ECE 448
Lecture 20
Software/Hardware Co-design Using the FPro System
Part 3:
I/O Register Map & a Wrapper of the Sorting Core

Developing a software/hardware implementation using an FPro system

Conceptual Design:

C1. Software/hardware partitioning

C2. I/O register map of the IP core

Hardware Design:

H1. Basic circuit performing the required functionality

* datapath * controller * top-level * functional verification

H2. A wrapper matching the interface of an MMIO core

*** design adjustments * coding * functional verification**

Software Design:

S1. Software driver

* declarations (.h) * implementations (.cpp) * testing

S2. Application based on functions of custom and standard cores

Setting Size of Memory
to Initialize & Sort

SW3..SW0=k=$\log_2$ N,
where N= number of elements to initialize and sort

k	N	w
4	16	8
8	256	8
9	512	16
10	1024	16
11	2048	16
12	4096	16
13	8192	16

Setting Size of Memory
to Initialize & Sort

Suggested values:

1. Debugging: k=4
   N=16 elements of the width w=8 bits

2. Demo: k=4 and k=8
   N=16 and N=256 elements of the width w=8 bits

3. Timing measurements: k=9..13
   from N=512 to N=8192 elements of the width w=16 bits

Developing a software/hardware implementation using an FPro system

Conceptual Design:

C1. Software/hardware partitioning

C2. I/O register map of the IP core

Hardware Design:

H1. Basic circuit performing the required functionality

*** datapath * controller * top-level * functional verification**

H2. A wrapper matching the interface of an MMIO core

* design adjustments * coding * functional verification

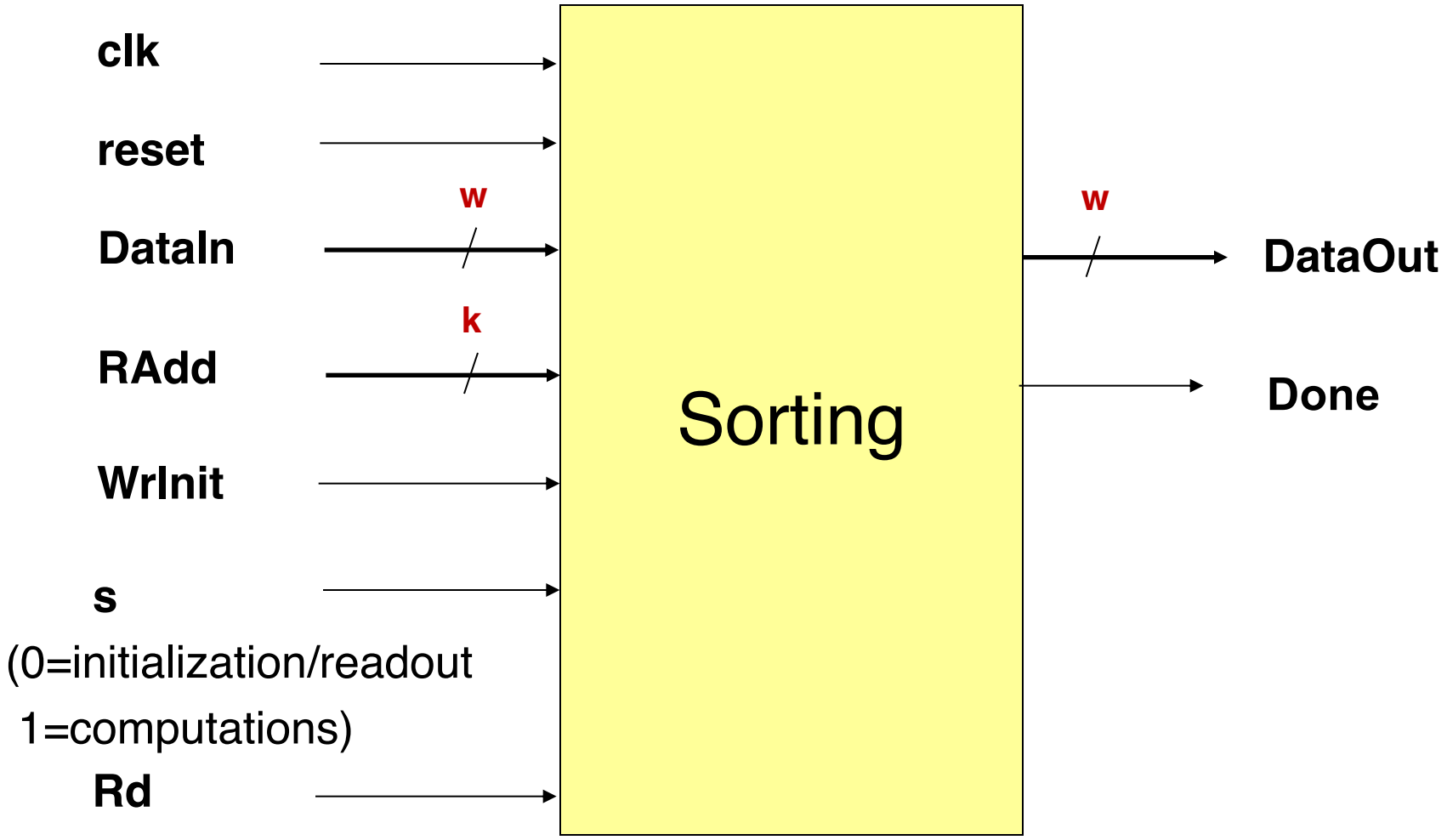
Software Design:

S1. Software driver

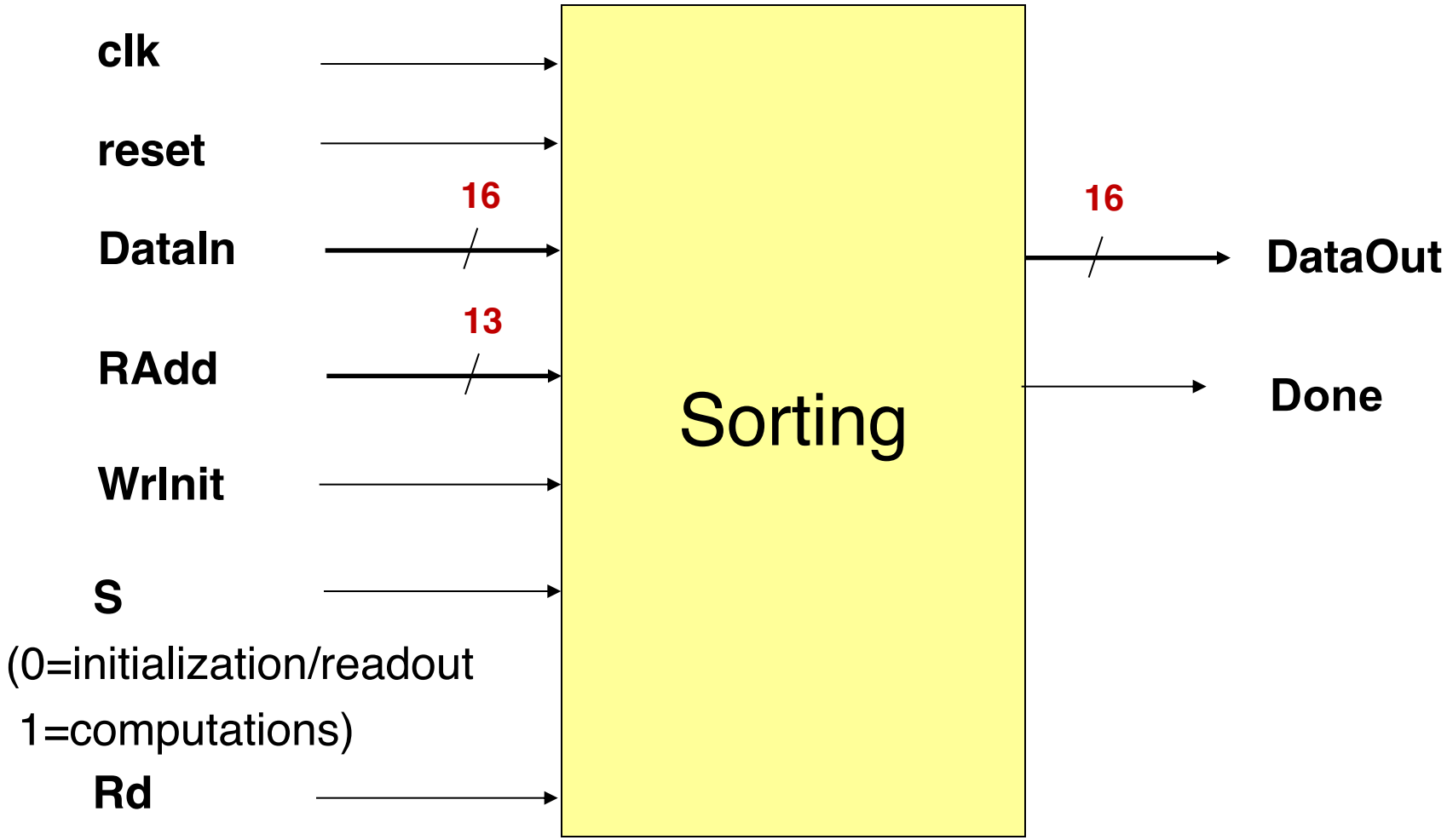
* declarations (.h) * implementations (.cpp) * testing

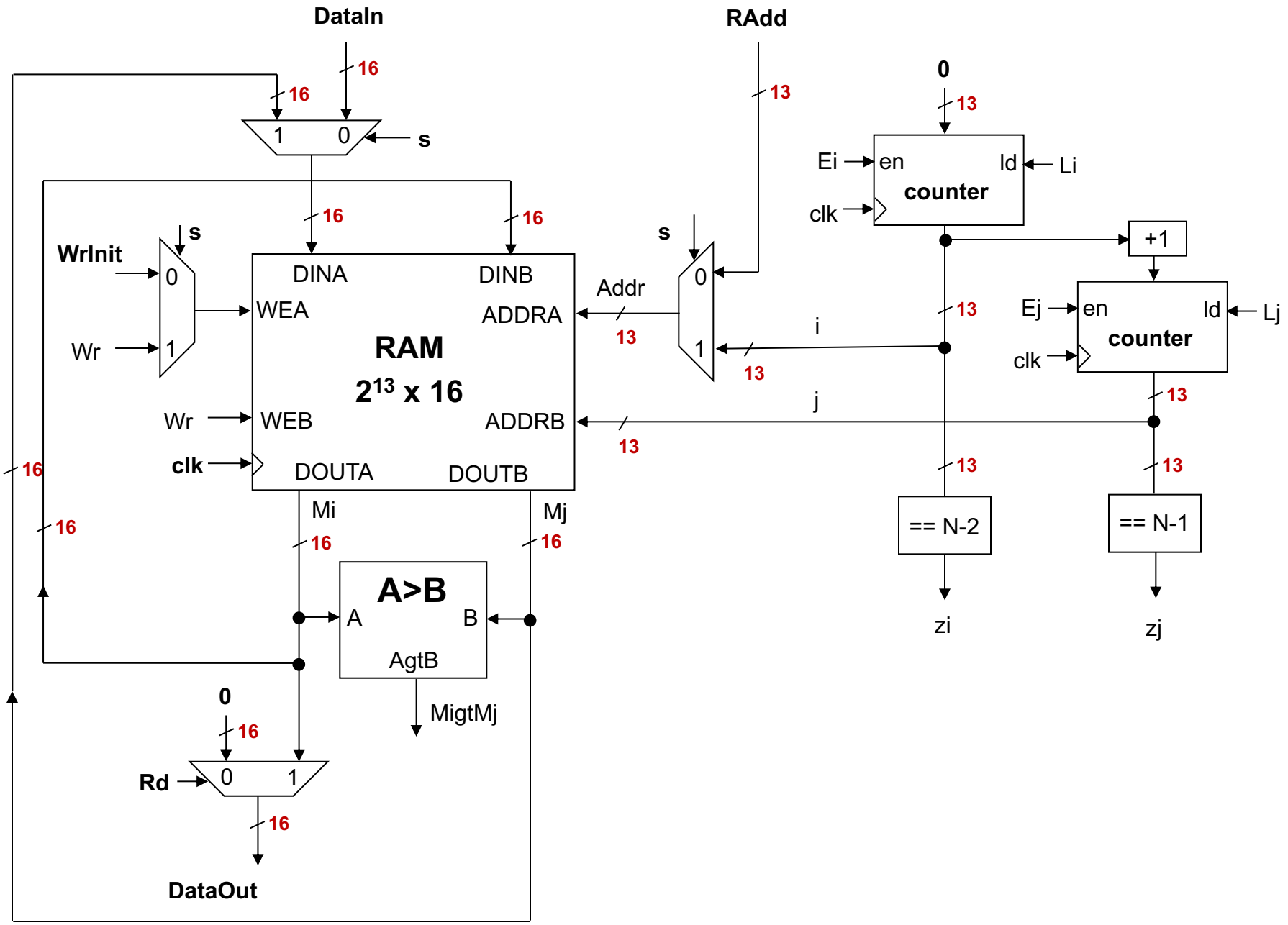
S2. Application based on functions of custom and standard cores

Sorting


clk
reset
w
DataIn
k
RAdd
WrInit
s
(0=initialization/readout
1=computations)
Rd
Sorting
w
DataOut
Done

Sorting


clk
reset
16
DataIn
13
RAdd
WrInit
S
(0=initialization/readout
1=computations)
Rd
Sorting
16
DataOut
Done


DataIn
RAdd
16
13
0
1
s
Ei
en
ld
Li
clk
counter
WrInit
DINA
DINB
WEA
ADDRA
RAM
2^{13} x 16
Wr
WEB
ADDRB
clk
DOUTA
DOUTB
Addr
i
j
+1
Ej
Lj
Mi
Mj
A>B
A
B
AgtB
MigtMj
== N-2
== N-1
zi
zj
Rd
DataOut

Developing a software/hardware implementation using an FPro system

Conceptual Design:

C1. Software/hardware partitioning

C2. I/O register map of the IP core

Hardware Design:

H1. Basic circuit performing the required functionality

* datapath * controller * top-level * functional verification

H2. A wrapper matching the interface of an MMIO core

*** design adjustments * coding * functional verification**

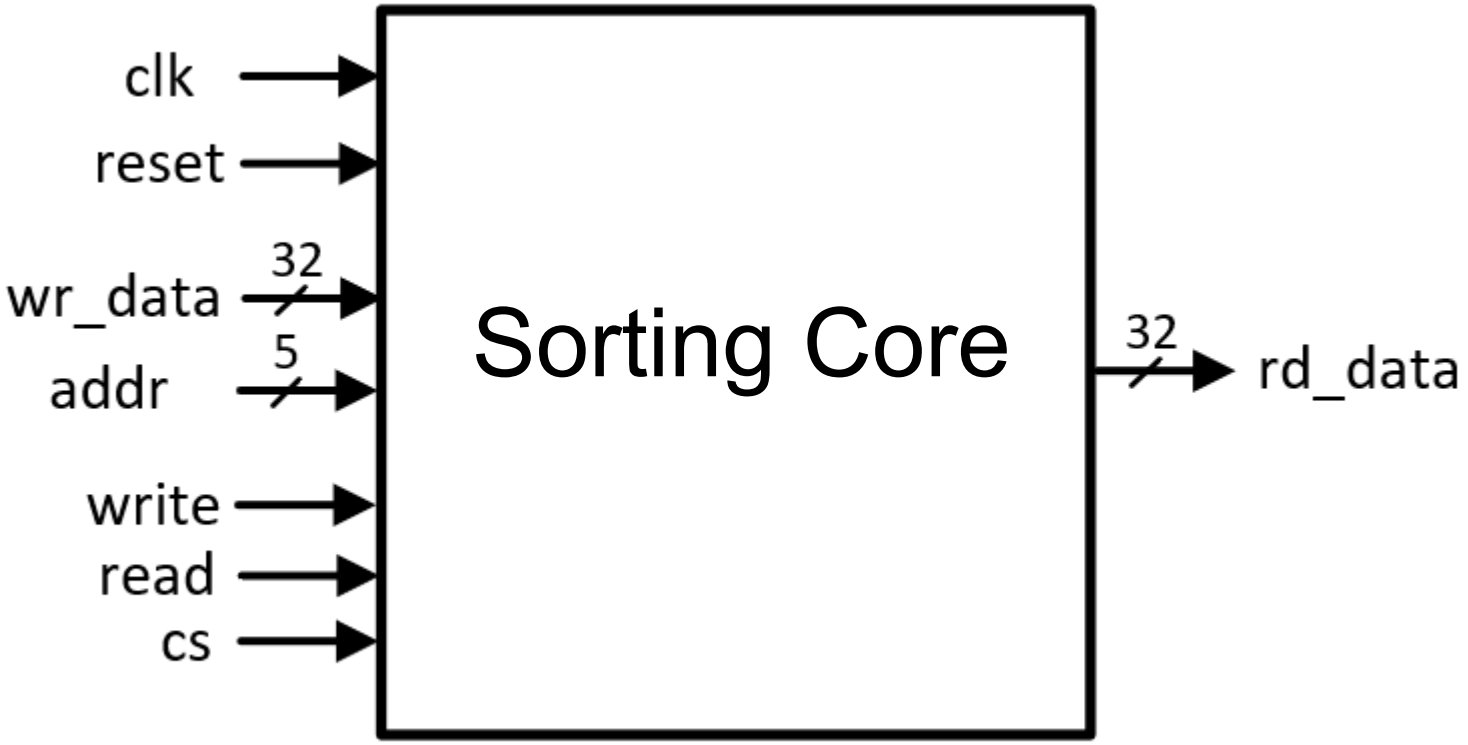
Software Design:

S1. Software driver

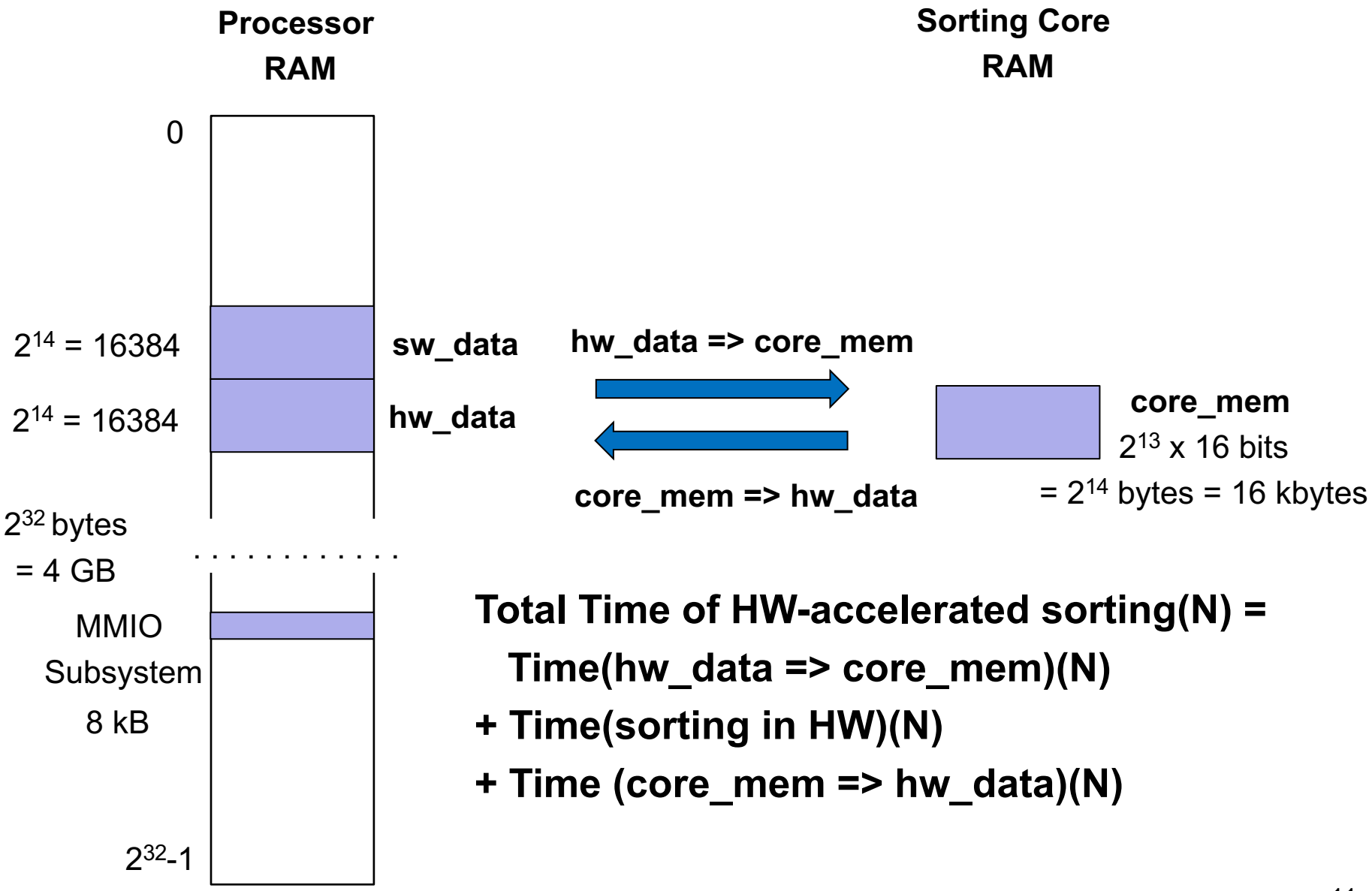
* declarations (.h) * implementations (.cpp) * testing

S2. Application based on functions of custom and standard cores

Interface of every MMIO Core



Data Transfer



Total Time of HW-accelerated sorting(N) =

Time(hw_data => core_mem)(N)

+ Time(sorting in HW)(N)

+ Time (core_mem => hw_data)(N)

Artix-7 FPGA Family

Device	Logic Cells	Configurable Logic Blocks (CLBs)		DSP48E1 Slices[2]	Block RAM Blocks[3]			CMTs[4]	PCIe[5]	GTPs	XADC Blocks	Total I/O Banks[6]	Max User I/O[7]
		Slices[1]	Max Distributed RAM (Kb)		18 Kb	36 Kb	Max (Kb)						
XC7A15T	16,640	2,600	200	45	50	25	900	5	1	4	1	5	250
XC7A35T	33,280	5,200	400	90	100	50	1,800	5	1	4	1	5	250
XC7A50T	52,160	8,150	600	120	150	75	2,700	5	1	4	1	5	250
XC7A75T	75,520	11,800	892	180	210	105	3,780	6	1	8	1	6	300
XC7A100T	101,440	15,850	1,188	240	270	135	4,860	6	1	8	1	6	300
XC7A200T	215,360	33,650	2,888	740	730	365	13,140	10	1	16	1	10	500

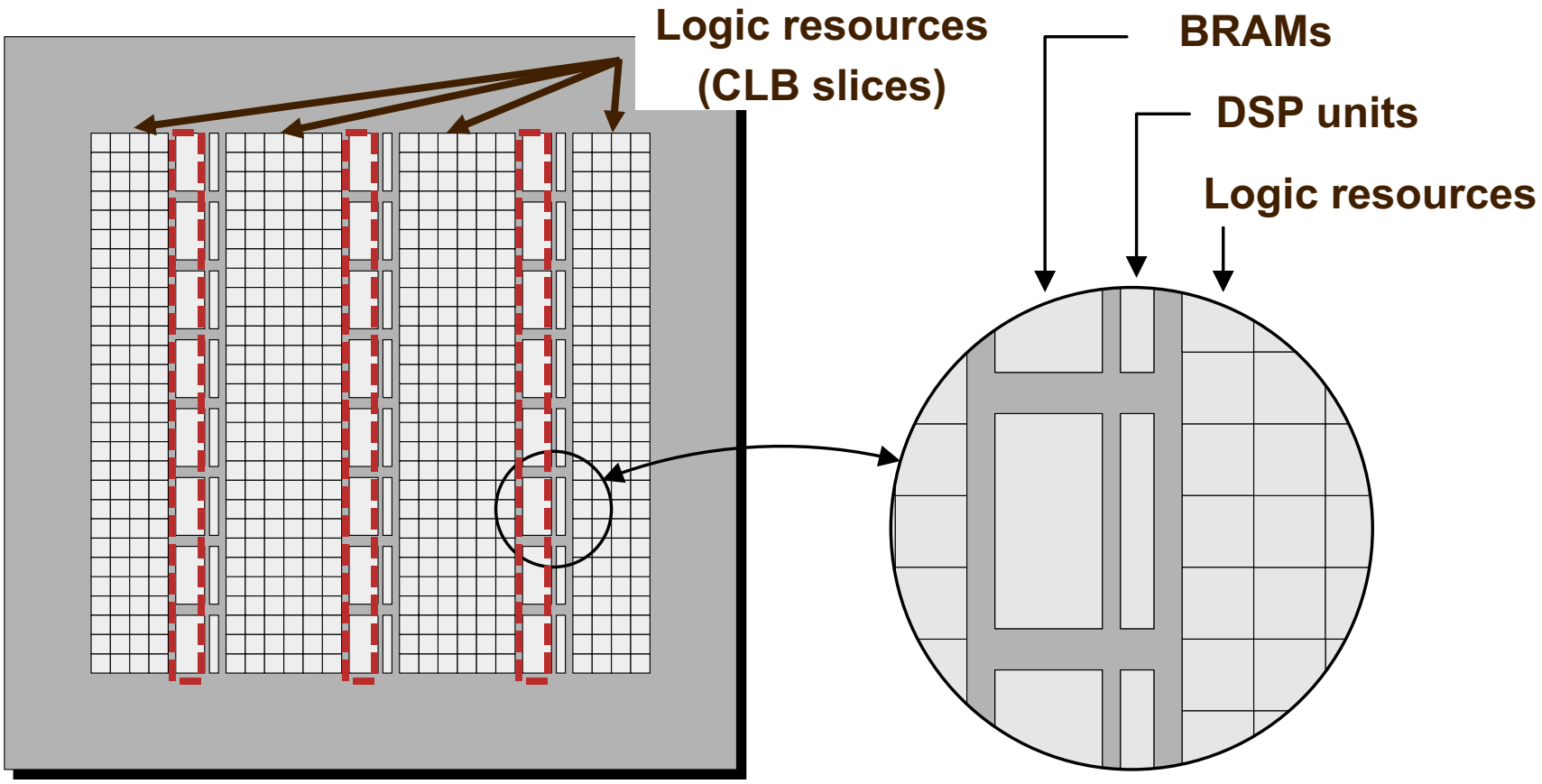
400 kbits = 50 kbytes

1800 kbits = 225 kbytes

- 128 kbytes (used by MicroBlaze)

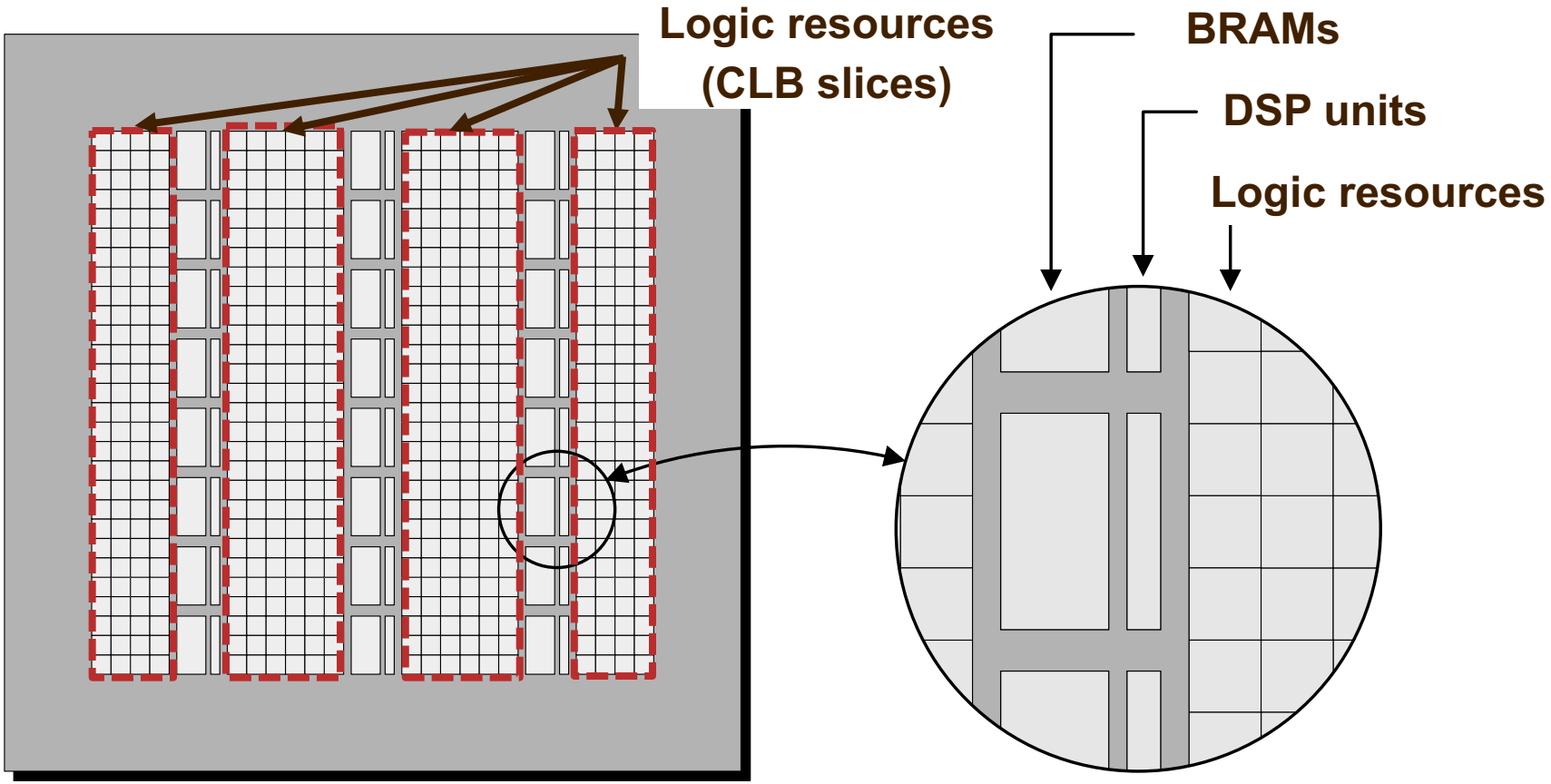
97 kbytes

Location of Block RAMs



Graphics based on The Design Warrior's Guide to FPGAs
Devices, Tools, and Flows. ISBN 0750676043

Location of Distributed RAM



Graphics based on The Design Warrior's Guide to FPGAs
Devices, Tools, and Flows. ISBN 0750676043

Developing a software/hardware implementation using an FPro system

Conceptual Design:

C1. Software/hardware partitioning

C2. I/O register map of the IP core

Hardware Design:

H1. Basic circuit performing the required functionality

* datapath * controller * top-level * functional verification

H2. A wrapper matching the interface of an MMIO core

* design adjustments * coding * functional verification

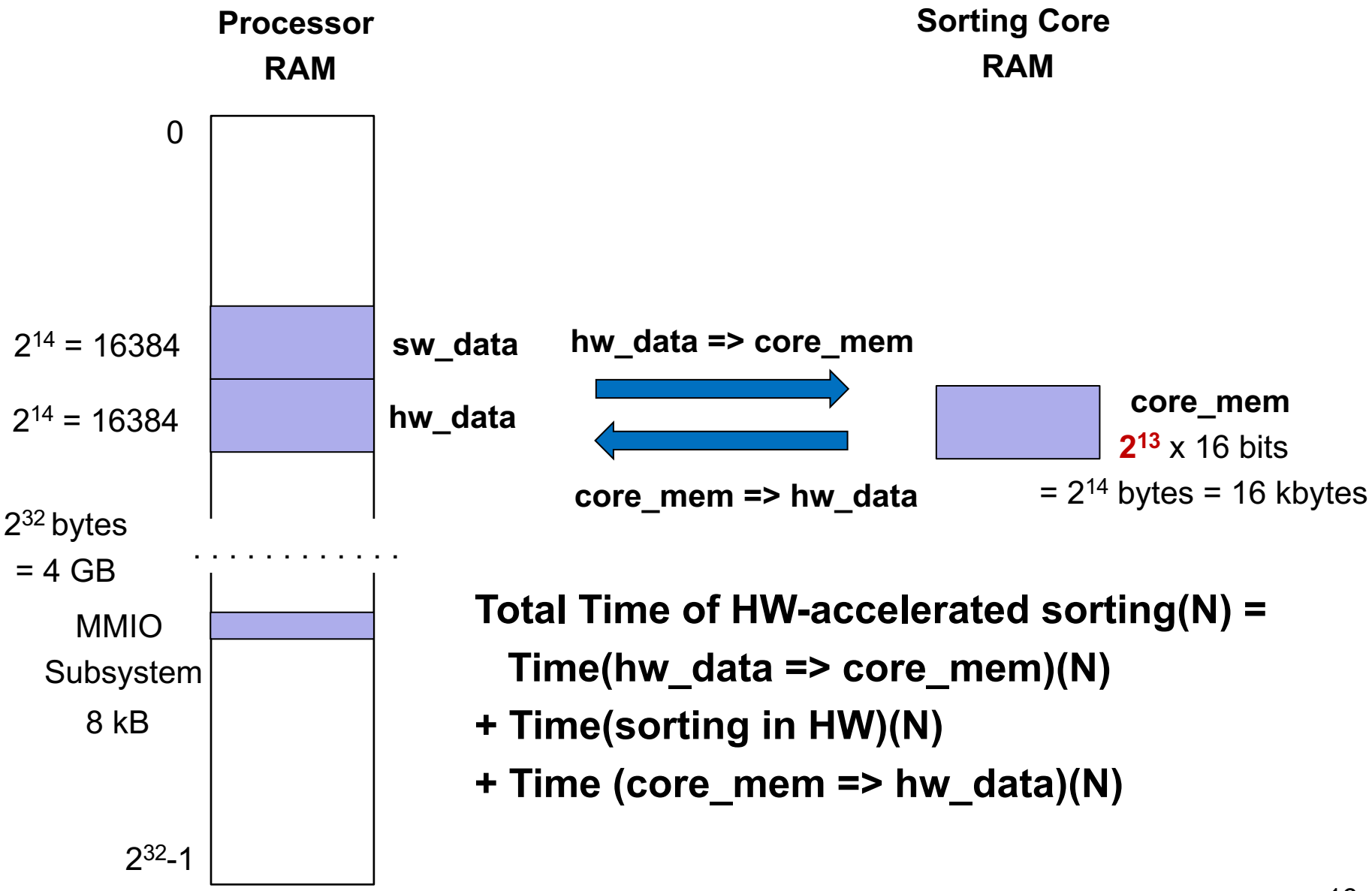
Software Design:

S1. Software driver

* declarations (.h) * implementations (.cpp) * testing

S2. Application based on functions of custom and standard cores

Data Transfer



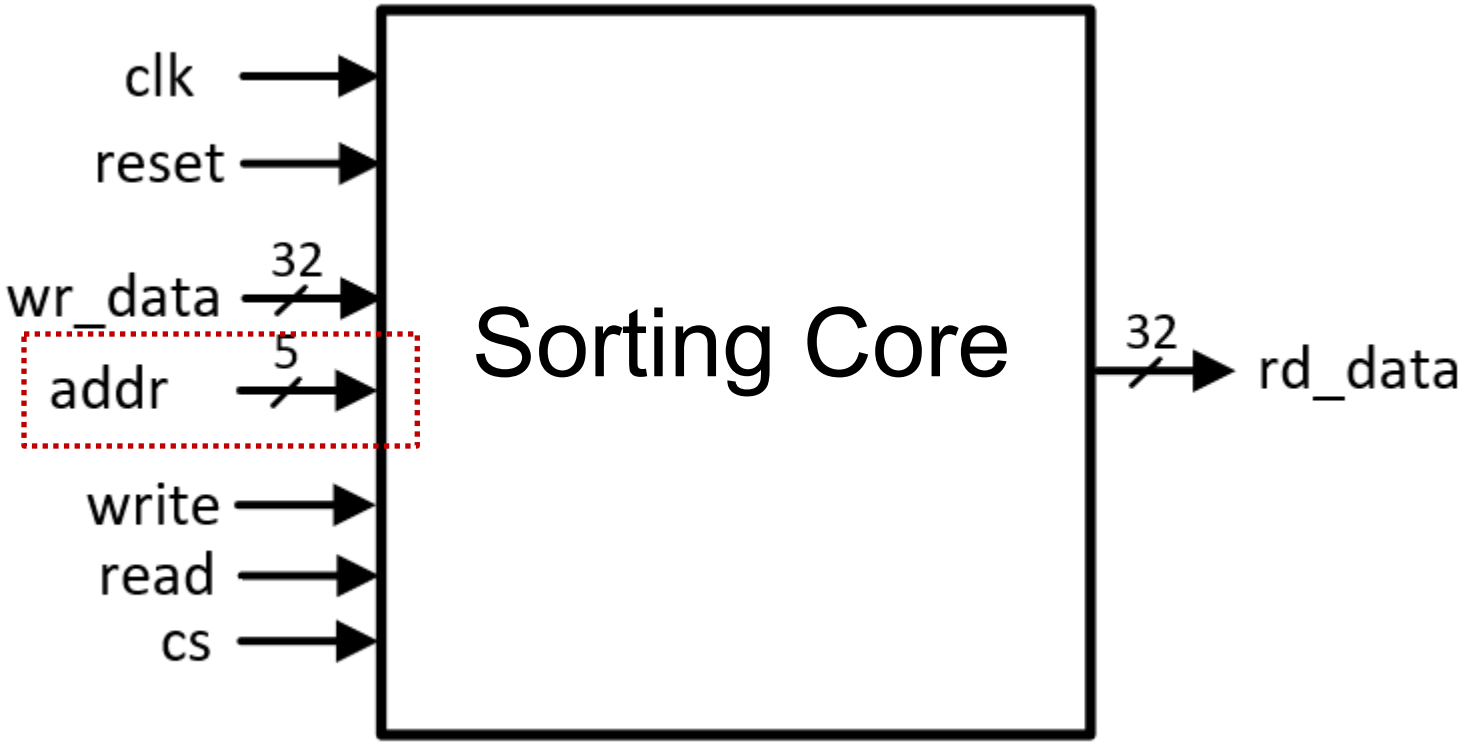
Total Time of HW-accelerated sorting(N) =

Time(hw_data => core_mem)(N)

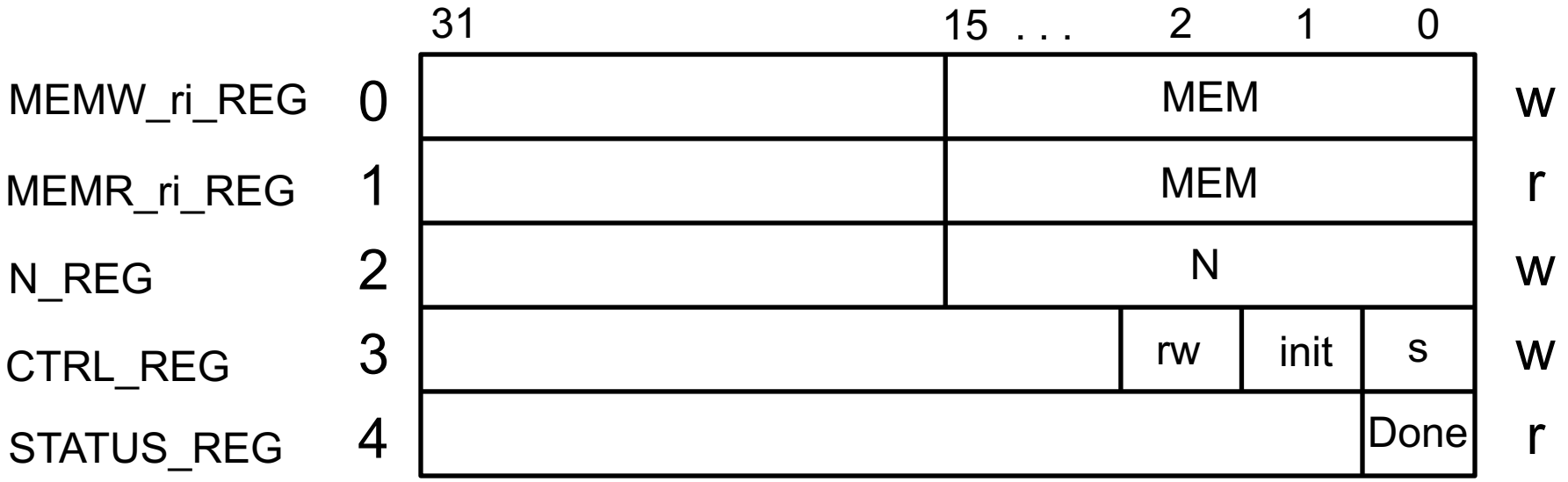
+ Time(sorting in HW)(N)

+ Time (core_mem => hw_data)(N)

Interface of every MMIO Core



I/O Register Map of the Sorting Core



Writing to MEMW_ri_REG : Writing to MEM[ri] & ri++

Reading from MEMR_ri_REG : Reading from MEM[ri] & ri++

Writing to NREG_REG : Initializing N

Reading from STATUS_REG : Reading Done

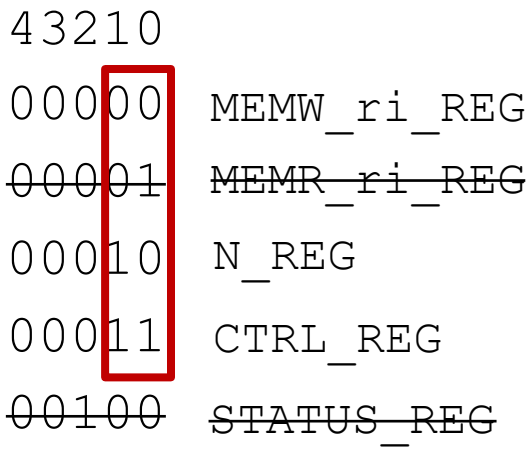
Writing to CRTL_REG:

rw	init	s	
0	0	1	Set s=1 : computations
0	0	0	Set s=0 : initialization/readout
1	1	0	Set ri=0 and WrInit_r=1 : start initialization
0	1	0	Set ri=0 and Rd_r=1 : start readout

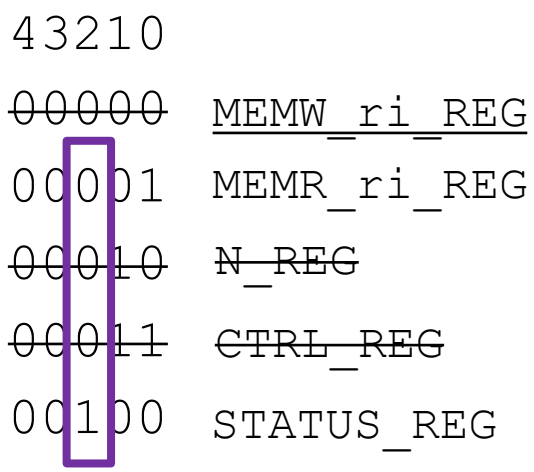
Address Decoding

```
43210
00000  MEMW_ri_REG
00001  MEMR_ri_REG
00010  N_REG
00011  CTRL_REG
00100  STATUS_REG
```

Simplified Address Decoding for Writing



Simplified Address Decoding for Reading



Developing a software/hardware implementation using an FPro system

Conceptual Design:

C1. Software/hardware partitioning

C2. I/O register map of the IP core

Hardware Design:

H1. Basic circuit performing the required functionality

* datapath * controller * top-level * functional verification

H2. A wrapper matching the interface of an MMIO core

*** design adjustments * coding * functional verification**

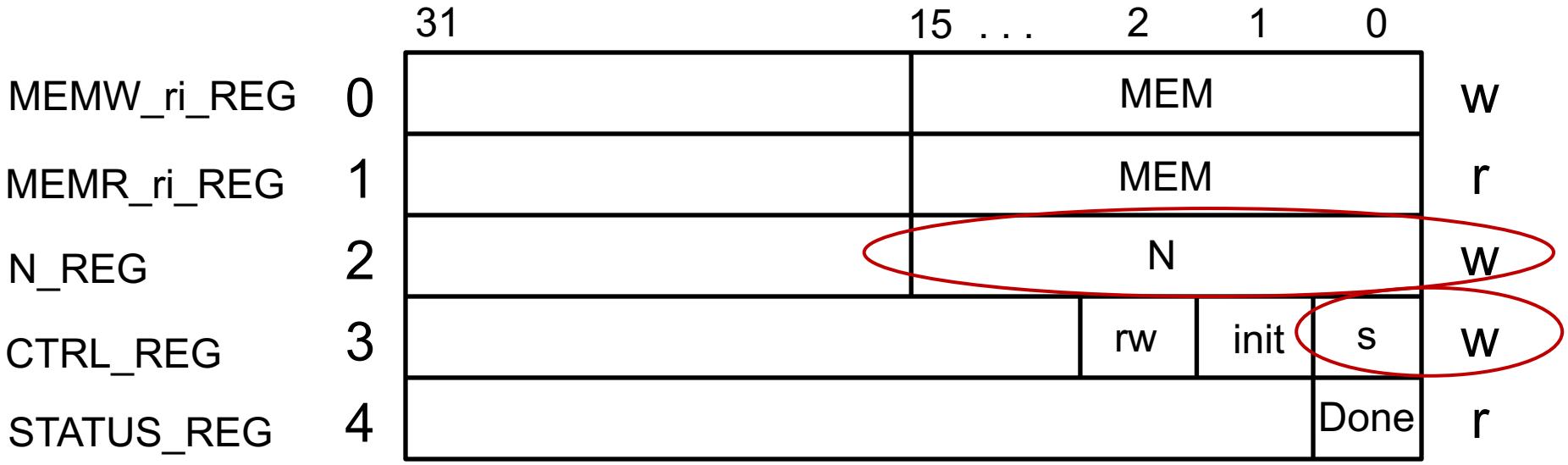
Software Design:

S1. Software driver

* declarations (.h) * implementations (.cpp) * testing

S2. Application based on functions of custom and standard cores

IO Register Map of the Sorting Core



Writing to MEMW_ri_REG : Writing to MEM[ri] & ri++

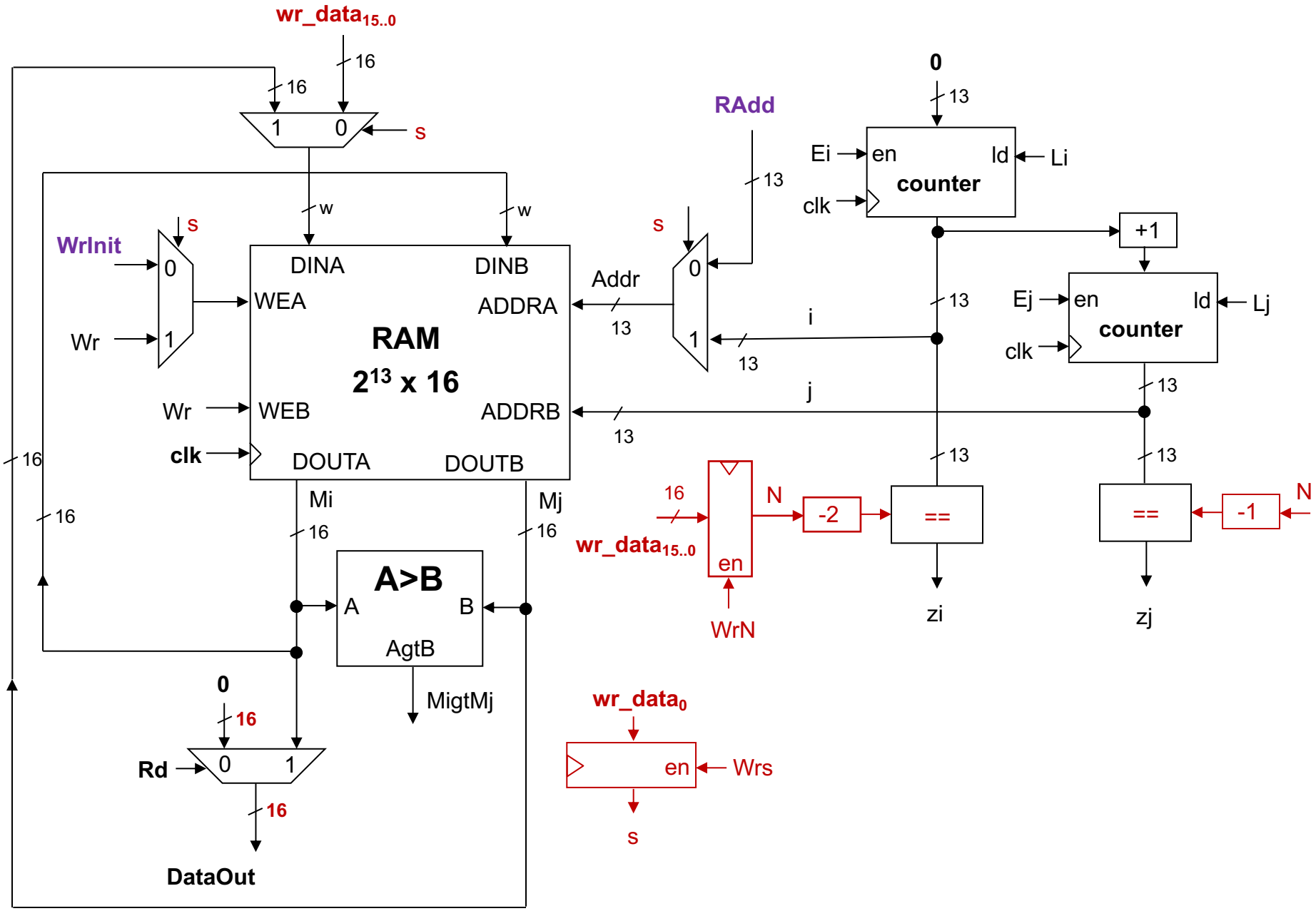
Reading from MEMR_ri_REG : Reading from MEM[ri] & ri++

Writing to NREG_REG : Initializing N

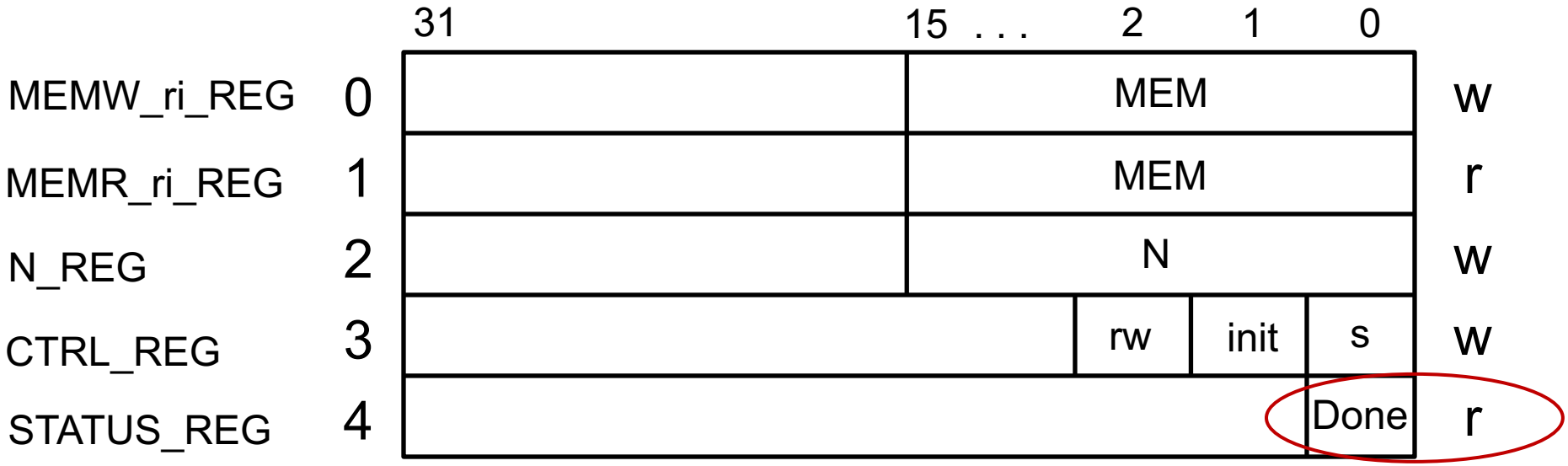
Reading from STATUS_REG : Reading Done

Writing to CRTL_REG:

rw	init	s	
0	0	1	Set s=1 : computations
0	0	0	Set s=0 : initialization/readout
1	1	0	Set ri=0 and WrInit_r=1 : start initialization
0	1	0	Set ri=0 and Rd_r=1 : start readout


wr_data15..0
16
16
1
0
s
w
w
WrInit
s
0
1
Wr
DINA
DINB
WEA
ADDRA
RAM
2^13 x 16
Wr
WEB
ADDRB
clk
DOUTA
DOUTB
16
16
Mi
Mj
16
16
A>B
A
B
AgtB
MigtMj
0
16
Rd
0
1
16
DataOut
RAdd
13
s
0
1
Addr
13
13
i
13
j
13
0
13
Ei
en
ld
Li
clk
counter
+1
Ej
en
ld
Lj
clk
counter
13
13
16
N
-2
==
zi
13
13
==
-1
N
zj
wr_data15..0
en
WrN
wr_data0
en
Wrs
s

IO Register Map of the Sorting Core



Writing to MEMW_ri_REG : Writing to MEM[ri] & ri++

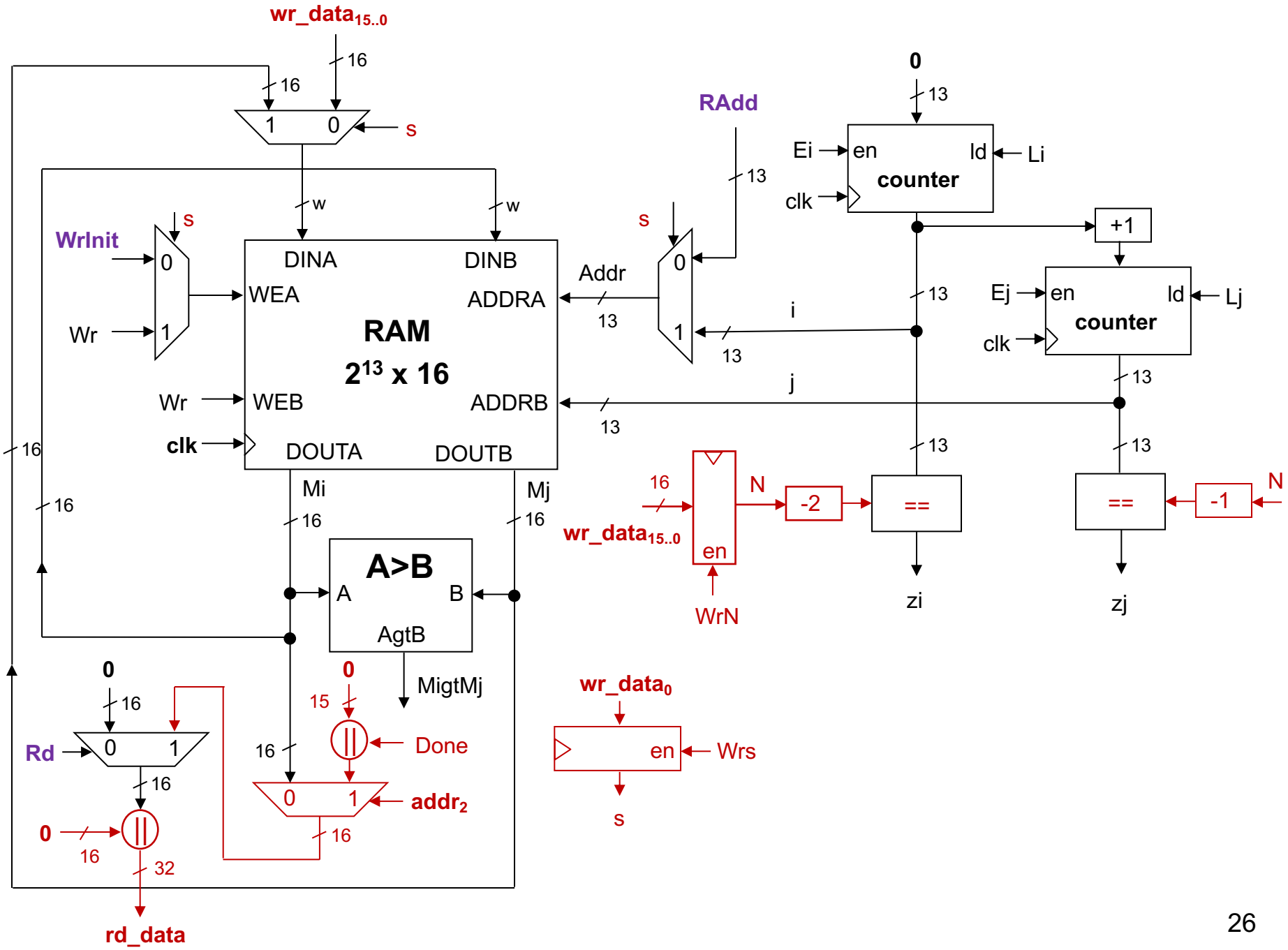
Reading from MEMR_ri_REG : Reading from MEM[ri] & ri++

Writing to NREG_REG : Initializing N

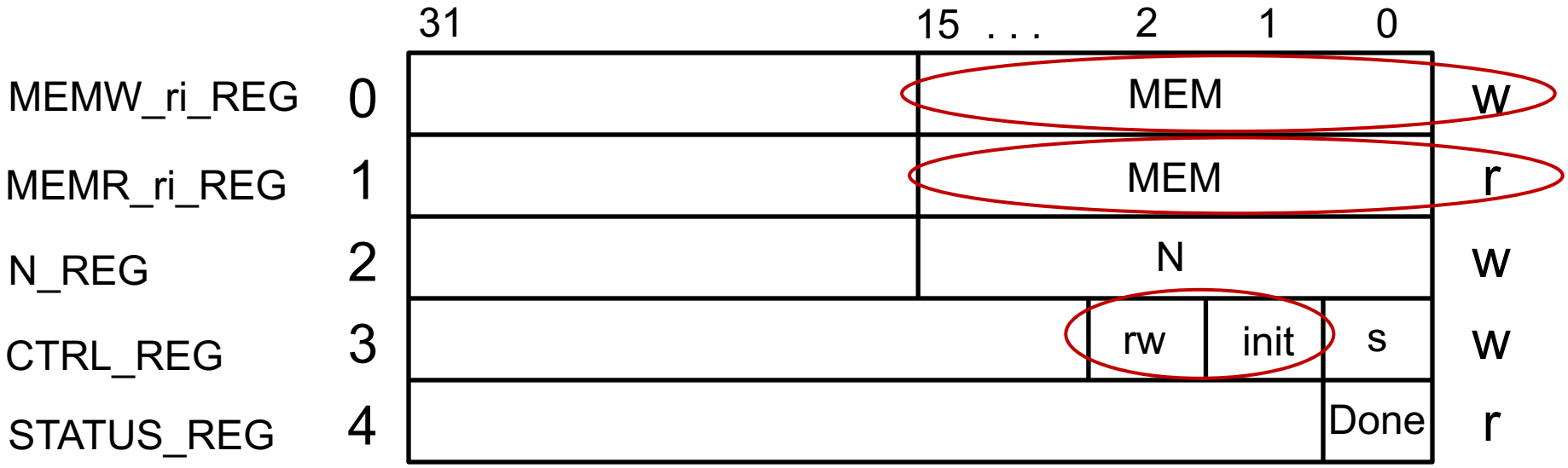
Reading from STATUS_REG : Reading Done

Writing to CRTL_REG:

rw	init	s	
0	0	1	Set s=1 : computations
0	0	0	Set s=0 : initialization/readout
1	1	0	Set ri=0 and WrInit_r=1 : start initialization
0	1	0	Set ri=0 and Rd_r=1 : start readout


wr_data15..0
16
16
1
0
s
w
w
WrInit
s
0
1
Wr
DINA
DINB
WEA
ADDRA
RAM
213 x 16
Wr
WEB
ADDRB
clk
DOUTA
DOUTB
16
16
Mi
Mj
16
16
A>B
A
B
AgtB
MigtMj
0
16
Rd
0
1
16
0
16
32
0
15
Done
16
0
1
addr2
16
rd_data
RAdd
13
s
0
1
Addr
13
13
i
13
j
13
0
13
Ei
en
ld
Li
counter
clk
+1
Ej
en
ld
Lj
counter
clk
13
13
16
N
-2
==
13
13
N
-1
==
wr_data15..0
en
WrN
zi
zj
wr_data0
en
Wrs
s

IO Register Map of the Sorting Core



Writing to MEMW_ri_REG : Writing to MEM[ri] & ri++

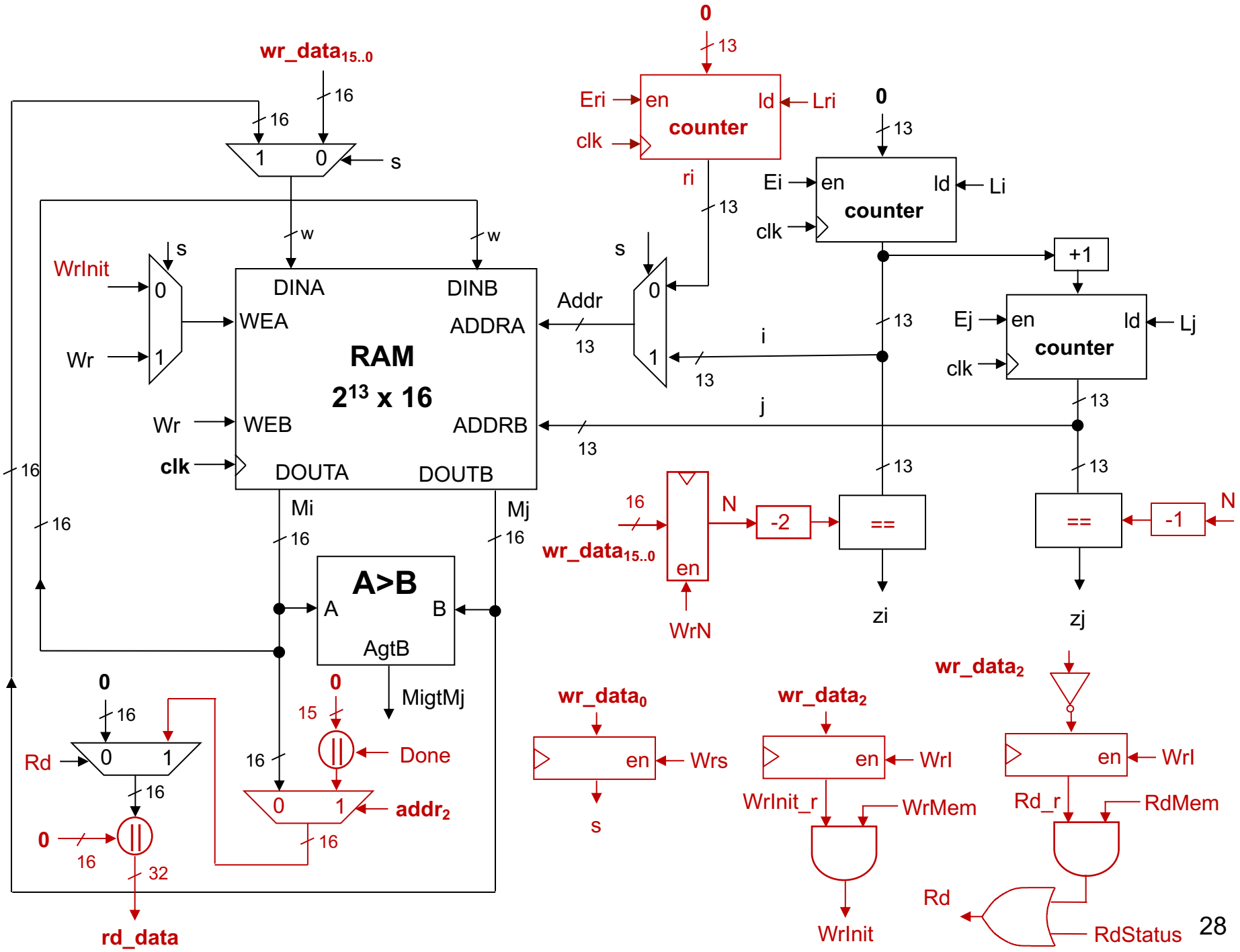
Reading from MEMR_ri_REG : Reading from MEM[ri] & ri++

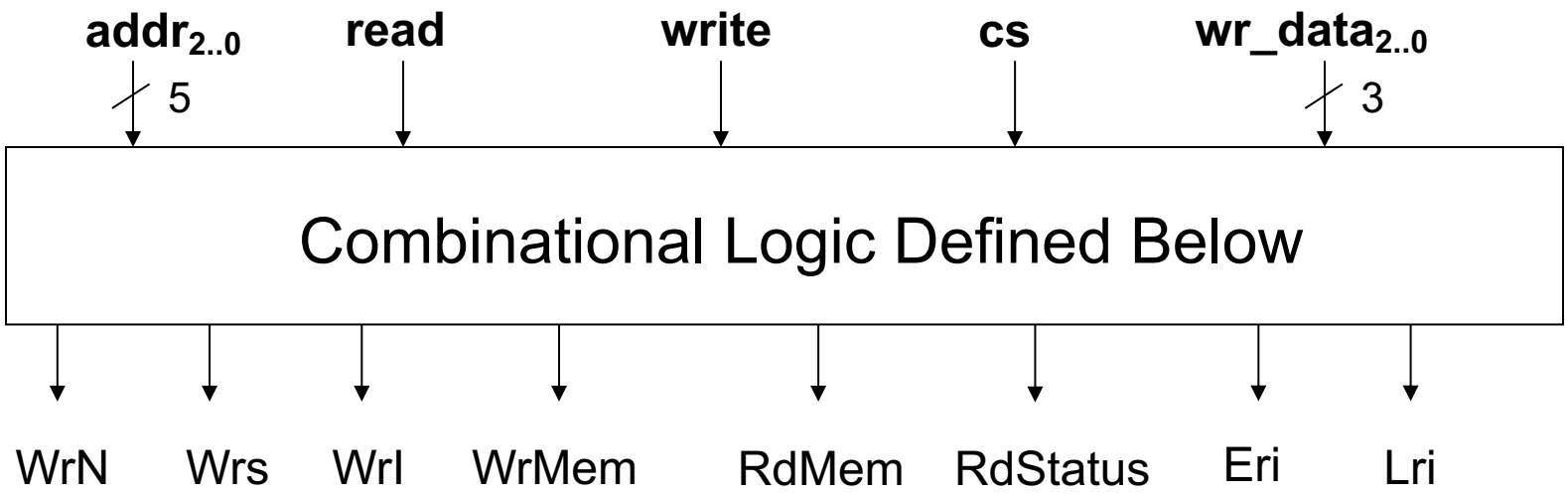
Writing to NREG_REG : Initializing N

Reading from STATUS_REG : Reading Done

Writing to CRTL_REG:

rw	init	s	
0	0	1	Set s=1 : computations
0	0	0	Set s=0 : initialization/readout
1	1	0	Set ri=0 and WrInit_r=1 : start initialization
0	1	0	Set ri=0 and Rd_r=1 : start readout


wr_data15..0
16
16
1
0
s
w
w
WrInit
s
0
1
Wr
DINA
DINB
WEA
ADDRA
RAM
2^13 x 16
Wr
WEB
ADDRB
clk
DOUTA
DOUTB
16
16
Mi
Mj
16
16
A>B
A
B
AgtB
MigtMj
0
16
Rd
0
1
16
0
16
32
rd_data
0
15
Done
16
0
1
addr2
16
0
13
Eri
en
ld
Lri
clk
counter
ri
13
s
0
1
Addr
13
i
13
13
0
13
Ei
en
ld
Li
clk
counter
+1
Ej
en
ld
Lj
clk
counter
13
j
13
13
13
16
wr_data15..0
N
-2
==
zi
WrN
en
13
==
-1
N
zj
wr_data0
en
Wrs
s
wr_data2
en
WrI
WrInit_r
WrMem
WrInit
wr_data2
en
WrI
Rd_r
RdMem
Rd
RdStatus

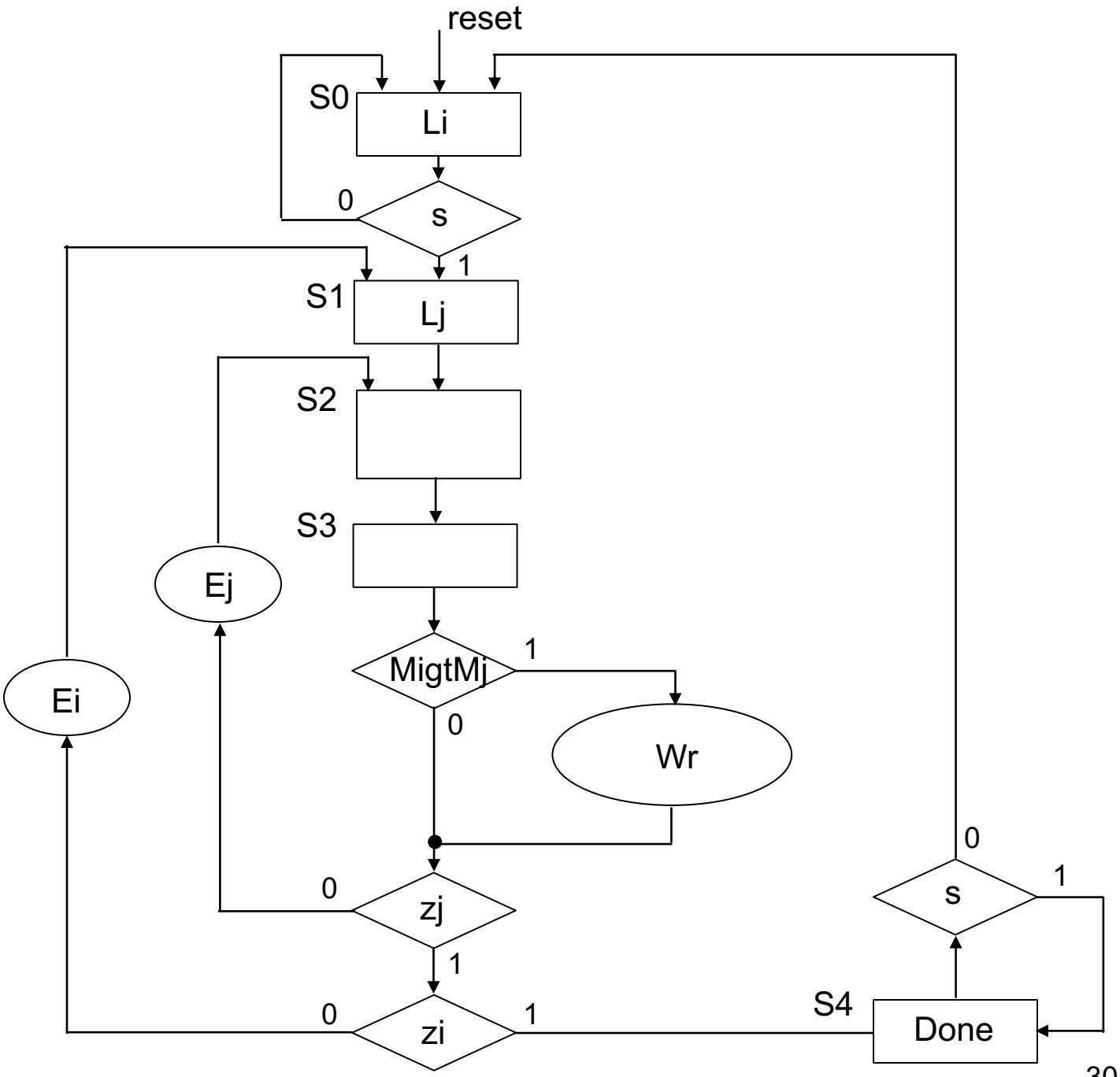


Output	cs	write	read	addr$_{2..0}$	wr_data$_{2..0}$
WrN=1	1	1	0	010	−−−
Wrs=1	1	1	0	011	−0−
Wrl=1	1	1	0	011	−10
WrMem=1	1	1	0	000	−−−
RdMem=1	1	0	1	001	−−−
RdStatus=1	1	0	1	100	−−−

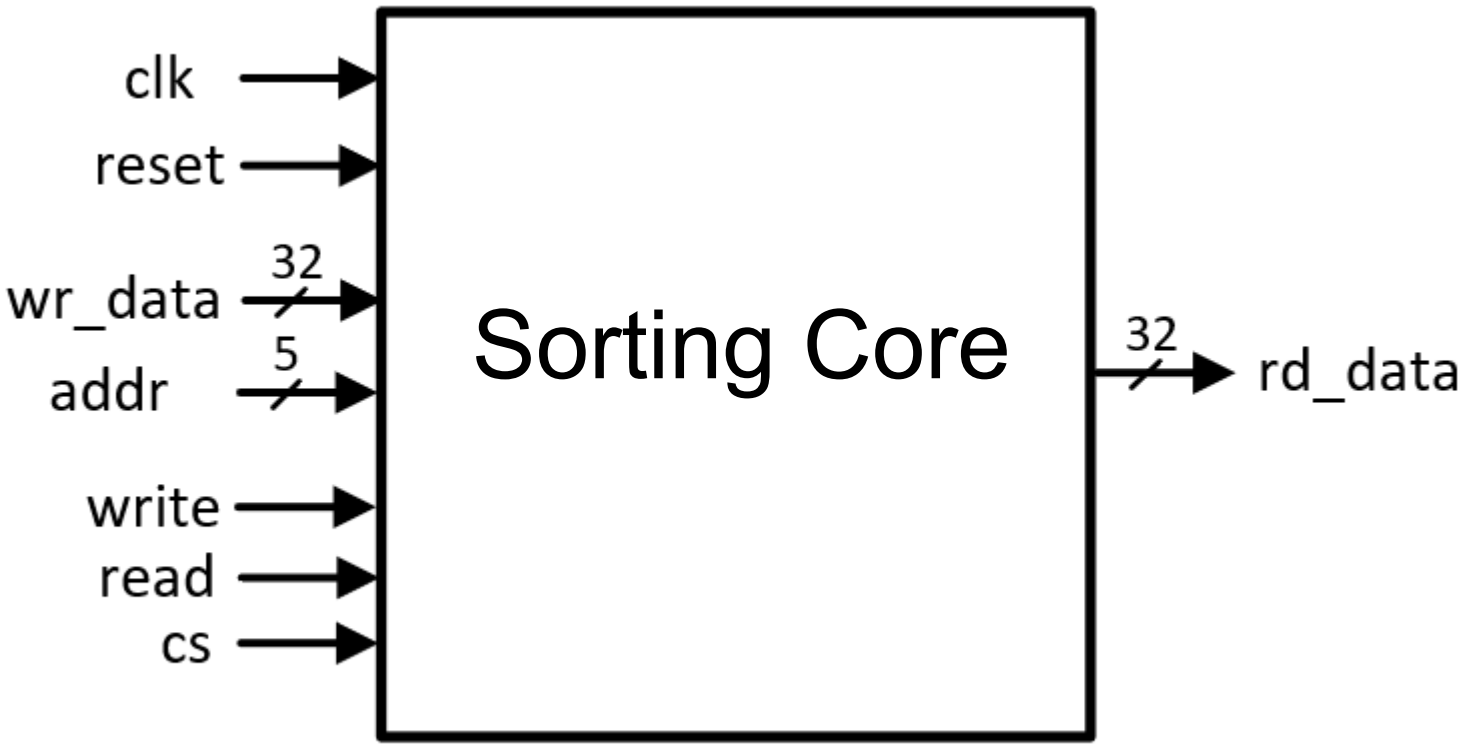
Eri = WrMem or RdMem

Lri = Wrl

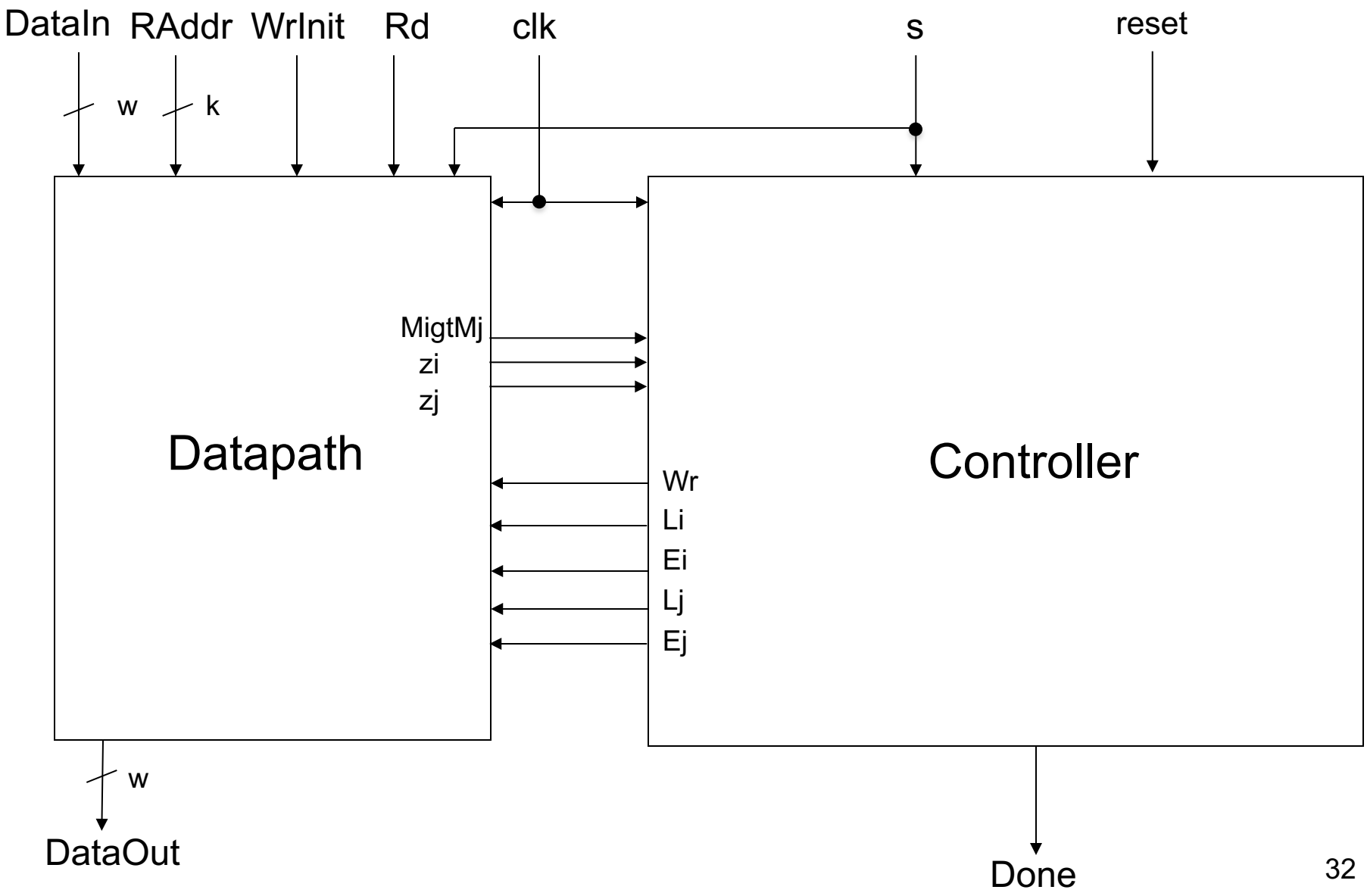
ASM Chart


reset
S0
Li
0
s
1
S1
Lj
S2
S3
Ej
MigtMj
1
0
Ei
Wr
0
zj
1
0
zi
1
S4
Done
0
s
1

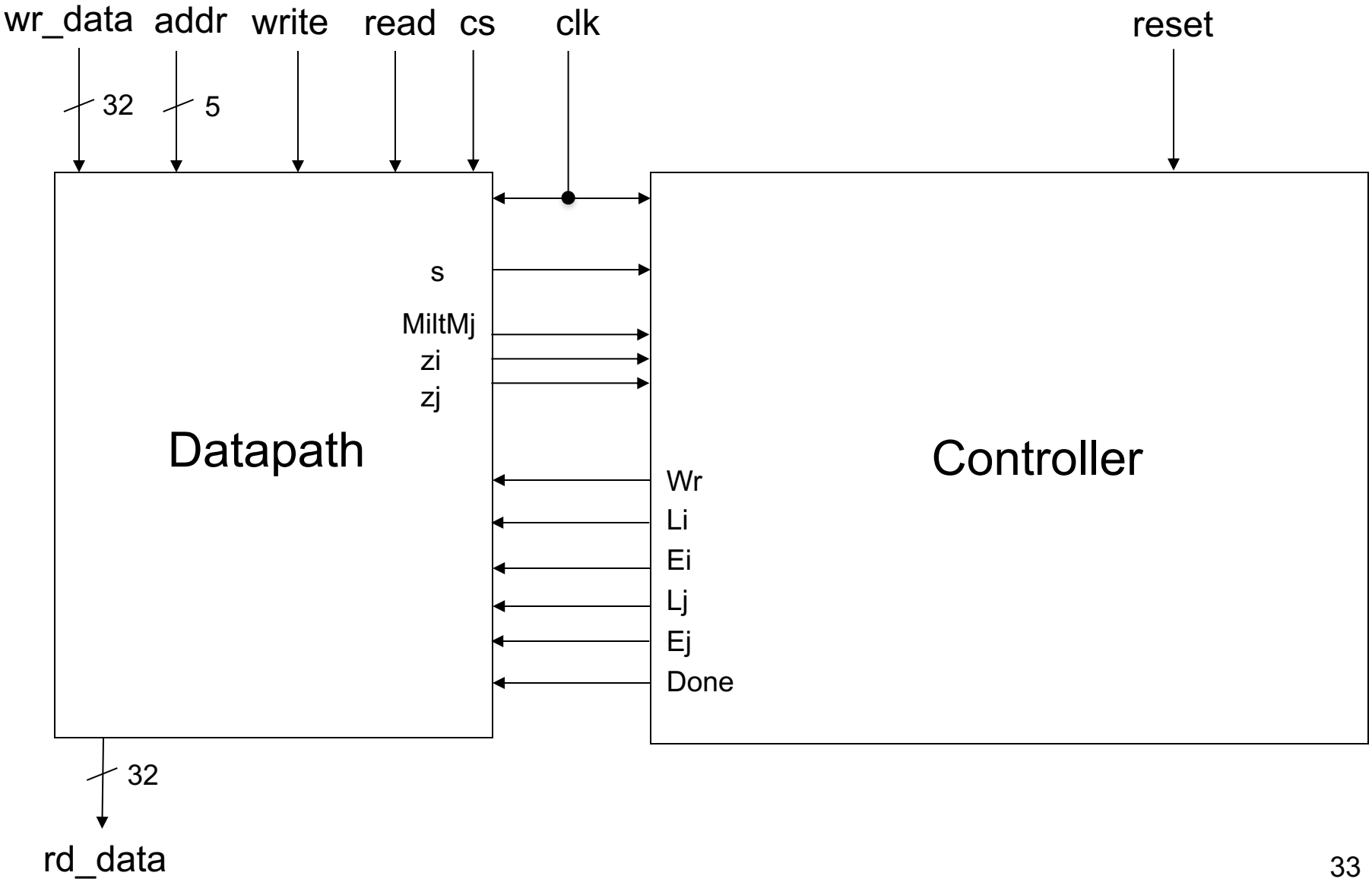
Interface

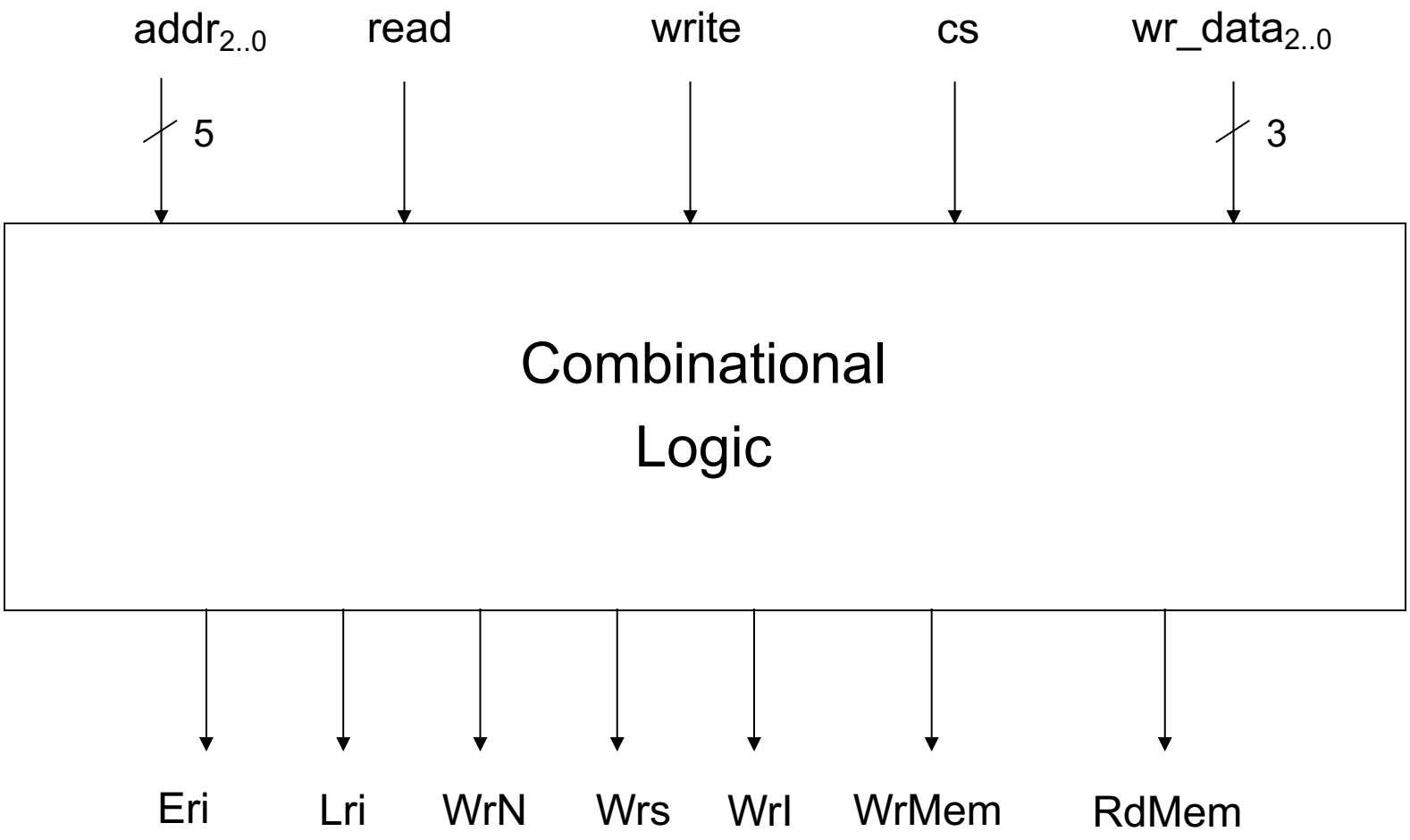

clk
reset
32
wr_data
5
addr
write
read
cs
Sorting Core
32
rd_data

Interface with the division into the Datapath and Controller w/o a Wrapper

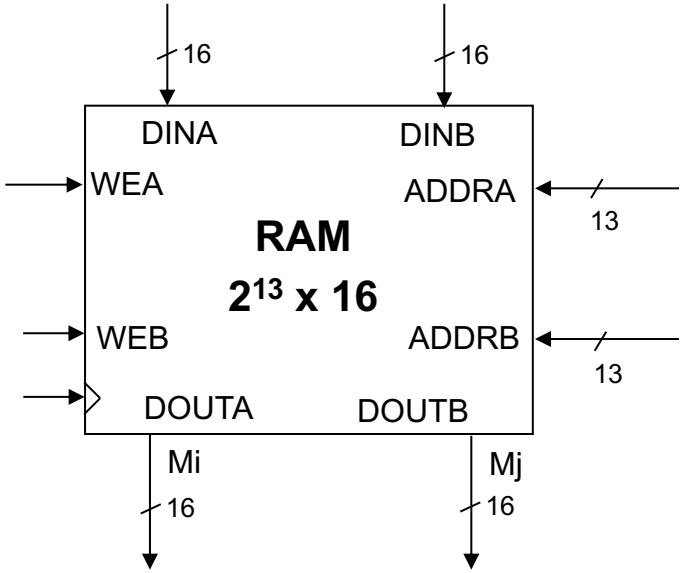


Interface with the division into the Datapath and Controller with a Wrapper




addr2..0
read
write
cs
wr_data2..0
5
3
Combinational
Logic
Eri
Lri
WrN
Wrs
Wrl
WrMem
RdMem

Dual-port RAM with Synchronous Read

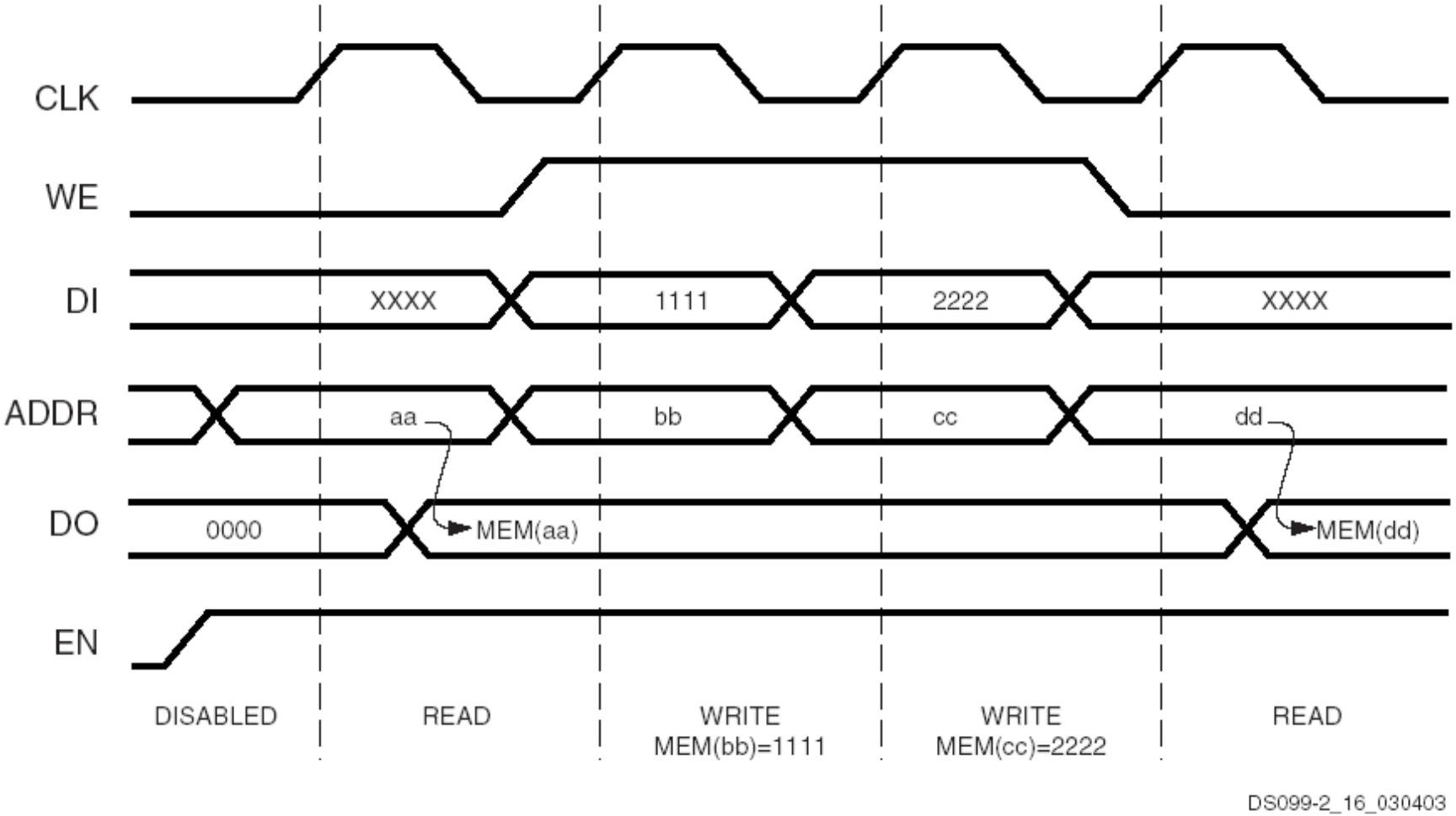


True Dual-Port Block RAM with Two Write Ports

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;

entity dpram is
    port(
     clk   : in  std_logic;
     wea   : in  std_logic;
     web   : in  std_logic;
     addra : in  std_logic_vector(12 downto 0);
     addrb : in  std_logic_vector(12 downto 0);
     dina   : in  std_logic_vector(15 downto 0);
     dinb   : in  std_logic_vector(15 downto 0);
     douta   : out std_logic_vector(15 downto 0);
     doutb   : out std_logic_vector(15 downto 0)
    );
end dpram;
```

Block RAM Waveforms – NO_CHANGE mode



Dual-Port Block RAM in NO_CHANGE mode (1)

```
architecture behavioral of dpram is
 type ram_type is array (0 to 2**13-1) of std_logic_vector(15 downto 0);
 shared variable RAM : ram_type := (others => (others => '0'));

begin
  process (clk)
  begin
    if rising_edge(clk) then
        if (wea = '1') then
           RAM(to_integer(unsigned(addra))) := dina;
        else
           douta <= RAM(to_integer(unsigned(addra)));
        end if;
    end if;
  end process;
```

Dual-Port Block RAM in NO_CHANGE mode (2)

```
process (clk)
  begin
    if rising_edge(clk) then
        if (web = '1') then
           RAM(to_integer(unsigned(addrb))) := dinb;
        else
           doutb <= RAM(to_integer(unsigned(addrb)));
        end if;
    end if;
  end process;

end behavioral;
```