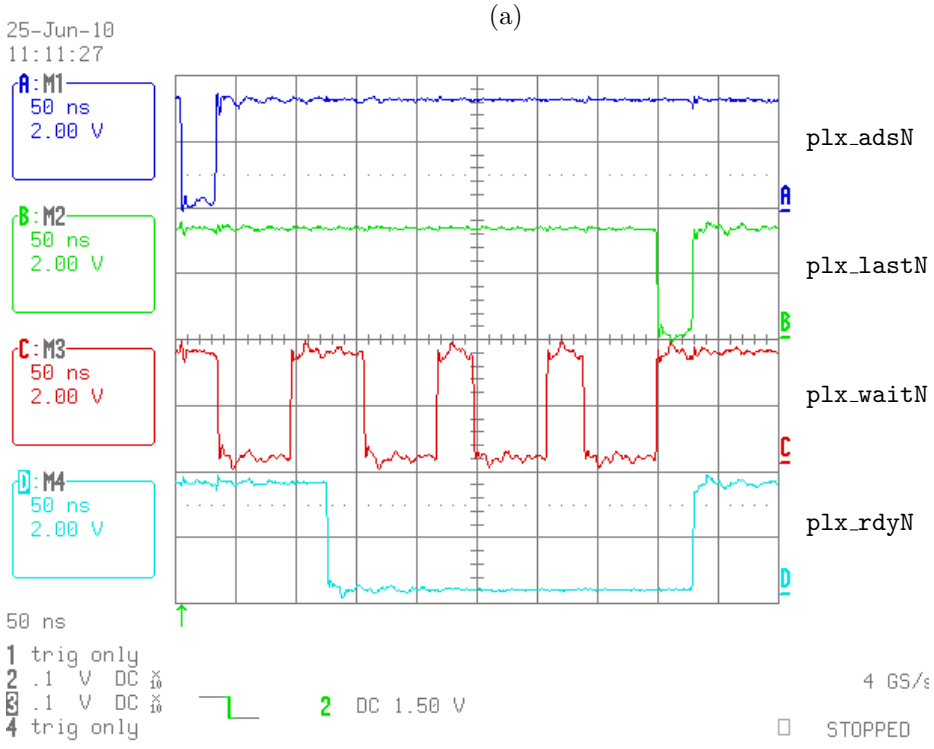


Figure 41: PLX PCI9054 local bus target burst-transaction RAM with master wait-state support write timing; (a) write-single with four wait-states, and (b) write-burst with two wait-states. The `last` signal (in red) is used in the top-level design to detect the transaction end and deassert outputs.



(a)



(b)

Figure 42: PLX PCI9054 local bus target burst-transaction RAM access waveforms with two master wait-states; (a) read-burst, and (b) write-burst (16-bytes, 4×32-bit words).

4.6 Master/Target interface

The PLX PCI9054 local bus supports a master/target interface as described in Sections 2.6 and 2.7. A device on the PLX PCI9054 local bus can arbitrate to acquire the bus, becoming the bus master, and then accessing the PCI bus or PCI9054 internal registers, i.e., the PCI9054 is a target device. The hardware tests in this document accessed the PCI9054 control registers via the PCI bus.

The COBRA board contains a Texas Instruments TMS320LC31 32-bit floating-point DSP. The PCI9054 and DSP both interface to the system controller FPGA (Altera FLEX10KE). The system controller FPGA implements master/target interfaces on both the PLX local bus, and the DSP local bus. The PCI interface can be used to access devices on the DSP bus, and the DSP can access devices on the PCI bus (though that feature is not used). The main reason for the system controller to implement a PLX local bus master/target interface is to provide the COBRA DSP with access to the PCI9054 internal registers so that the doorbell and mailbox registers can be used for communications between the host CPU and DSP (see the COBRA Device Driver documentation [1]).

TODO
Add an example PCI9054 local bus FPGA master interface example. This will require a DSP BFM, an FPGA DSP bus target interface, the FPGA internal logic, and a PLX bus master interface. This code will need to be extracted from the COBRA HDL code.

5 Software interfacing

The simplest way to develop a software interface to the PLX PCI9054 is to first start with direct register access. Under recent kernels, the Linux `sysfs` filesystem provides device nodes that expose the PCI base address regions (BARs) of a PCI device; see the Linux documentation file `Documentation/filesystems/sysfs-pci.txt`. Older kernels that do not support the `sysfs` device nodes can use the `pci_io.ko` driver described in 'Linux Device Driver Design' (along with its user-space interface) [2].

The user-space application `pci_debug` was developed while writing this document to access PCI device registers. The application accesses PCI BARs using the `sysfs` method, and uses the `readline` library to provides an interactive command-line (command history, editing, and up-arrow access to previously entered commands). Hardware measurements in this document were obtained by powering-up a COBRA board with a blank configuration EEPROM, then `pci_debug` was used to access the PLX registers in BAR0 to enable Local Address Space 0 (LAS0). LAS0 was then accessed and patterns written to the PLX target device (in the local bus FPGA). Read/write accesses to LAS0 were used to capture single-access transaction waveforms. Read/write burst-accesses were generated by accessing BAR0 and programming the DMA controller to generate transfers between either a second COBRA board, or the host CPU main memory.

DMA transfers from the COBRA board to host memory require a device driver that provides the physical address of a block of host memory. The 'Linux Device Driver Design' [2] source contains a driver, `simple_memory.ko`, and user-space application `simple_memory_test`. The driver can be installed (using `insmod`), and then the kernel message log (`dmesg`) can be used to determine the physical address of that memory. That address can then be used to program the DMA controller. A PLX PCI9054 local bus write-burst transaction is generated by performing a DMA from the host memory to the COBRA board. The `simple_memory_test` application can be used to write a pattern to the memory, and then after the DMA, `pci_debug` can be used to view the LAS0 memory to confirm the DMA was successful (no data corruption). A PLX PCI9054 local bus read-burst transaction is generated by performing a DMA from the COBRA board to host memory. In this case, `pci_debug` is used to setup a pattern, and `simple_memory_test` used to confirm it on the host after the DMA completes.

Once the basic transfers between a host CPU and a custom board are tested, a design-specific device driver can be developed. The COBRA board device driver is documented in detail in [1]. That driver documentation also includes a generic description on how the doorbell registers in the PCI9054 can be used to implement flow-control between the host CPU and a target CPU located on the local bus of the PCI9054. Documentation and source code can be accessed from 'COBRA docs' link on the left-side of the page: http://www.ovro.caltech.edu/~dwh/correlator/.

References

[1] D. W. Hawkins. COBRA Device Driver. Correlator Documentation, 2006. http://www.ovro.caltech.edu/~dwh/correlator/pdf/cobra_driver.pdf.

[2] D. W. Hawkins. LNX-723: Linux Device Driver Design. Embedded Systems Conference, 2006. http://www.ovro.caltech.edu/~dwh/correlator/pdf/LNX-762-Hawkins.pdf.

[3] PLX Technologies, Inc. PCI 9054 PCI I/O Accelerator (version 2.1). Data Book, January 2000. http://www.plxtech.com/.