Chapter 14

Exercise 14.1 The design of the 2-read/1-write register file can be implemented with registers with 2 tristate outputs (A and B), as shown in Figure **??**. We call these registers: RF Registers. Besides the two tristate outputs, they have one input din, a write enable input EWi, and two "enable output" inputs: EOA and EOB to control each of the register tristate outputs. The design of this component based on a conventional register and tristate buffers is also shown. The interconnection of all the A outputs of the RF registers forms the $RBUSA$ (read bus A). Similarly for the B outputs. All the RF register inputs are also interconnected, forming the write bus $WBUS$. The binary decoders are used to generate the control signals for writing and reading the register file.

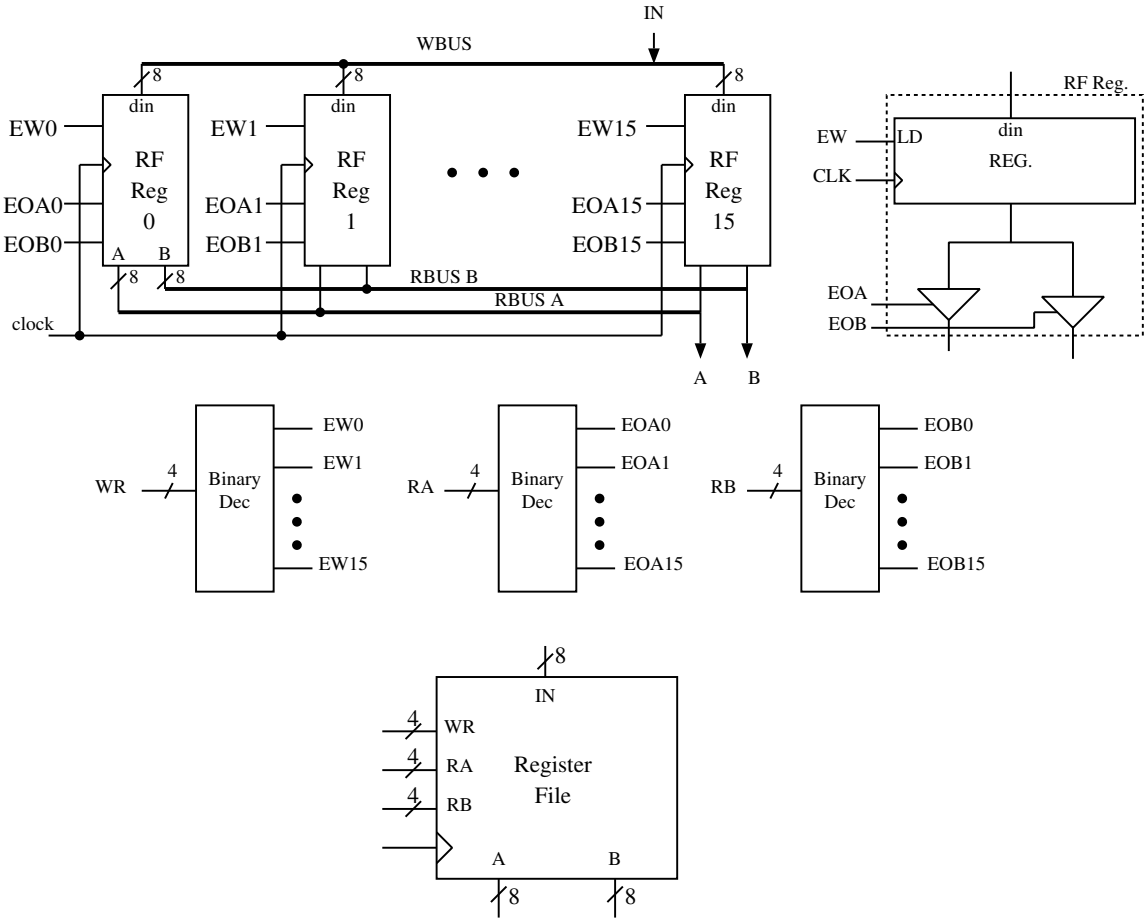


Figure 14.1: 2-read/1-write register file

Exercise 14.3

(a)

$$\overline{T} = \sum_{d=0}^{2^n-1} (dC_1 + C_2)p(d)$$

where $p(d)$ is the probability of distance d. As distances are uniformly distributed, we get:

$$p(d) = \frac{1}{2^n}$$

Based on the previous equations we obtain:

$$\overline{T} = \sum_{d=0}^{2^n-1} (dC_1 + C_2)\frac{1}{2^n} = \frac{1}{2^n}\left(2^n C_2 + C_1 \frac{2^n}{2}(2^n - 1)\right)$$

$$\overline{T} = C_2 + \frac{C_1(2^n - 1)}{2}$$

(b) when

$$p(d) = \begin{cases} 1 & \text{if } d = 1 \\ 0 & \text{otherwise} \end{cases} \tag{14.1}$$

the average time becomes:

$$\overline{T} = \sum_{d=0}^{2^n-1} (dC_1 + C_2)p(d) = C_1 + C_2$$

(c) *Block access.* Time to access a block of k words is $kC_1 + C_2$ (when assuming pipelined execution C_2 is absorbed in intermediate steps). The average time to go from block position i to block at position j, and access k consecutive positions is given as:

$$\begin{aligned} \overline{T}_{Block} &= \sum_{d=0}^{2^n-1} (dC_1 + (k-1)C_1 + C_2)p(d) \\ &= \frac{1}{2^n}\left[\frac{2^n}{2}(2^n - 1)C_1 + 2^n((k-1)C_1 + C_2)\right] \\ &= \frac{(2^n - 1)}{2}C_1 + (k-1)C_1 + C_2 \\ &= \frac{1}{k}\left[\frac{(2^n - 1)}{2}C_1 + (k-1)C_1 + C_2\right] \end{aligned}$$

Exercise 14.5

```
R3 <= R1; LP1 <= R2; RP1 <= R1; R1 <= P1;
LP2 <= R3; RP2 <= R4; R4 <= P2;
R2 <= R4; LP1 <= R2; RP1 <= R1; R1 <= P1;
LP2 <= R2; RP2 <= R3; R2 <= P2
```

The datapath for this computation is shown in Figure **??**. **Exercise 14.7** A state diagram for the control subsystem is shown in Figure **??**.

(a) execution of the computation for $X = 00101110$ and $Y = 11100010$.

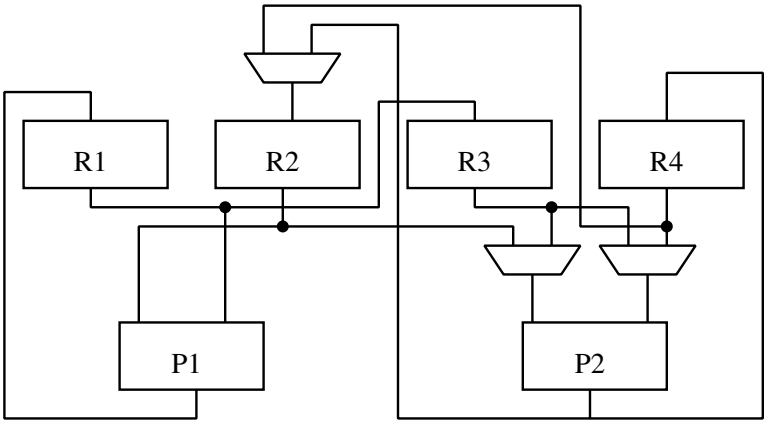


Figure 14.2: Datapath for Exercise 14.5

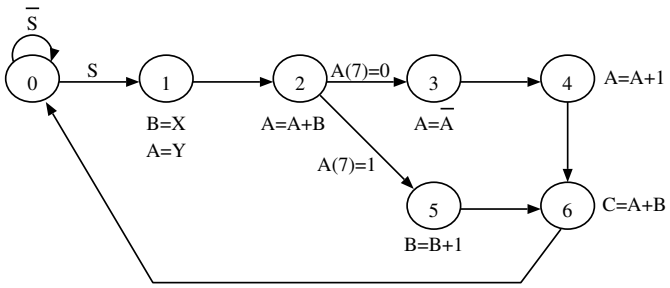


Figure 14.3: State Diagram for System in Exercise 14.7

state	A	B	C
0/1	xxxxxxxx	xxxxxxxx	xxxxxxxx
2	11100010	00101110	xxxxxxxx
5	00010000	00101110	xxxxxxxx
6	00010000	00101111	xxxxxxxx
0	00010000	00101111	00111111

Observe that the value of $A(7)$ tested in state 2, corresponds to the present value of the variable A (1110010), not the value that is going to be valid when the machine leaves state 2 (00010000).

(b) The data section that is required by the system is shown in Figure **??**.

(c) The state diagram that describes the control behavior and the control signals activation is shown in Figure **??**. For convenience, we replace the s input by start.

The implementation of the control circuit using a counter, multiplexer, a decoder, and gates is shown in Figure **??**.

The parallel load of the counter is defined as:

$$I = \begin{cases} 0 & \text{if } S_0 \text{ or } S_6 \\ 5 & \text{if } S_2 \\ 6 & \text{if } S_4 \end{cases} \tag{14.2}$$

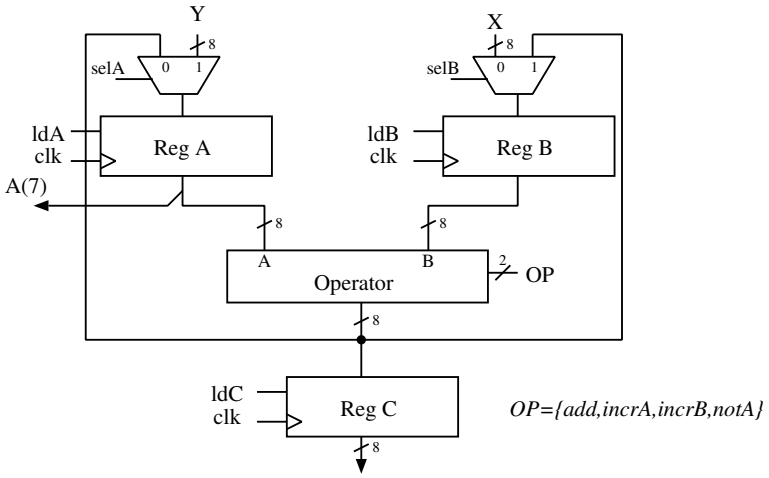


Figure 14.4: Datapath for Exercise 14.7

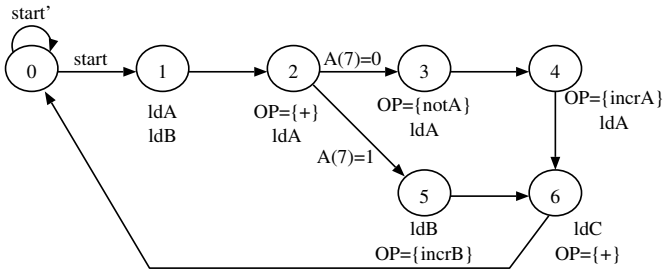


Figure 14.5: State diagram for the control section - Exercise 14.7

A table that represents these values in a table is:

state	code	I
S_0	000	000
S_1	001	- - -
S_2	010	101
S_3	011	- - -
S_4	100	110
S_5	101	- - -
S_6	110	000
S_7	111	- - -

Minimal expressions are obtained with the following Kmaps:

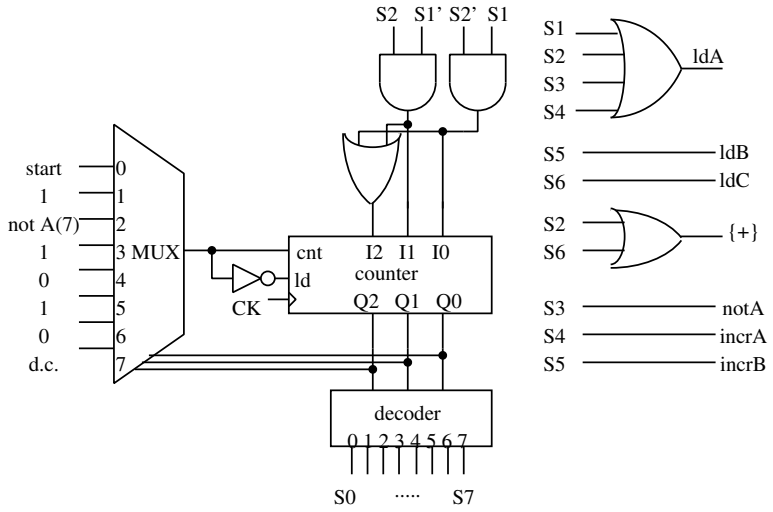
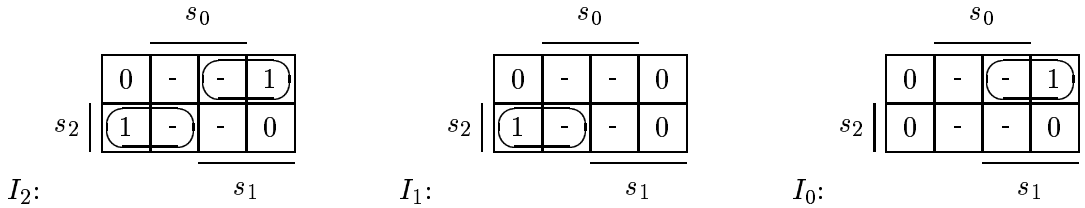


Figure 14.6: Control network for Exercise 14.7



$$
\begin{aligned}
I_2 &= s_2' s_1 + s_2 s_1' = s_2 \oplus s_1 \\
I_1 &= s_2 s_1' \\
I_0 &= s_2' s_1
\end{aligned}
$$

The counter enable input is activated according to the expression:

$$CNT = start.S_0 + S_1 + S_2 A(7)' + S_3 + S_5$$

and

$$LD = CNT'$$

Another implementation of the control system using one FF per state approach is shown in Figure **??**.

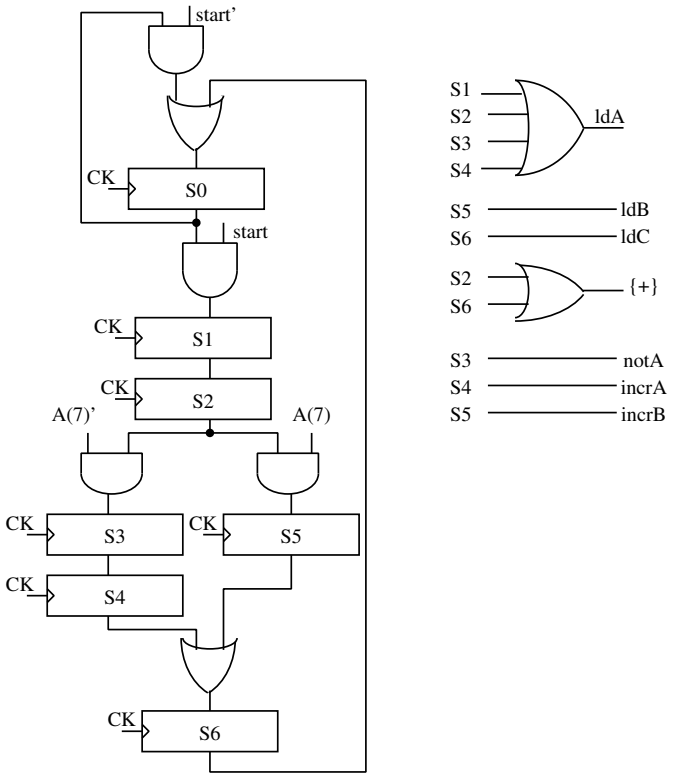


Figure 14.7: One-flip-flop-per-state implementation of control system - Exercise 14.7

Exercise 14.9 The datapath for the system is shown in Figure **??**.

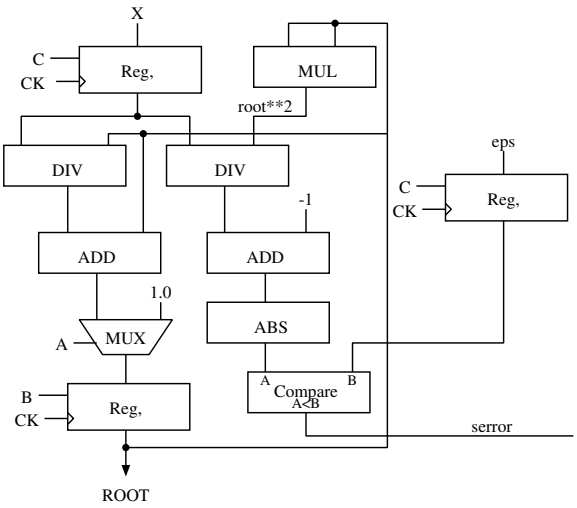


Figure 14.8: Datapath for the system in Exercise 14.9

The state diagram that describes the control section is shown in Figure **??**. We consider that the system has a *start* input that indicates when another calculation must begin.

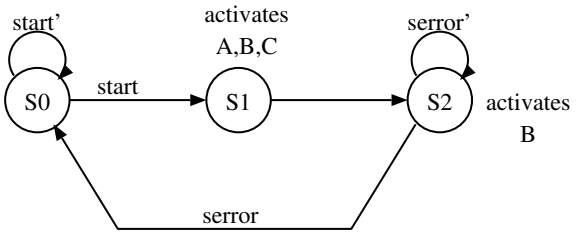


Figure 14.9: State Diagram for control section - Exercise 14.9

(a) implementation of the control section using D-type FFs. The states are encoded as:

state	y_1y_0
S0	00
S1	01
S2	10

From these state codes we determine the binary transition table as follows:

	Input (start/serror)			
y_1y_0	00	01	11	10
00	00,000	00,000	01,000	01,000
01	10,111	10,111	10,111	10,111
10	10,010	00,010	00,010	10,010
	NS(Y_1Y_0, ABC)			

These are minimal expressions for the next state and output signals:

$$\begin{aligned} Y_1 &= y_1 serror' + y_0 \\ Y_0 &= y_1' y_0' start \\ A &= y_0 = C \\ B &= y_1 + y_0 \end{aligned}$$

The gate network for the control section is shown in Figure **??**.

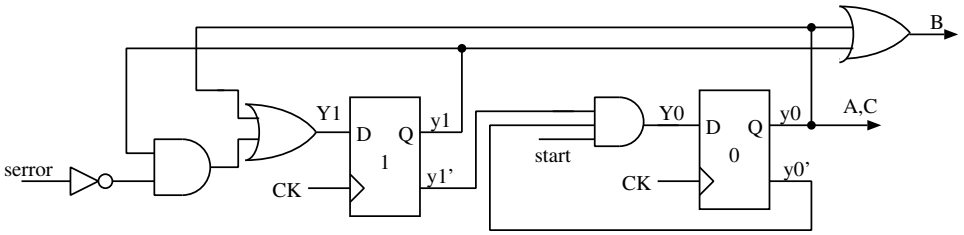


Figure 14.10: Gate Network for control section - Exercise 14.9

(b) implementation of the control section using PLA. For this case we use the same expressions obtained for solution (a). The implementation is presented in Figure **??**. **Exercise 14.11**

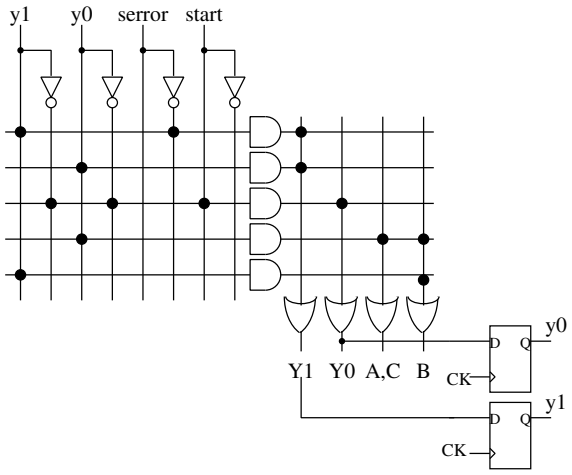


Figure 14.11: PLA implementation of control section - Exercise 14.9

(a) $R0 \leftarrow max\{R0, R1, \ldots, R7\}$, $R8$ and $R9$ don't need to be preserved, $\{R1, R2, \ldots, R7\}$ should be preserved.

Considering that the values are already in the registers, the control signals $S, L1, \ldots,$ and $L7$ will have a value zero all the time. Since there is no "compare" operation, the comparison between two values x and y must be made by add/sub operation. When the operation $x - y$ is executed, the carry out bit (cy) is 0 only for the case $y \leq x$.

The RTL description of the computation is as follows:

```
R9 <- R0
R8 <- R0 - R0  -- R8=0
R0 <- R1 - R9
if cy=0 R9 <- R1
R0 <- R2 - R9
if cy=0 R9 <- R2
.
.
.
R0 <- R7 - R9
if cy=0 R9 <- R7
R0 <- R9 + R8   -- R0=R9
```

If the registers were mapped to an array, we would be able to index each register, and this way use the **FOR ... LOOP** structure.

(b) micro-instructions (horizontal/ implicit addressing)

We use a two-format instruction. The condition field is defined as: 000-add, 001-sub, 010-incl, 011-xor, 100-and. The instruction formats and the instruction sequence is shown in the next table. Since the values of S and $L1$ to $L7$ are always zero, we don't show these signals in the microcode. Since there is only one condition to be tested, basically, the $(cy, 0)$ field could be eliminated. We kept the condition in the instruction to keep the format similar to the one shown in the text.

addr	Mode	source Register									OP	destination register		
	0	E0	E1	E2	E3	E4	E5	E6	E7	E8		L0	L8	L9
	1	(condition,value)										address		
0	0	1	0	0	0	0	0	0	0	0	- - -	0	0	1
1	0	1	0	0	0	0	0	0	0	0	0 0 1	0	1	0
2	0	0	1	0	0	0	0	0	0	0	0 0 1	1	0	0
3	1	(cy,0)										5		
4	0	0	1	0	0	0	0	0	0	0	- - -	0	0	1
5	0	0	0	1	0	0	0	0	0	0	0 0 1	1	0	0
6	1	(cy,0)										8		
7	0	0	0	1	0	0	0	0	0	0	- - -	0	0	1
8	0	0	0	0	1	0	0	0	0	0	0 0 1	1	0	0
9	1	(cy,0)										11		
10	0	0	0	0	1	0	0	0	0	0	- - -	0	0	1
11	0	0	0	0	0	1	0	0	0	0	0 0 1	1	0	0
12	1	(cy,0)										14		
13	0	0	0	0	0	1	0	0	0	0	- - -	0	0	1
14	0	0	0	0	0	0	1	0	0	0	0 0 1	1	0	0
15	1	(cy,0)										17		
16	0	0	0	0	0	0	1	0	0	0	- - -	0	0	1
17	0	0	0	0	0	0	0	1	0	0	0 0 1	1	0	0
18	1	(cy,0)										20		
19	0	0	0	0	0	0	0	1	0	0	- - -	0	0	1
20	0	0	0	0	0	0	0	0	1	0	0 0 1	1	0	0
21	1	(cy,0)										23		
22	0	0	0	0	0	0	0	0	1	0	- - -	0	0	1
23	0	0	0	0	0	0	0	0	0	1	0 0 0	1	0	0

Exercise 14.13 FIFO implementation.

Based on the VHDL description of the FIFO operation, the data section must be able to store an input to any of the queue positions (following a given insertion order). This feature can be obtained with a broadcast of the input signal to all memory elements, and the control over the write enable signal of each element. Another required property is the ability to shift the information one position when a FIFO element is read. The proposed data section is shown in Figure **??**. The *shift* signal is used in the read operation, to remove one element from the queue. The vector of control signals $(w_{n-1}, ..., w_1, w_0)$ is used to select one of the registers that will receive the input element during a write operation.

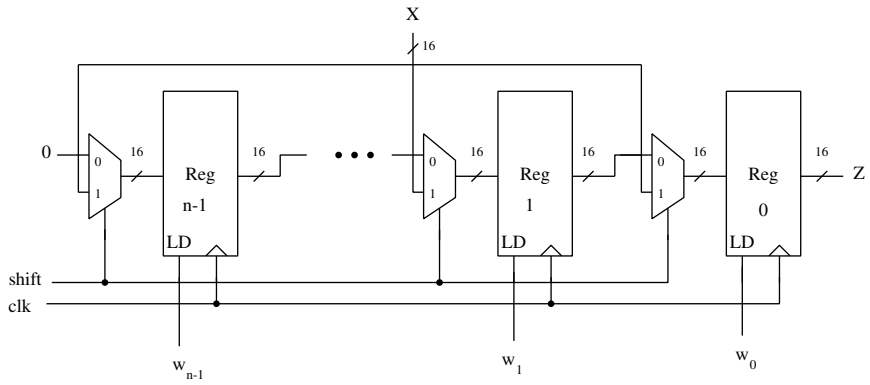


Figure 14.12: Data section for the FIFO memory - Exercise 14.13

The control section for the FIFO receives the Rd and Wr signals from the FIFO interface and generate the output signal *Empty* and *Full*, besides the internal control signals $(w_{n-1}, ..., w_1, w_0)$. The network for the control section is shown in Figure **??**. It corresponds to a bidirectional shift register with parallel output and some additional control for the cases when the FIFO is empty or full. Some aspects of the design are important:

1. a *reset* signal was introduced to initialize the state of the FIFO control section, making $w_0 = 1$.
2. instead of having a reference to the position of the last inserted element, we opted to have a reference to the next free position for writing (w_i).
3. when the FIFO is full $(w_{n-1}, ..., w_1, w_0) = (0, ..., 0, 0)$, and $Full = 1$.
4. the empty condition is true when $w_0 = 1$.

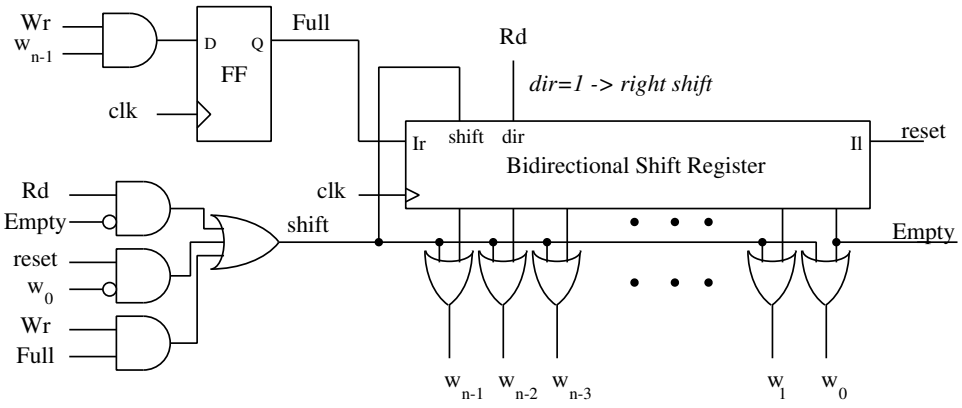


Figure 14.13: Control section for the FIFO memory - Exercise 14.13