ECE 448
Lecture 19a

I/O Register Map of an MMIO Core

Part 1:
Exact Address Decoding

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

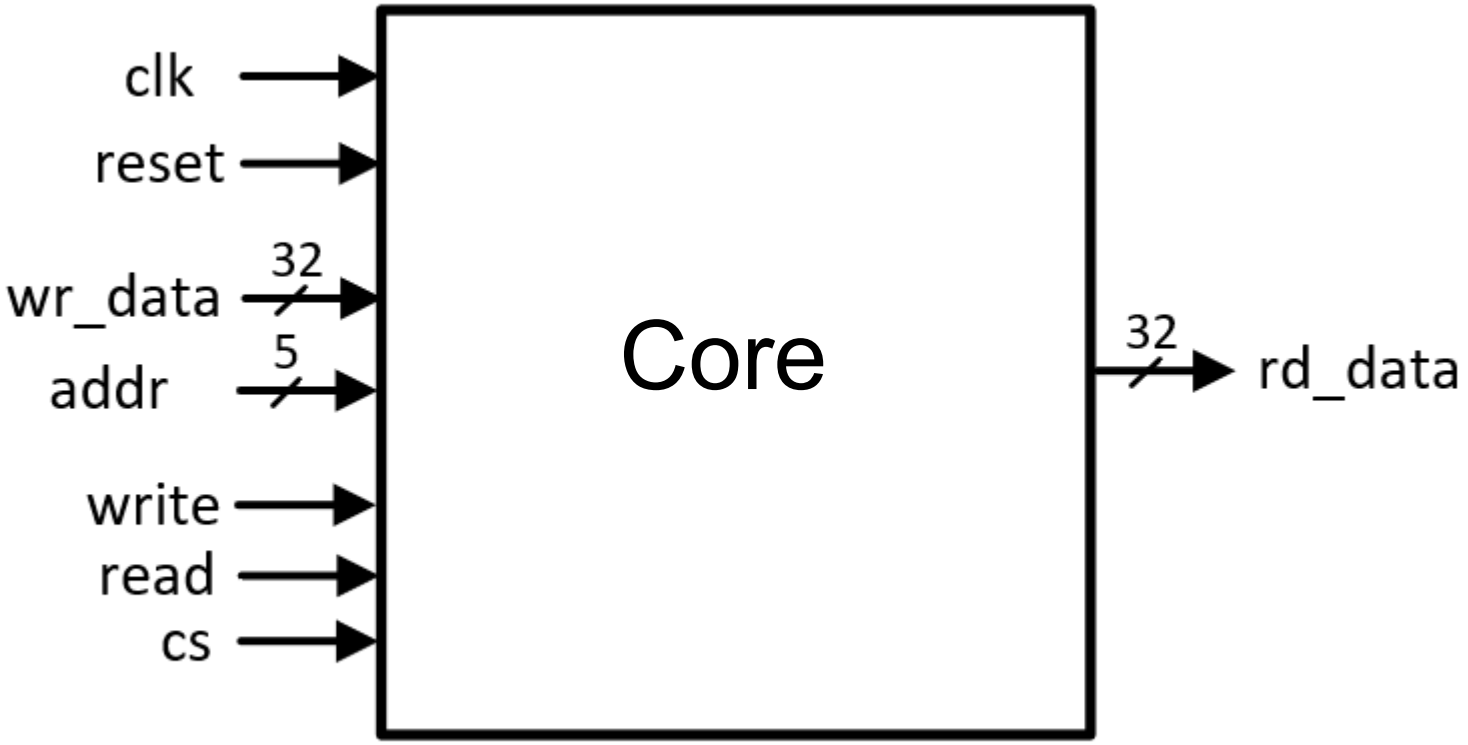
Software Design:

S1. Software driver

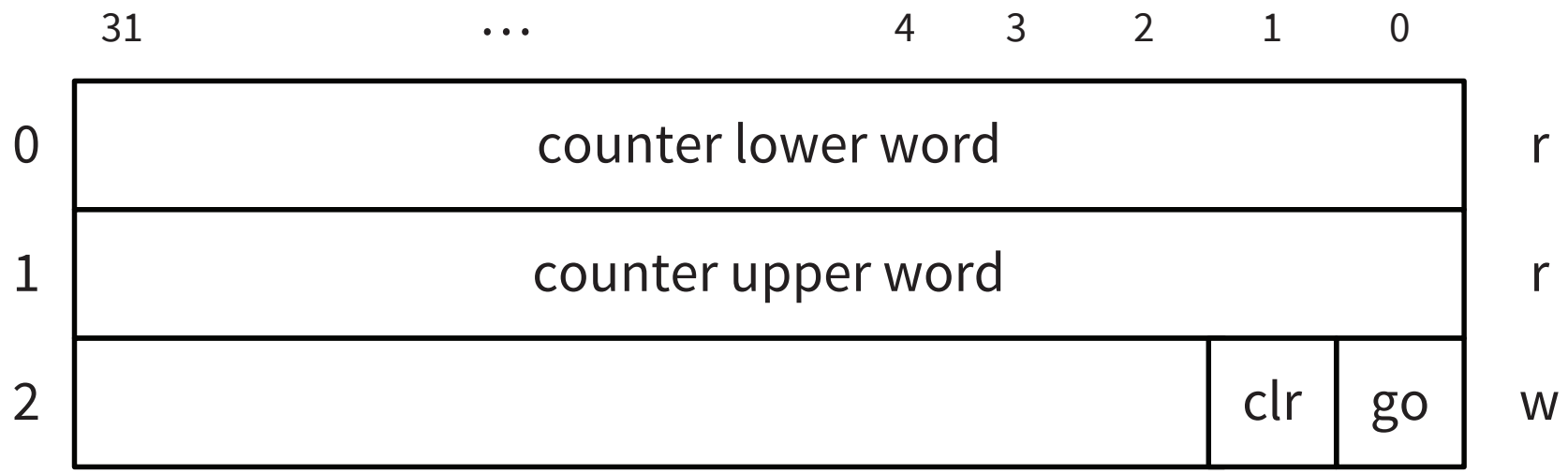
* declarations (.h) * implementations (.cpp) * testing

S2. Application based on functions of custom and standard cores

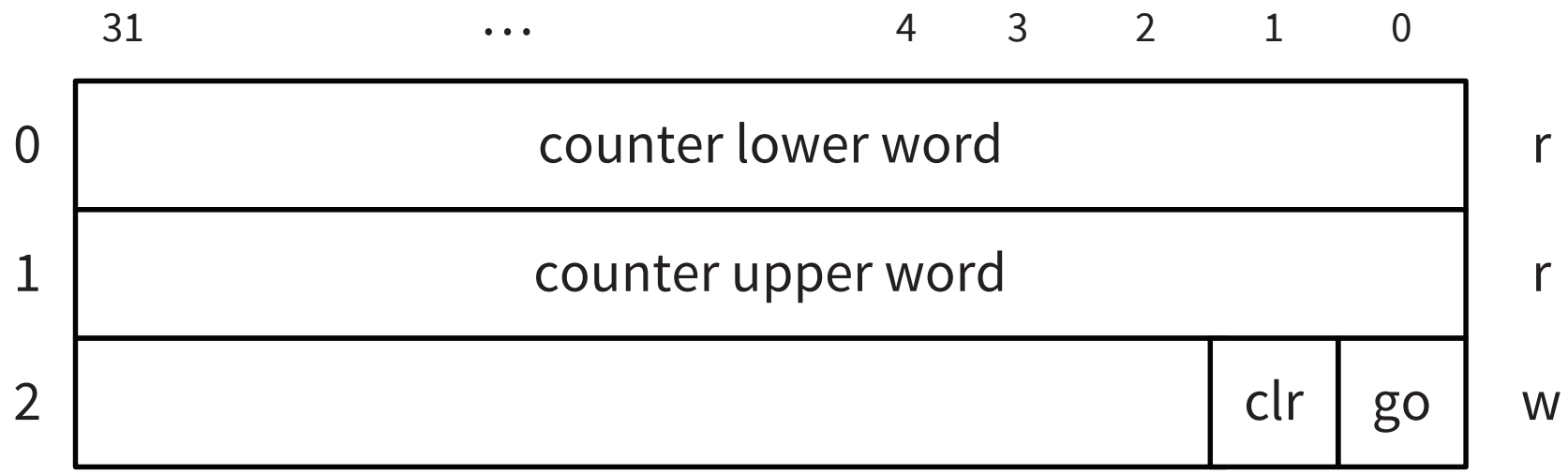
Interface of an MMIO Core



Example:
I/O Register Map of the Timer Core

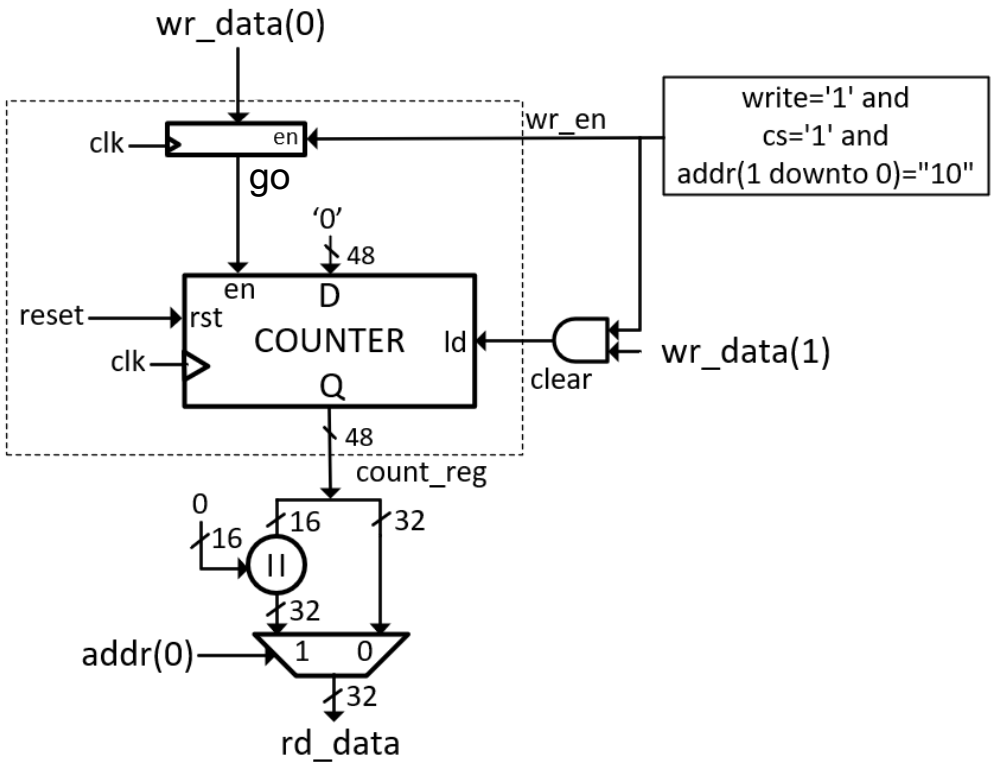


IO Register Map of the Timer Core



cs	write	read	addr(1:0)	wr_data(1:0)	Operation
1	0	1	00	−−	Read lower word
1	0	1	01	−−	Read upper word
1	1	0	10	01	Set “go” signal
1	1	0	10	00	Reset “go” signal
1	1	0	10	1−	Set “clear” signal

Wrapping Circuit



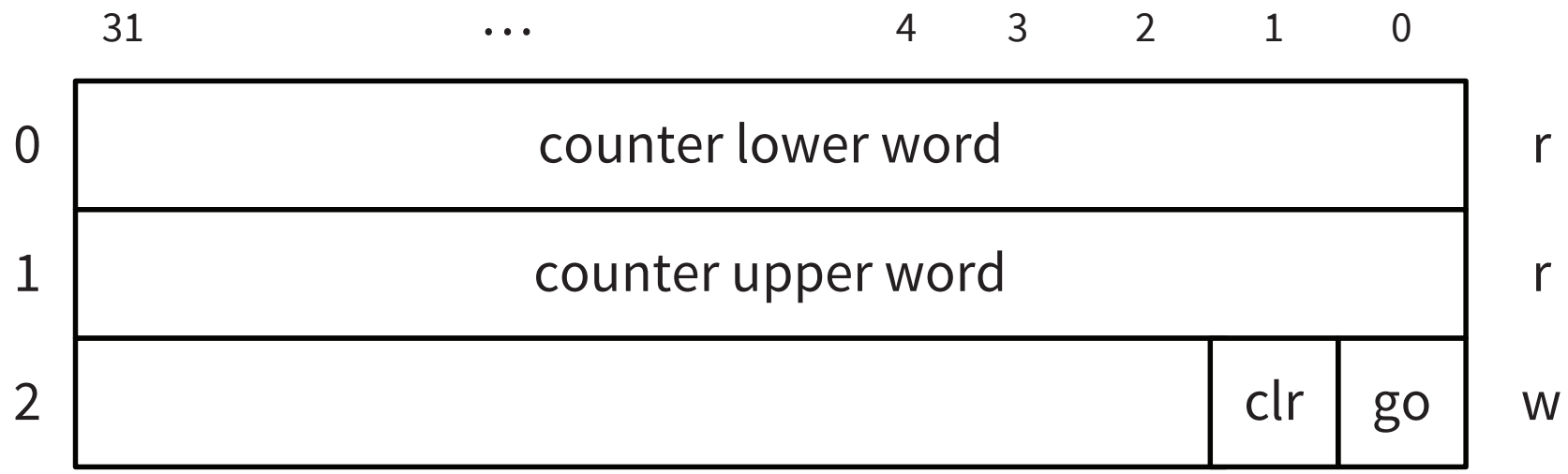
cs	write	read	addr(1:0)	wr_data(1:0)	Operation
1	0	1	00	--	Read lower word
1	0	1	01	--	Read upper word
1	1	0	10	01	Set “go” signal
1	1	0	10	00	Reset “go” signal
1	1	0	10	1-	Set “clear” signal

Timer Core in VHDL (4)

```
 -- decoding logic
   wr_en <= '1' when write='1' and cs='1' and addr(1 downto 0)="10" else '0';
   clear <= '1' when wr_en='1' and wr_data(1)='1' else '0';
   go    <= ctrl_reg;

 -- slot read multiplexing
   rd_data <=
      std_logic_vector(count_reg(31 downto 0)) when addr(0)='0' else
      x"0000" & std_logic_vector(count_reg(47 downto 32));
end arch;
```

IO Register Map of the Timer Core



cs	write	read	addr	wr_data(1:0)	Operation
1	0	1	00000	−−	Read lower word
1	0	1	00001	−−	Read upper word
1	1	0	00010	01	Set “go” signal
1	1	0	00010	00	Reset “go” signal
1	1	0	00010	1−	Set “clear” signal

Timer Core in VHDL (4)

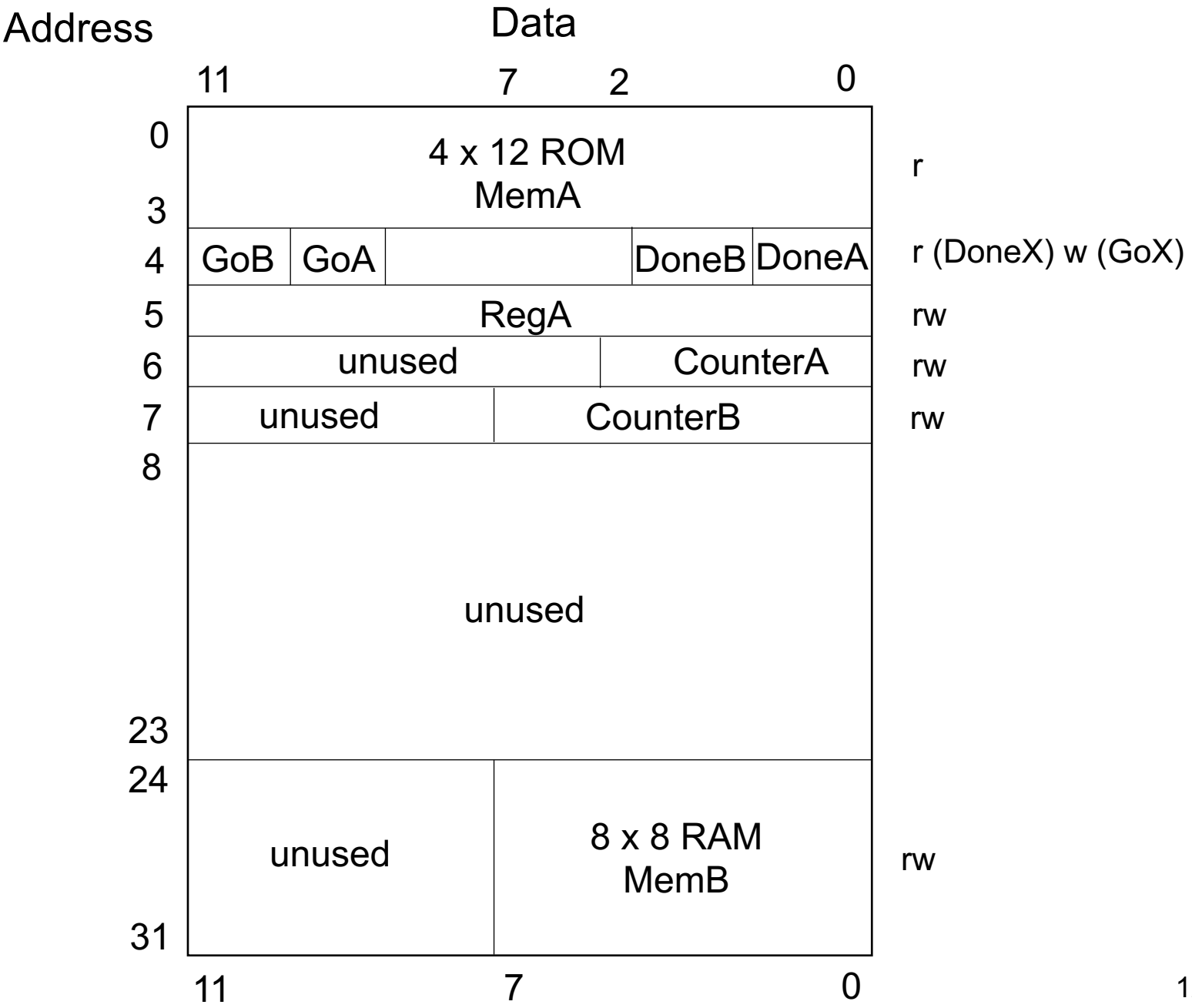
```
 -- decoding logic
   wr_en <= '1' when write='1' and cs='1' and addr="00010" else '0';
   clear <= '1' when wr_en='1' and wr_data(1)='1' else '0';
   go    <= ctrl_reg;

 -- slot read multiplexing
   rd_data <=
      std_logic_vector(count_reg(31 downto 0)) when addr="00000" else
      x"0000" & std_logic_vector(count_reg(47 downto 32)) when addr="00001" else
      x"00000000";
end arch;
```

Class Exercise Specification

Task:

Draw a detailed internal block diagram of
an FPro MMIO unit with the standard interface
and the Memory & I/O Register Map
shown on the next page


Address
Data
11
7
2
0
0
3
4 x 12 ROM
MemA
r
4
GoB
GoA
DoneB
DoneA
r (DoneX) w (GoX)
5
RegA
rw
6
unused
CounterA
rw
7
unused
CounterB
rw
8
unused
23
24
unused
8 x 8 RAM
MemB
rw
31
11
7
0

Assumptions

- Registers must be implemented using actual registers, not memory. Register RegA can be written to and read from.
- CounterA is a 3-bit counter, CounterB is an 8-bit counter. Writing to a counter initializes this counter, reading from a counter, reads its current value.
  Assume that initializing a counter requires only asserting ld.
- GoA and GoB and 1-bit write-only flags located at the two most significant bit locations at address 4.
- DoneA and DoneB and 1-bit read-only flags located at the two least significant bit locations at address 4.

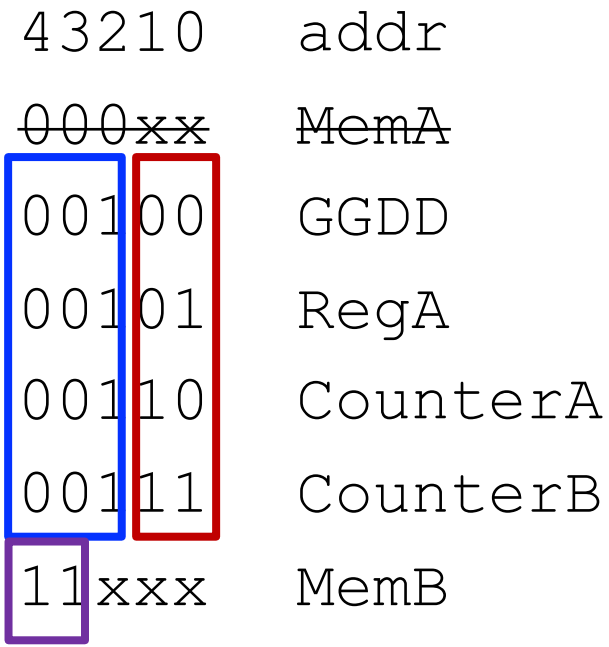
Address Decoding

	16	8	4	2	1
addr	4	3	2	1	0
min	0	0	0	0	0
max	0	0	0	1	1
MemA	0	0	0	x	x

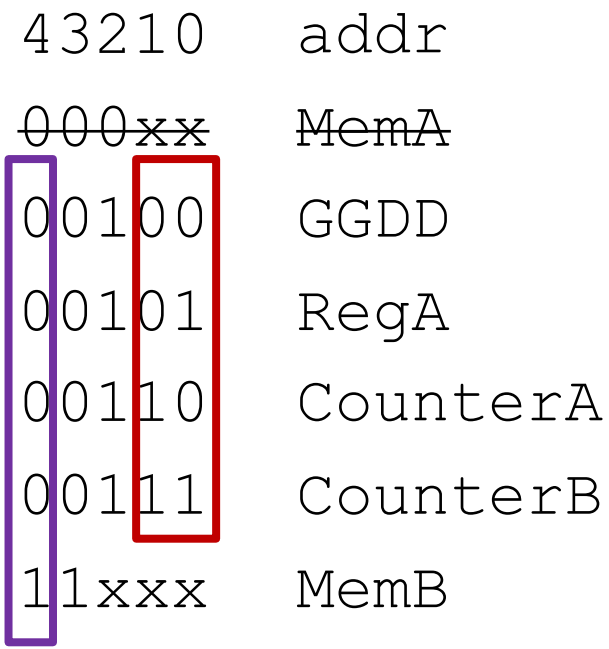
	16	8	4	2	1
addr	4	3	2	1	0
min	1	1	0	0	0
max	1	1	1	1	1
MemB	1	1	x	x	x

43210	addr
000xx	MemA
00100	GGDD
00101	RegA
00110	CounterA
00111	CounterB
11xxx	MemB

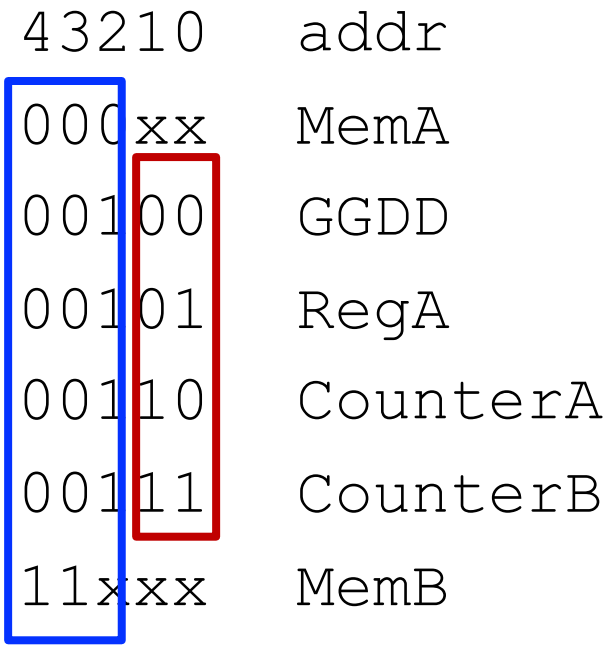
Exact Address Decoding for Writing



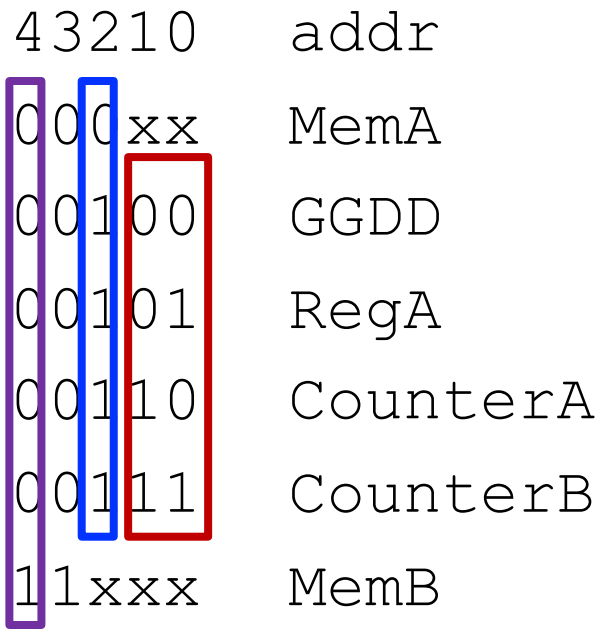
Simplified Address Decoding for Writing



Exact Address Decoding for Reading

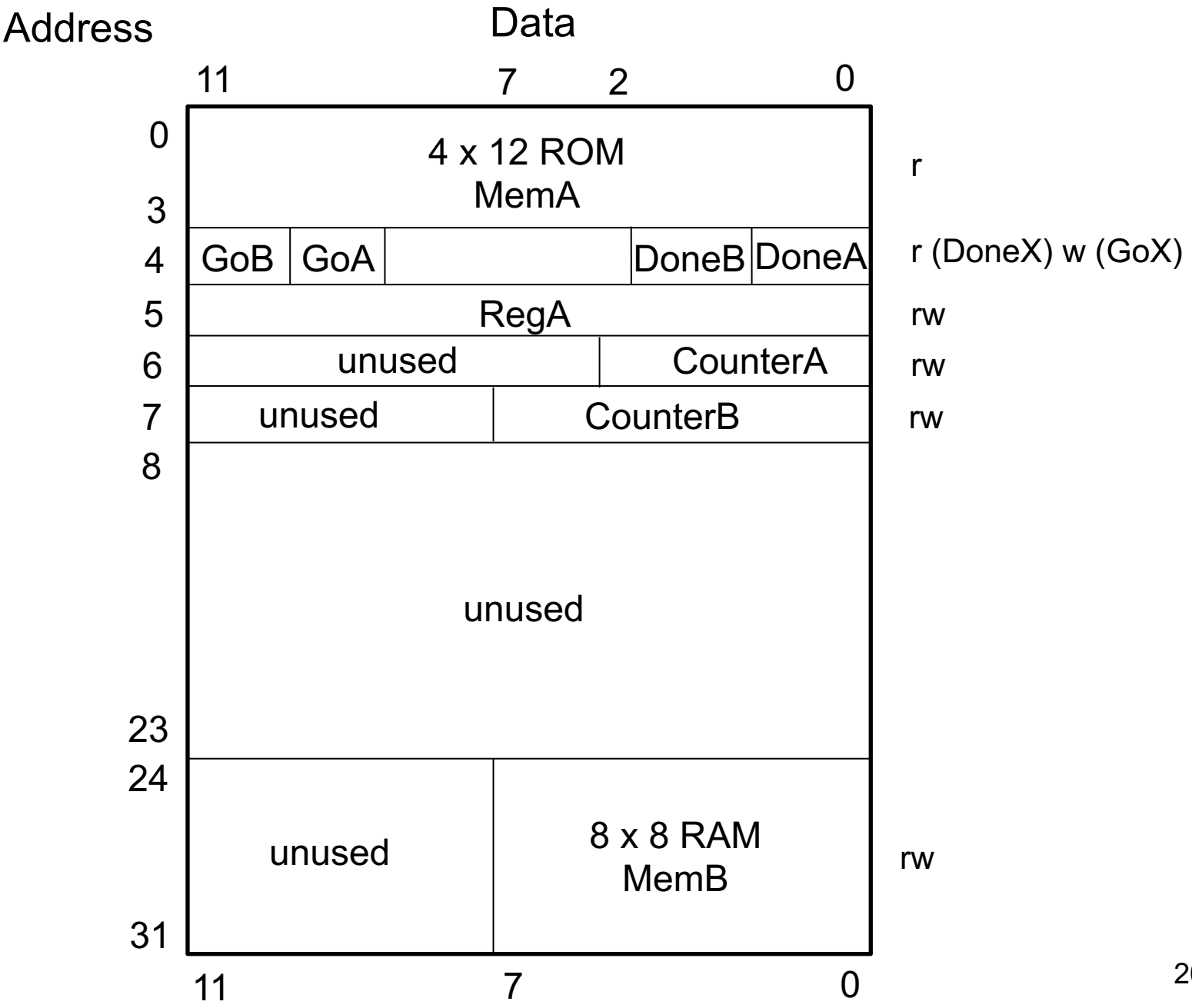


Simplified Address Decoding for Reading

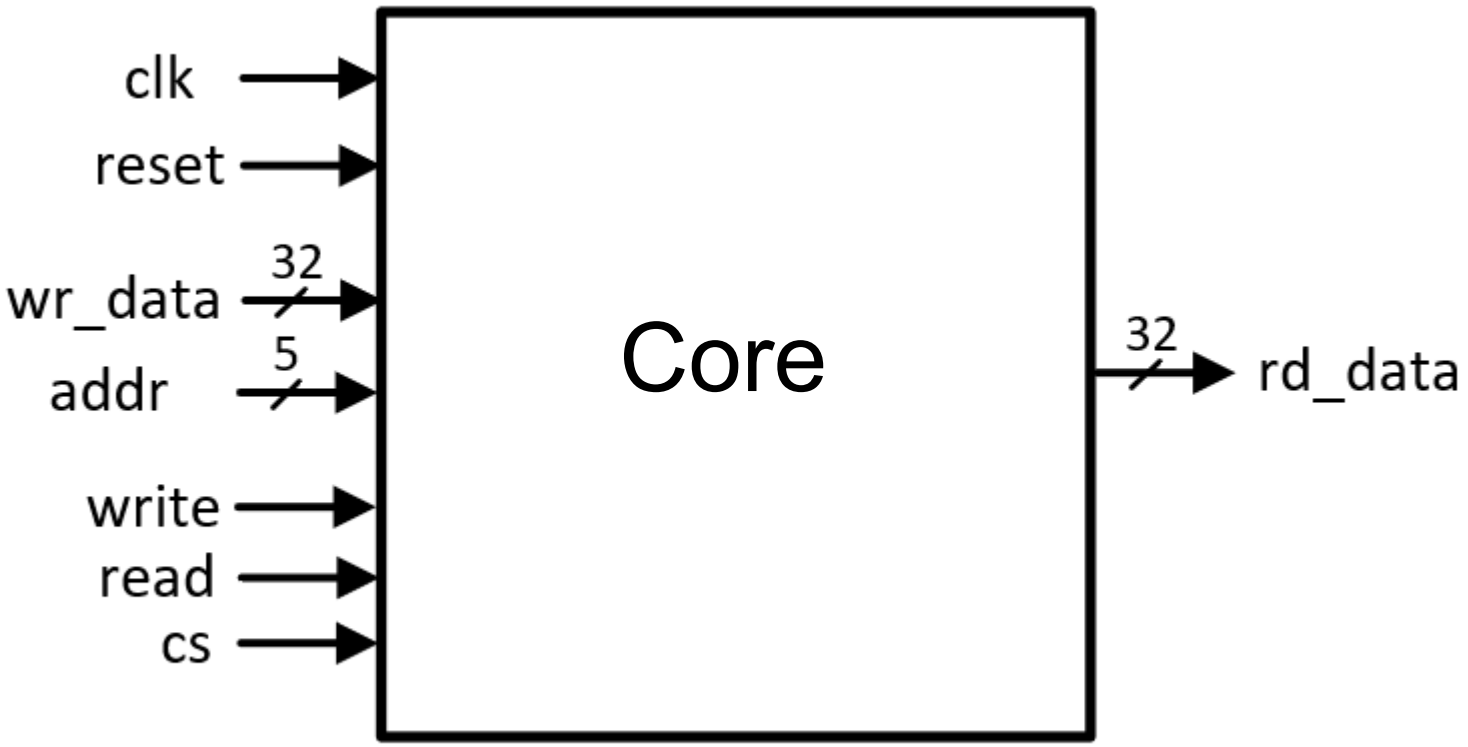


Approach 1

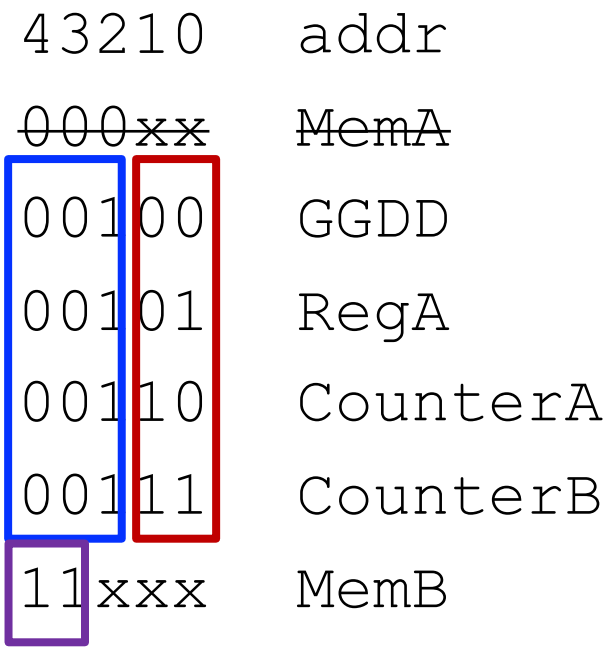
Exact Address Decoding


Address
Data
11
7
2
0
0
3
4 x 12 ROM
MemA
r
4
GoB
GoA
DoneB
DoneA
r (DoneX) w (GoX)
5
RegA
rw
6
unused
CounterA
rw
7
unused
CounterB
rw
8
unused
23
24
unused
8 x 8 RAM
MemB
rw
31
11
7
0

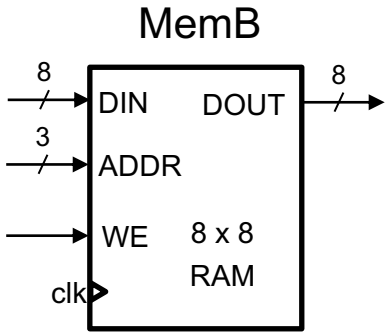
Interface of an MMIO Core



Address Decoding for Writing

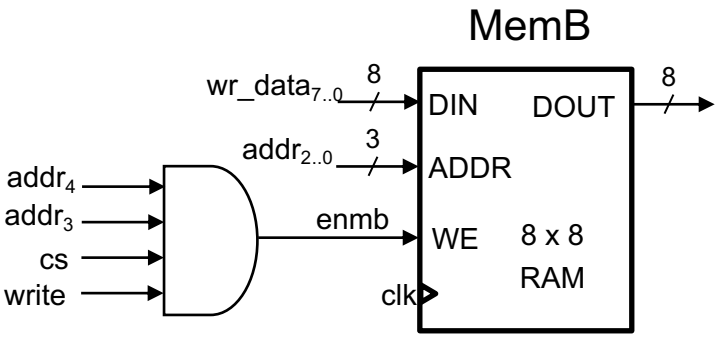


MemB

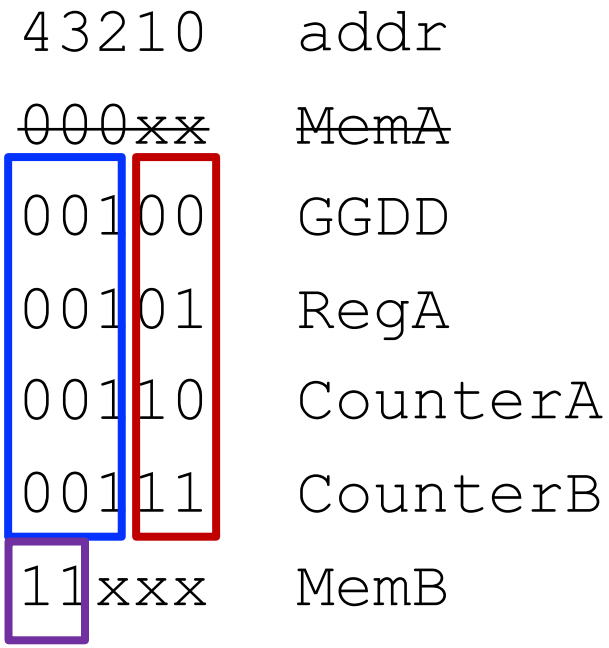


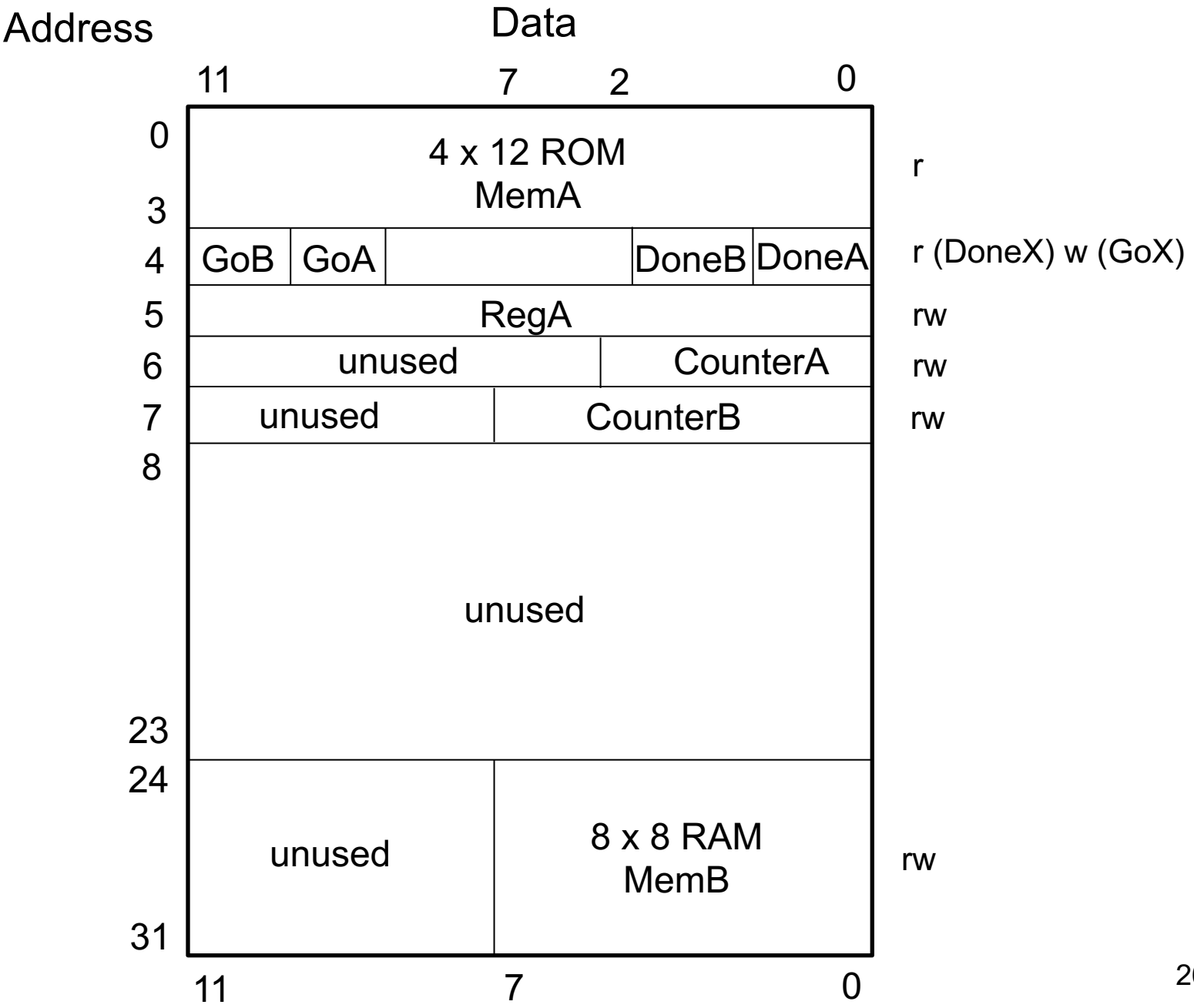
	16	8	4	2	1
addr	4	3	2	1	0
min	1	1	0	0	0
max	1	1	1	1	1
MemB	1	1	x	x	x

MemB

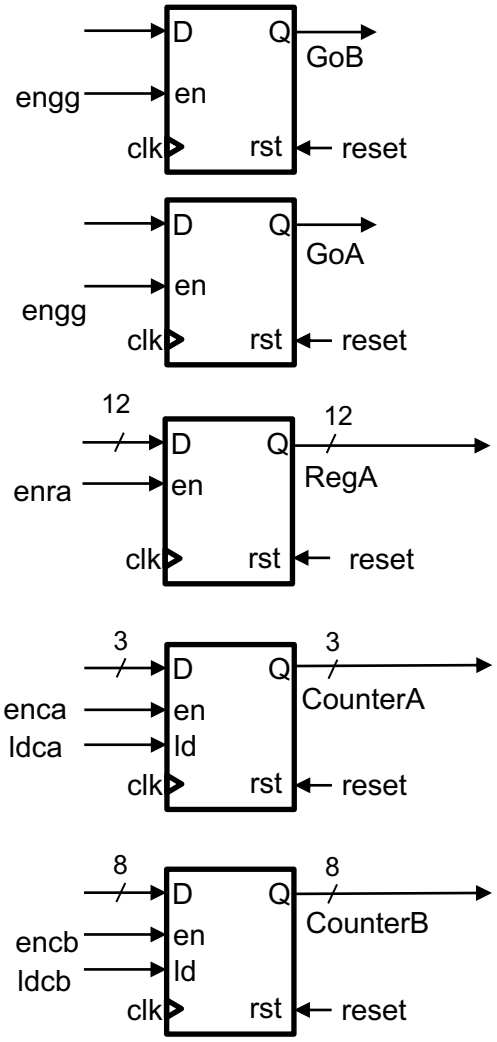

MemB
wr_data7..0
8
DIN
DOUT
8
addr2..0
3
ADDR
addr4
addr3
cs
write
enmb
WE
8 x 8
RAM
clk

Address Decoding for Writing

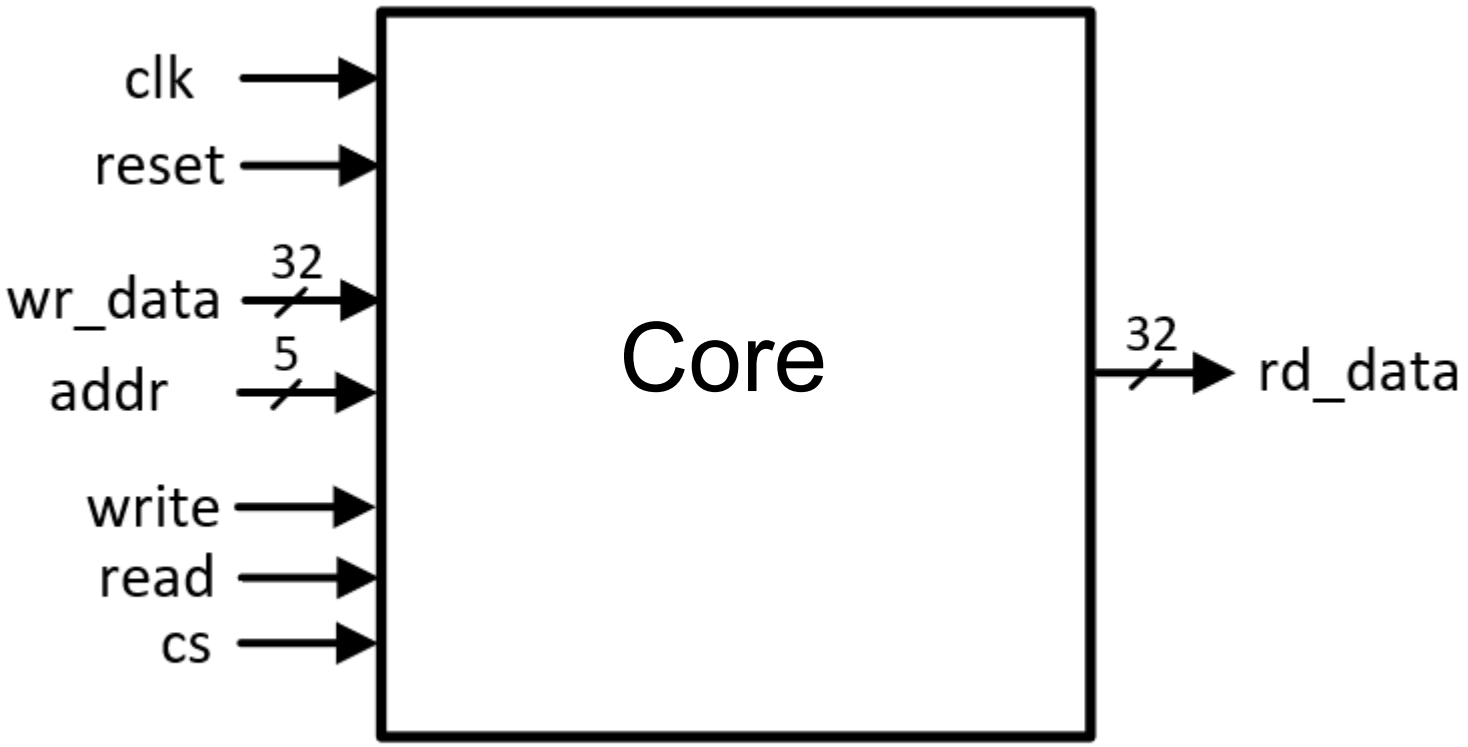



Address
Data
11
7
2
0
0
3
4 x 12 ROM
MemA
r
4
GoB
GoA
DoneB
DoneA
r (DoneX) w (GoX)
5
RegA
rw
6
unused
CounterA
rw
7
unused
CounterB
rw
8
unused
23
24
unused
8 x 8 RAM
MemB
rw
31
11
7
0

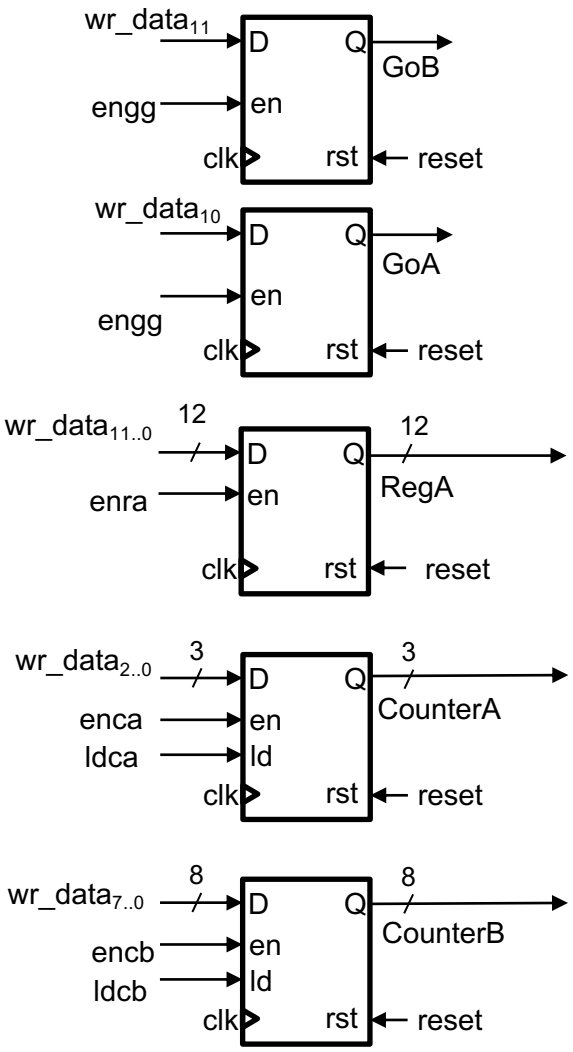
GoA, GoB, RegA, CounterA, and CounterB



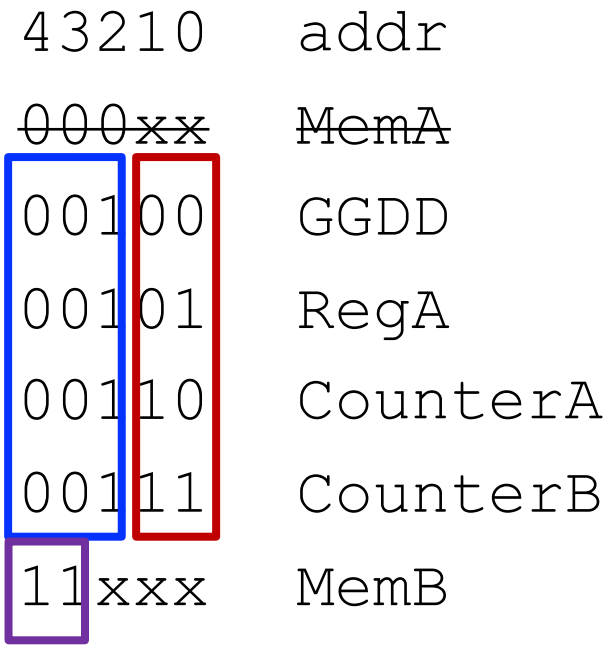
Interface of an MMIO Core



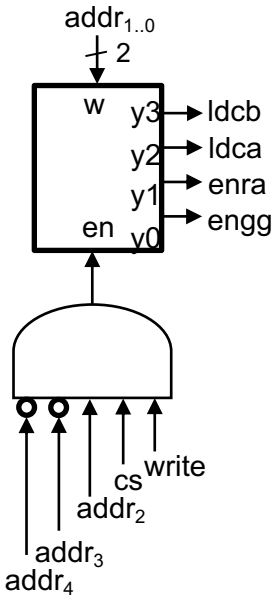
GoA, GoB, RegA, CounterA, and CounterB

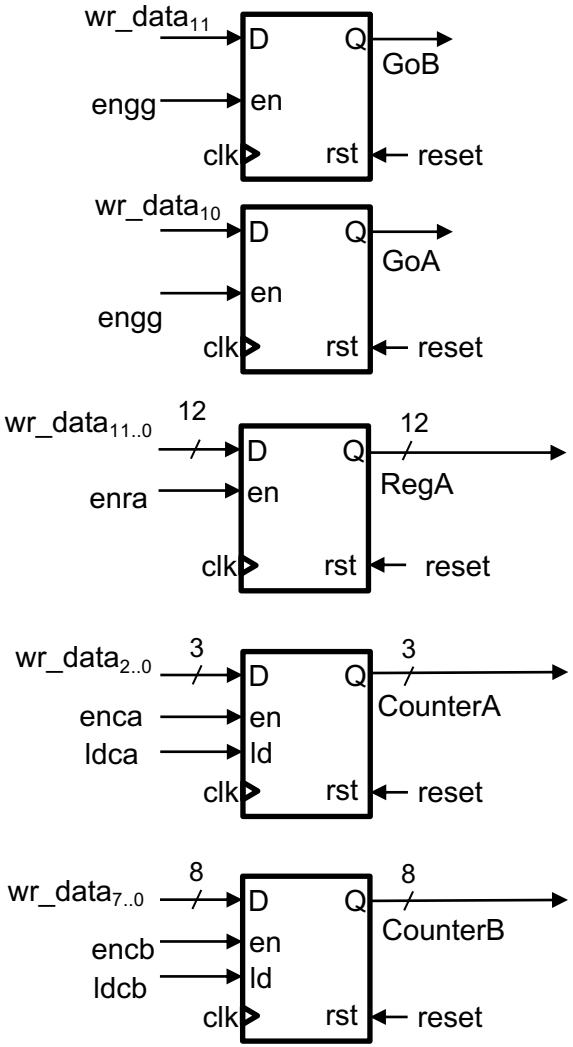


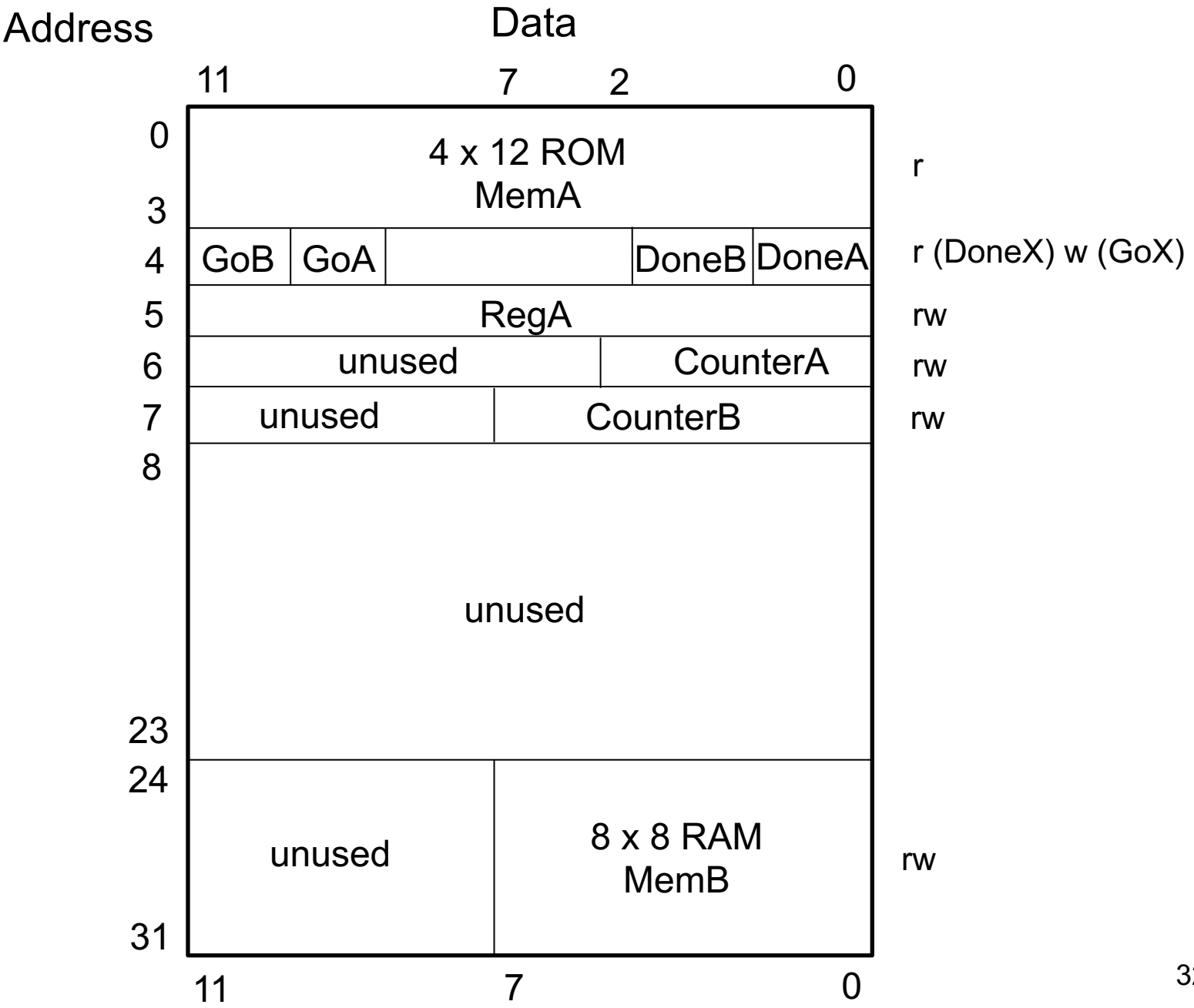
Address Decoding for Writing

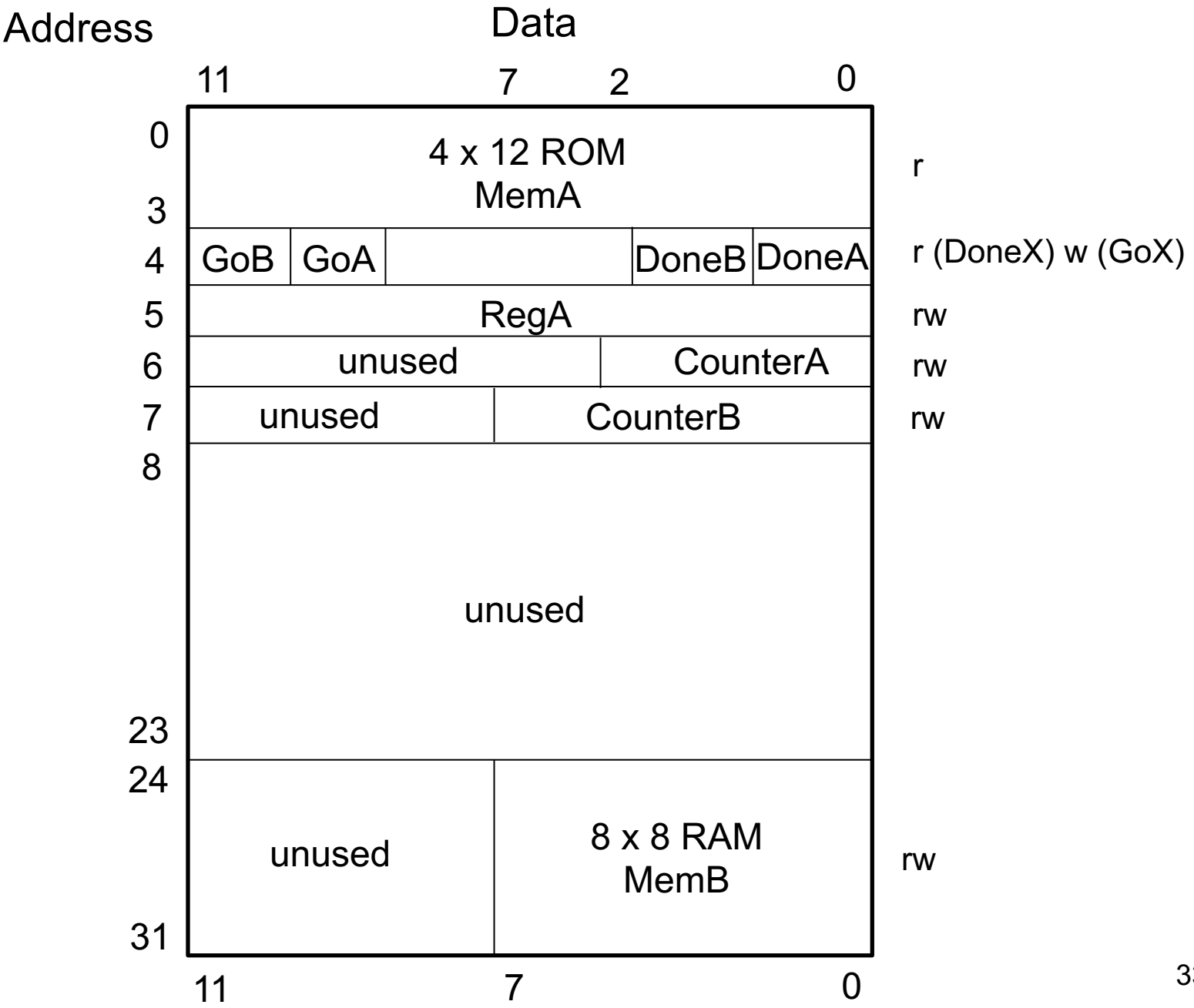


GoA, GoB, RegA, CounterA, and CounterB

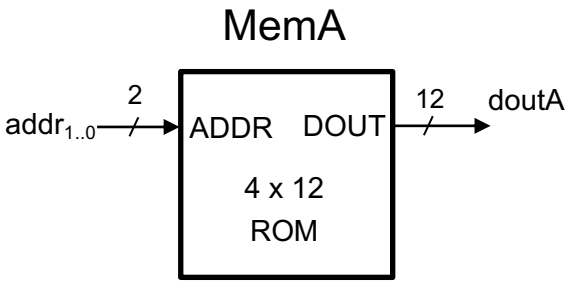


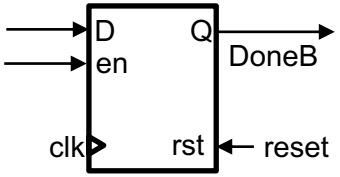


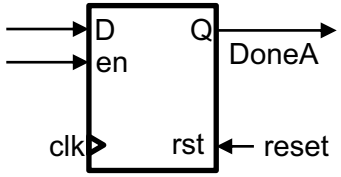

Address
Data
11
7
2
0
0
3
4 x 12 ROM
MemA
r
4
GoB
GoA
DoneB
DoneA
r (DoneX) w (GoX)
5
RegA
rw
6
unused
CounterA
rw
7
unused
CounterB
rw
8
unused
23
24
unused
8 x 8 RAM
MemB
rw
31
11
7
0


Address
Data
11
7
2
0
0
3
4 x 12 ROM
MemA
r
4
GoB
GoA
DoneB
DoneA
r (DoneX) w (GoX)
5
RegA
rw
6
unused
CounterA
rw
7
unused
CounterB
rw
8
unused
23
24
unused
8 x 8 RAM
MemB
rw
31
11
7
0

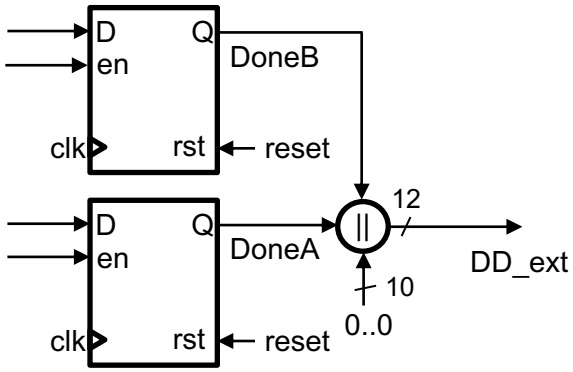
MemA, DoneA, DoneB



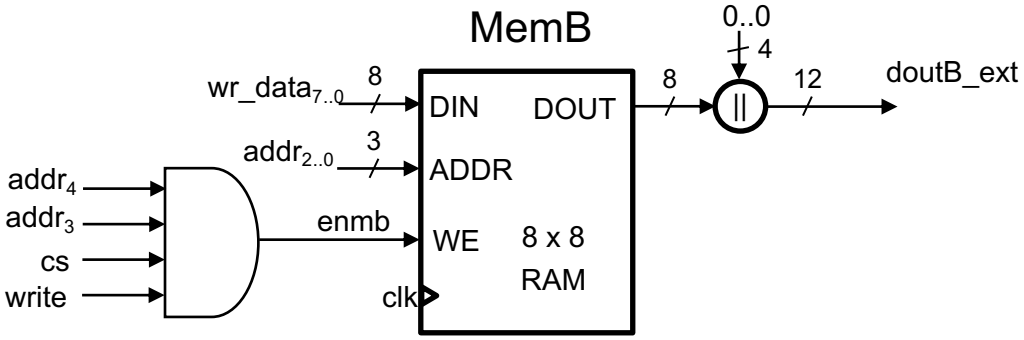




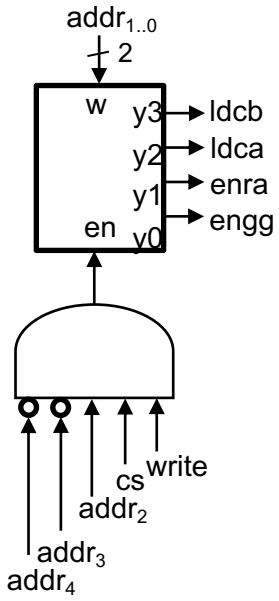
DoneA & Done B

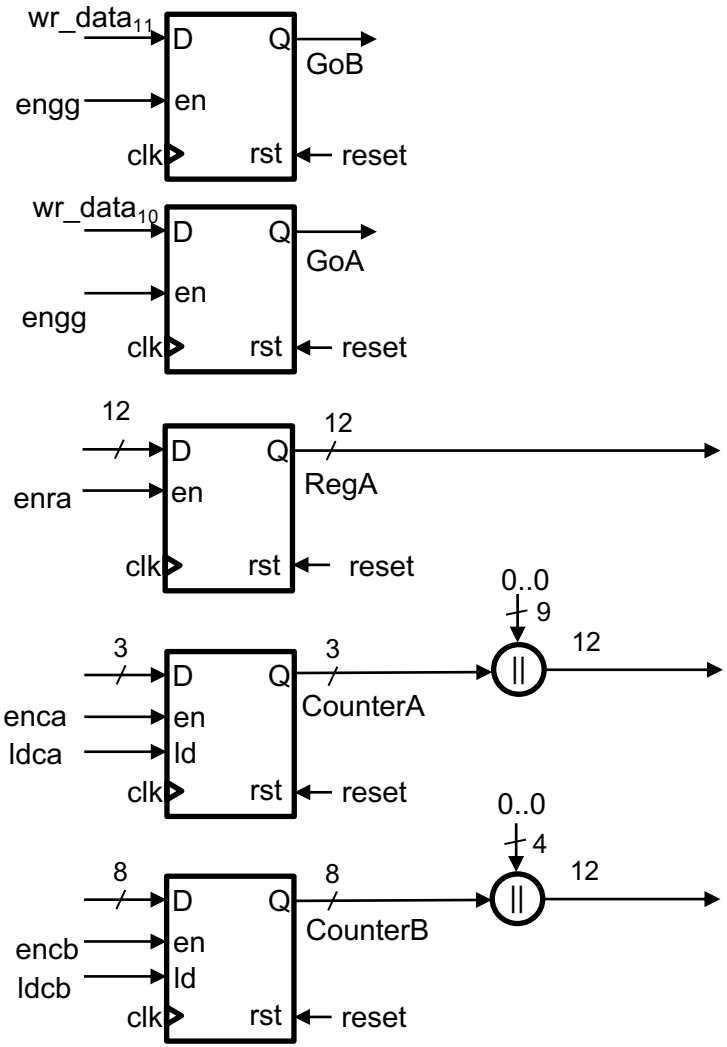

D
en
Q
DoneB
clk
rst
reset
D
en
Q
DoneA
clk
rst
reset
||
12
10
0..0
DD_ext

MemB

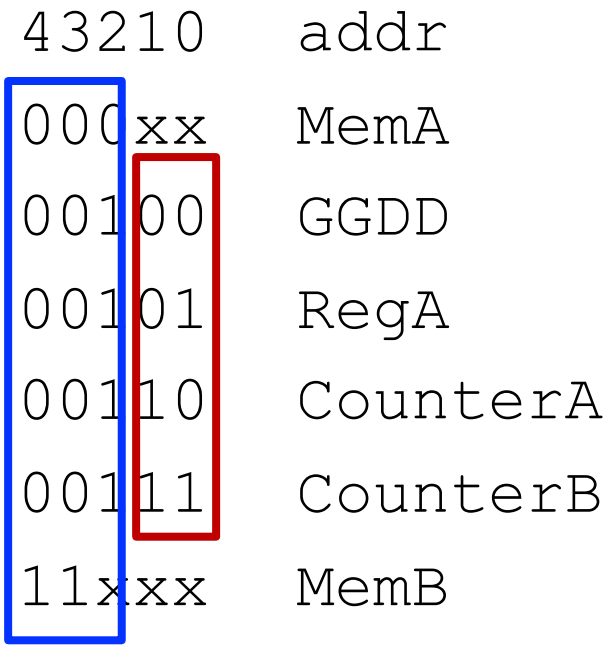

MemB
0..0
4
wr_data7..0
8
DIN
DOUT
8
12
doutB_ext
addr2..0
3
ADDR
addr4
addr3
cs
write
enmb
WE
8 x 8
RAM
clk

GoA, GoB, RegA, CounterA, and CounterB

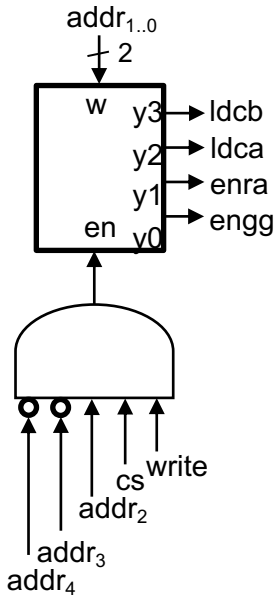


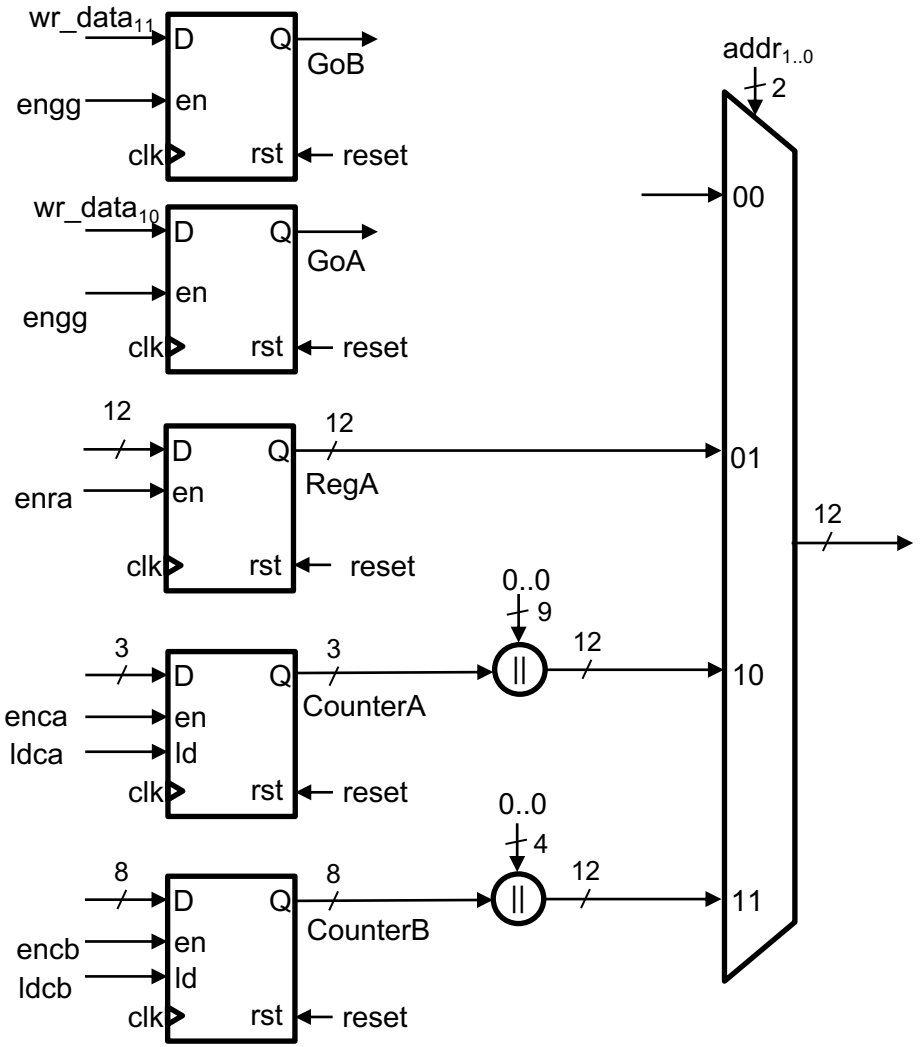


Exact Address Decoding for Reading

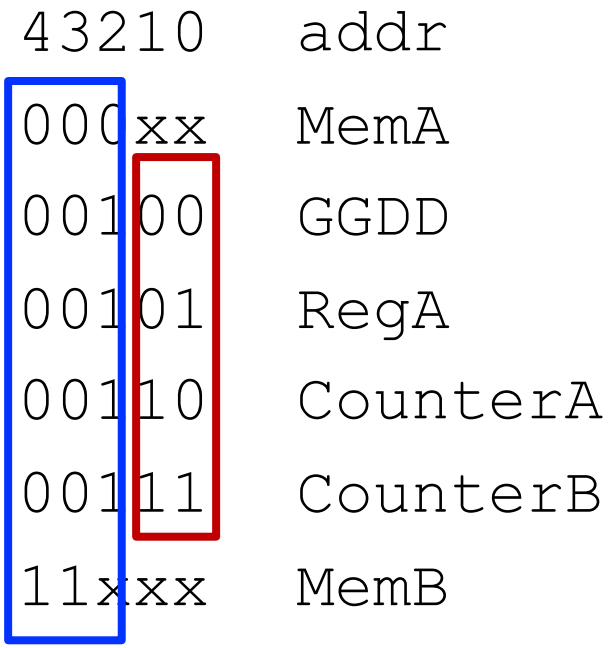


GoA, GoB, RegA, CounterA, and CounterB

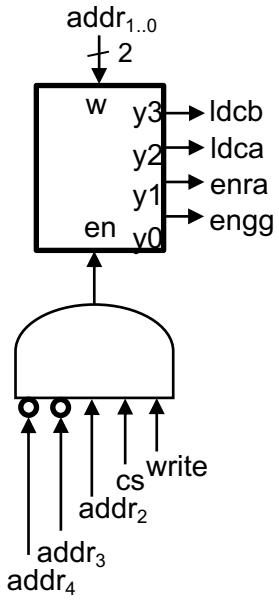


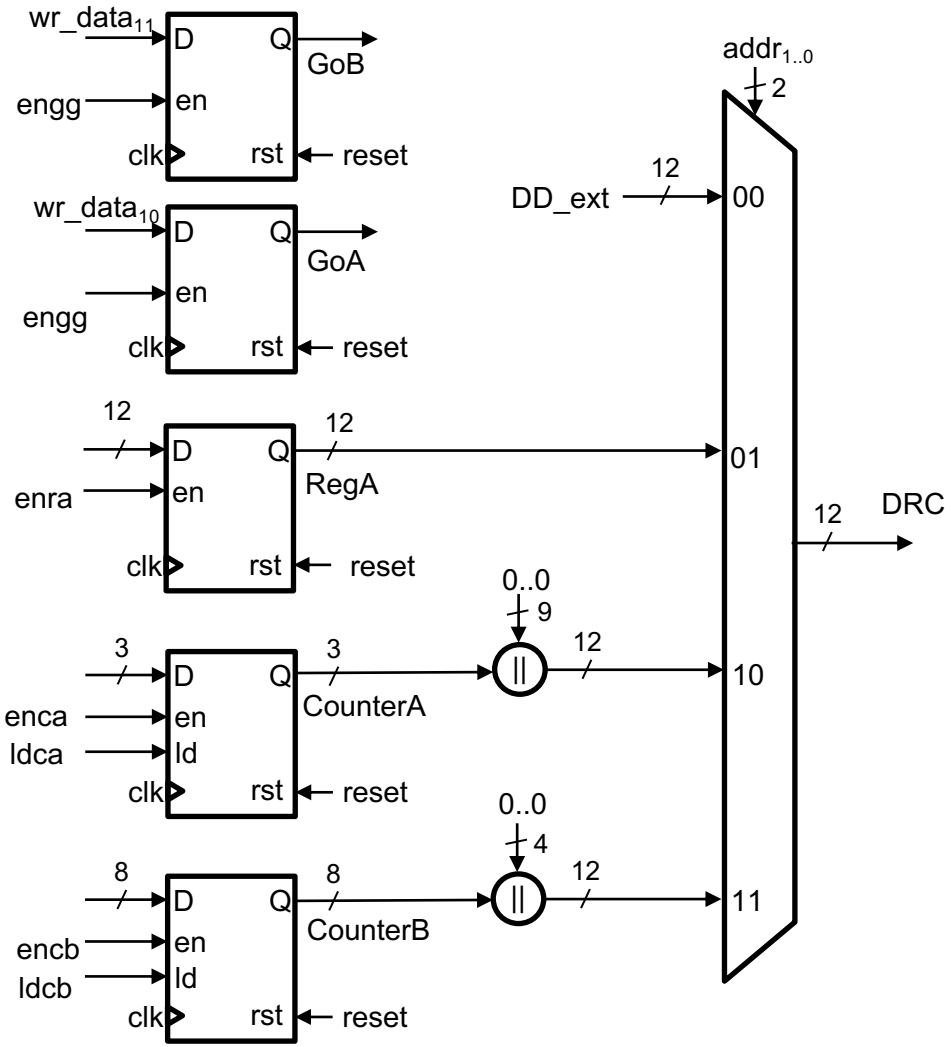


Address Decoding for Reading – Method 1

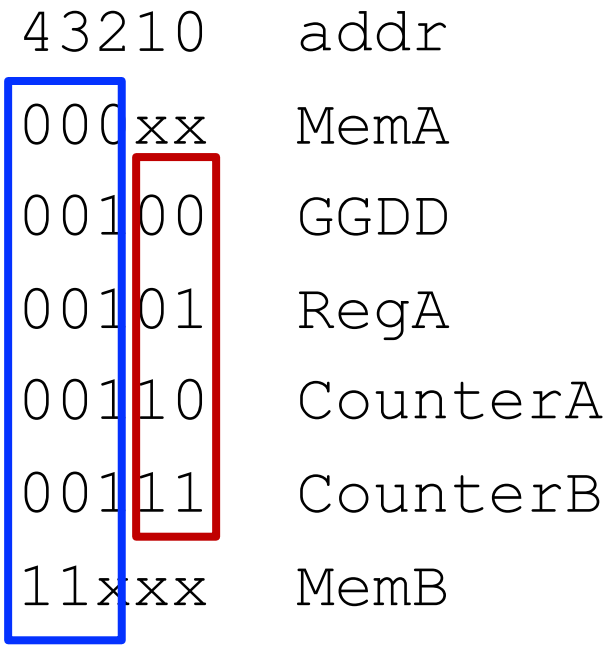


GoA, GoB, RegA, CounterA, and CounterB

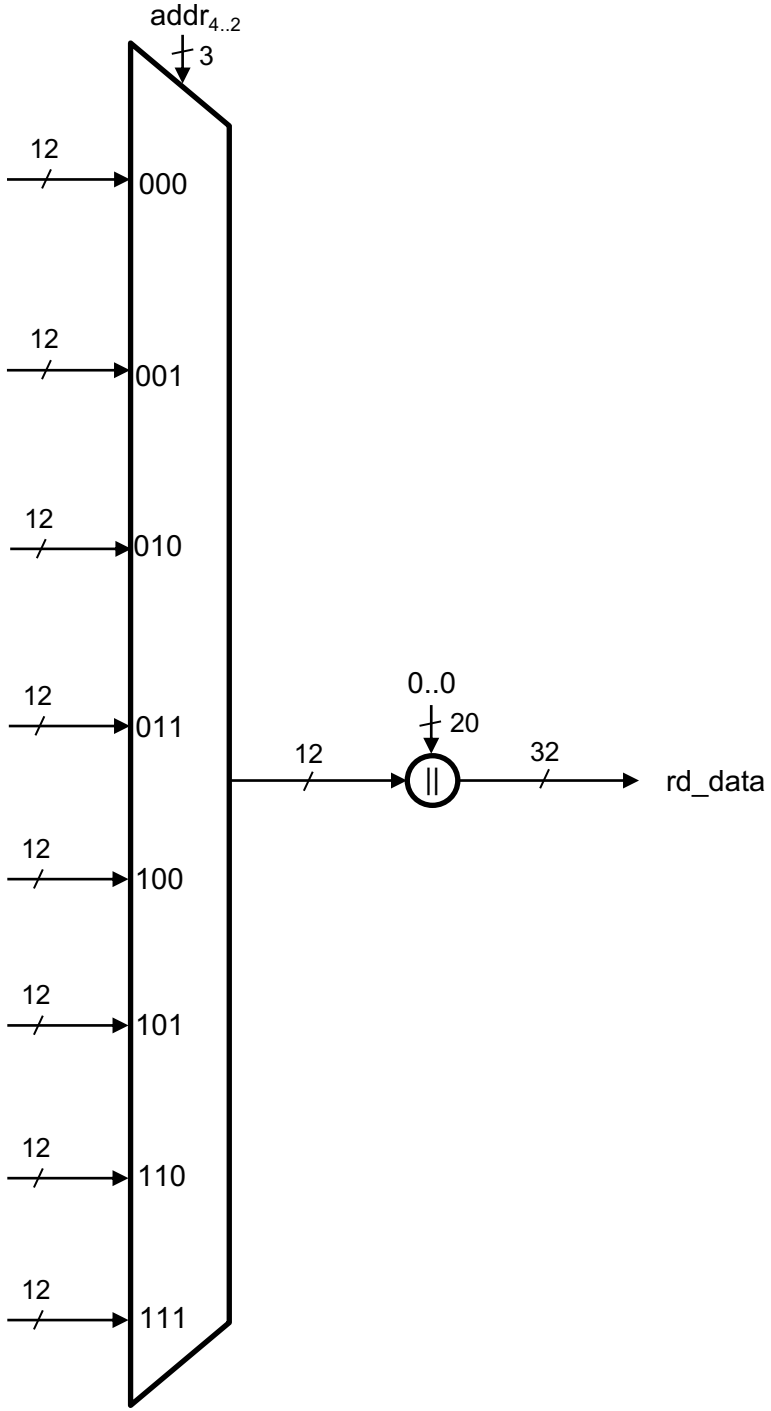




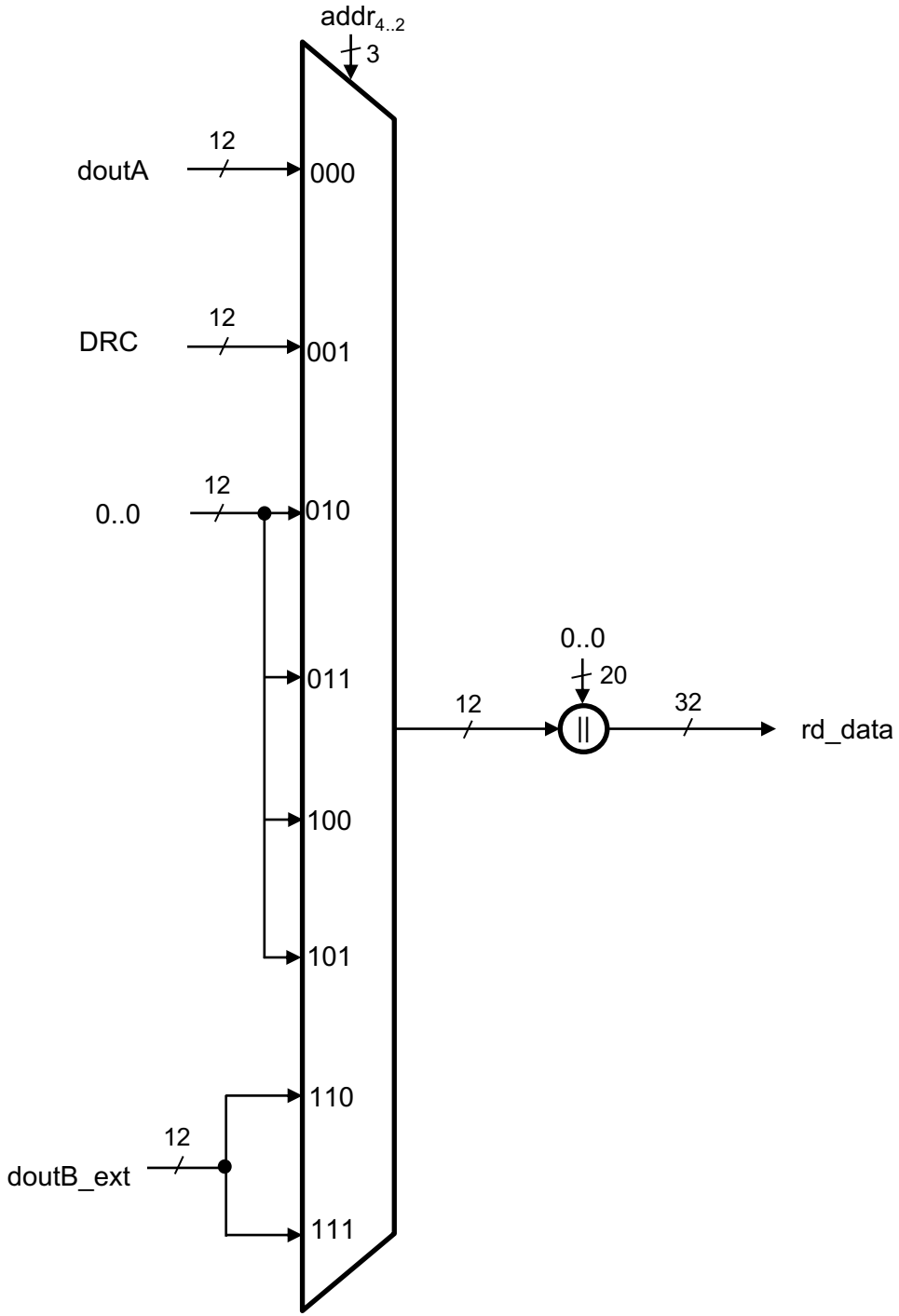
Address Decoding for Reading – Method 1



Output MUX


addr4..2
3
12
000
12
001
12
010
12
011
12
100
12
101
12
110
12
111
0..0
20
12
32
rd_data

Output MUX


addr4..2
3
12
doutA
000
12
DRC
001
12
0..0
010
011
100
101
110
12
doutB_ext
111
0..0
20
12
32
rd_data