Integrated Device Technology, Inc.

DEPTH EXPANSION OF IDT'S SYNCHRONOUS FIFOs USING THE RING COUNTER APPROACH

APPLICATION NOTE AN-69

By Bhanu V. R. Nanduri

INTRODUCTION

This application note describes a concise design approach to expand in depth IDT's synchronous FIFOs. As an example we will use the IDT72211 synchronous FIFO to demonstrate depth expansion using the ring counter approach. The discussion in this paper is however applicable to expand in depth all synchronous FIFOs from IDT. The first part of this paper discusses how one can expand in depth two IDT72211 synchronous FIFOs with the help of two industry standard PALs the 20R8s. The second part of this note discusses how one can accomplish depth expansion using four IDT72211s using three 20R8s, that is identical in pin out to a single large four-deep IDT72211. All PALs were programmed using "Capilano Computing Systems LPLC™ PLD software package" and a copy of the PAL programs is presented at the end of this paper. As the density and speed of industry standard PALs increases, it should be possible to expand in depth more of these FIFOs with a fewer number of PALs and with minimal loss in performance.

Traditionally, asynchronous FIFOs have been expanded in depth with the help of the $\overline{XI}$ and $\overline{XO}$ pins provided on these FIFOs. These FIFOs are cascaded in depth by connecting the $\overline{XO}$ pin of the present FIFO to the $\overline{XI}$ pin of the next FIFO in the cascade. This procedure is carried out for all the FIFOs in the cascade until the last FIFO in the cascade is reached. The $\overline{XO}$ pin of the last FIFO in the chain is connected to the $\overline{XI}$ pin of the first FIFO to complete the ring. A pin called the first load pin ($\overline{FL}$) on one of the FIFOs is grounded to indicate that the first write and read operations will begin in that FIFO. The $\overline{FL}$ pins of all the other FIFOs in the cascade are however tied to Vcc. The Full flags of all the FIFOs are ORed to provide a composite Full flag, similarly the Empty flags of all the FIFOs are ORed to provide a composite Empty flag. The user is urged to refer to IDTs technical note TN-09 "Cascading FIFOs or FIFO Modules" for further information on expanding asynchronous FIFOs.

The IDT72215 and the IDT72225 Synchronous FIFOs are provided with two pairs of $\overline{XI}$ and $\overline{XO}$ pins to assist the user in expanding these FIFOs using the daisy chain arrangement. One pair of $\overline{XI}$ and $\overline{XO}$ pins are used to synchronize write operations in the cascade and are controlled by the WCLK, while the other pair of $\overline{XI}$ and $\overline{XO}$ pins are used to synchronize read operations in the cascade and are controlled by the RCLK. Because of the $\overline{XI}$ and $\overline{XO}$ pins on the IDT72215 and the IDT72225 the daisy chain arrangement used in asynchronous FIFO expansion can be likewise used.

DEPTH EXPANSION OF IDT72211 SYNCFIFOS™

In this section we shall describe how to expand in depth the IDT72211 synchronous FIFO without any $\overline{XI}$ and $\overline{XO}$ pins. This discussion as stated earlier is applicable for expanding in depth all synchronous FIFOs from IDT. The expansion we will describe uses one ring counter to supervise the write operations and another ring counter to supervise the read operations. As a first step let us expand in depth two IDT72211s. Figure 1 illustrates the arrangement to carry out this two-deep depth expansion, the PAL labelled WRENCNTLR supervises the write operations whereas the PAL labelled RENCNTLR supervises the read operations. In addition to carrying out the write and the read operations these PALs also generate the Almost Full, Almost Empty, Full and Empty Flags for the expanded FIFO. Write operations in the IDT72211 are carried out when the two write enables are asserted and occur synchronously on the rising edge of the WCLK. Similarly the read operations are carried out when the two read enables are asserted and occur synchronously on the rising edge of the RCLK. The RCLK and the WCLK inputs of the FIFO can be tied together and connected to a system clock or they can be separately connected to two separate system clocks. The two system clocks can be of either the same frequency or different frequencies as long as the minimum clock period of the device is not violated. IDT synchronous FIFOs have a one deep pipelined architecture and because of this there will be a read latency of one cycle after reset.

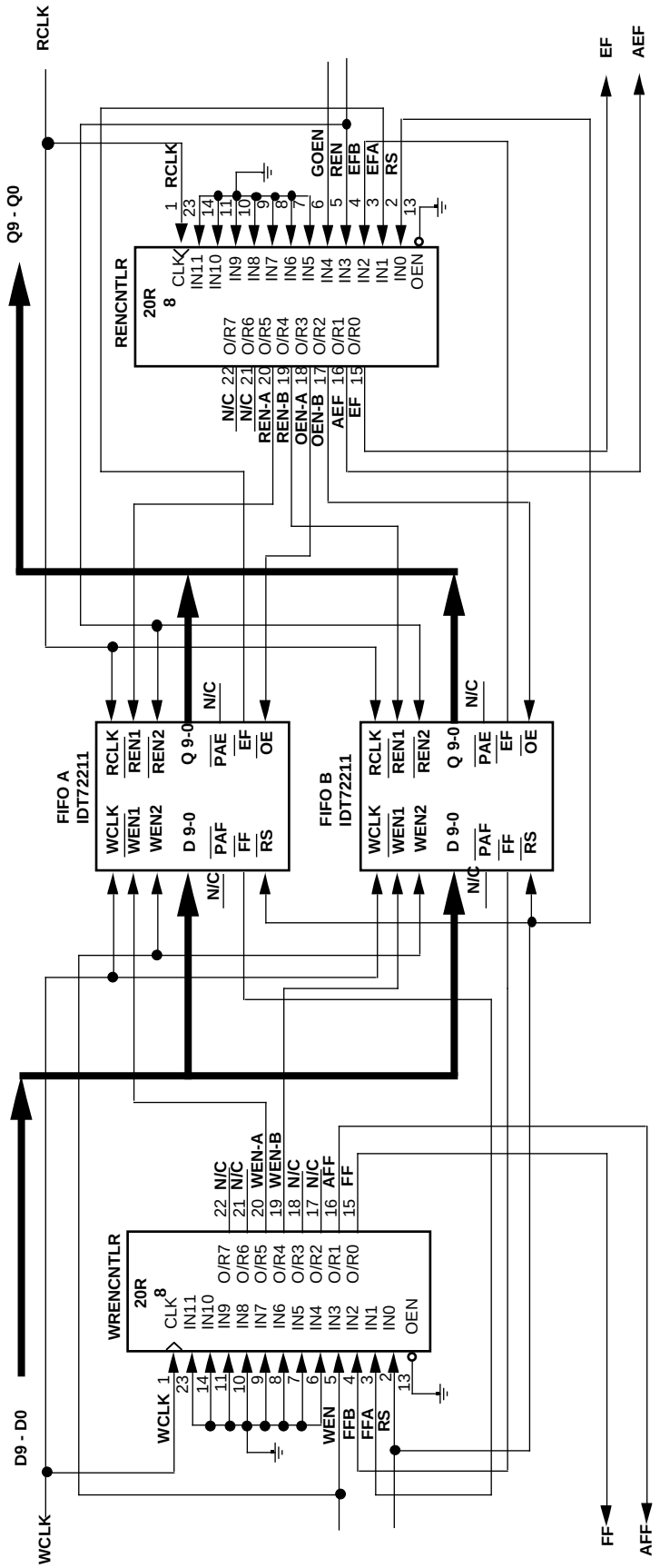

Figure 1.

2702 drw 01

DESCRIPTION OF THE WRITE ENABLE CONTROLLER (WRENCNTLR)

The write enable controller (WRENCNTLR) is implemented using a fast (t_{pd} = 7.5ns) industry standard 20R8 PAL. Write operations start when the user asserts the write clock and the global write enable "WEN" input to the WRENCNTLR. Write operations are performed synchronously and occur on the rising edge of the WCLK. After reset, the first write operation starts in FIFO A, the second in FIFO B, the third in FIFO A and the fourth in FIFO B. This pattern is repeated for all subsequent write operations. WEN2 inputs of FIFOs A and B are tied to the global write enable "WEN" as shown. The WRENCNTLR provides separate WEN1 strobes to each FIFO. These strobes have been labelled WEN_A and WEN_B. A user can stop the write operations to the FIFO by deasserting the global write enable (WEN) input. This will ensure that the WRENCNTLR will stall the present WEN1 signal and not generate the next WEN1 pulse. Care has been taken to ensure that even if the user asserts the global write enable (WEN) input to the WRENCNTLR, the WRENCNTLR will not generate the next WEN1 pulse when the composite Full flag is asserted. The composite Full flag will be asserted when all the FIFOs in the cascade have their Full flags asserted. The WRENCNTLR thus remains in the same state until the composite Full flag is deasserted ensuring that the same write sequence is maintained. The WRENCNTLR will generate an Almost Full flag only when any one of the FIFOs asserts a Full flag. This gives the user a prior warning of one word that he is approaching the end of his data queue. All outputs generated by the WRENCNTLR occur synchronously and are asserted and deasserted on the rising edge of the WCLK.

The FIFOs will each assert their Full flags after 512 words have been written into them but only if no read operations had occurred during this period. A write operation that is performed while the Full flag is asserted is ignored by the FIFOs. Internal to the IDT72211 and transparent to the user is a write inhibit signal that is generated when the FIFO is full. This signal blocks out any subsequent write operations that occur, ensuring that the state of the internal write pointer is not altered and the data in the FIFO is not overwritten.

DESCRIPTION OF THE READ ENABLE CONTROLLER (RENCNTLR)

The read enable controller (RENCNTLR) like the write enable controller is implemented using a fast industry standard PAL, the 20R8. Read operations start when the user enables the read clock (RCLK) and asserts the active low global read enable input ($\overline{REN}$) and the active low global output enable input ($\overline{GOEN}$) to the RENCNTLR. In our design read sequences are identical to write sequences. Hence after reset the first read starts in FIFO A, the second in FIFO B, the third in FIFO A and the fourth in FIFO B. This pattern is repeated for all subsequent read operations. The $\overline{REN2}$ inputs of FIFOs labelled A and B are tied to the global read enable "$\overline{REN}$" as shown. The RENCNTLR provides separate $\overline{REN1}$ strobes to each FIFO, these $\overline{REN1}$ signals have been labelled REN_A and REN_B, in addition the RENCNTLR provides separate output enables labelled OEN_A and OEN_B. OEN_A follows REN_A and OEN_B follows REN_B. A user wishing to stop the read operations can do so by deasserting the global read enable ($\overline{REN}$) input to the RENCNTLR. This will ensure that the RENCNTLR will stall the present $\overline{REN1}$ signal and not generate the next $\overline{REN1}$ pulse. Care has been taken to ensure that even if the user asserts the global read enable ($\overline{REN}$) to the RENCNTLR, the RENCNTLR will not generate the next $\overline{REN1}$ pulse if the composite Empty flag is asserted. The RENCNTLR thus remains in the same state until the composite Empty flag is deasserted thus ensuring that the same read sequence is maintained. The RENCNTLR will generate an Almost Empty flag only when one of the FIFOs asserts its Empty flag. This give the user a prior warning of one word that he is approaching the end of his data queue. The RENCNTLR also generates the composite Empty flag when both the FIFOs in the cascade assert their Empty flags. All outputs generated by the RENCNTLR occur synchronously and are asserted and deasserted on the rising edge of the RCLK.

The IDT72211 synchronous FIFO is provided with a transparent output register on the read port. This register is loaded with a new word once every read cycle (provided the FIFO is not empty). This register puts its contents on the output bus only when the output enable input ($\overline{OE}$) is asserted. The function of this output register is to allow the user multiple reads of the same word without incrementing the internal read pointer. A user interested in reading the same word multiple times without altering the state of the internal read pointer can do so by disabling the $\overline{REN1}$ and $\overline{REN2}$ inputs to the FIFO while asserting its output enable. The user can then sample the output pins of the FIFO once every read clock cycle multiple times. The user can also skip sampling certain data by deasserting the output enable ($\overline{OE}$) while asserting the read enables ($\overline{REN1}$) and ($\overline{REN2}$).

A user wishing to read the same word in our expanded FIFO without incrementing the read pointer can do so by deasserting the global read enable ($\overline{REN}$) input while asserting the global output enable ($\overline{GOEN}$) of the RENCNTLR. This operation stalls the read operations and the data in the output register can then be sampled. The user can then sample this output once every read clock cycle for multiple read cycles. Since the IDT SyncFIFOs have a one deep pipelined architecture, depth expansion as described in this section will result in a read latency of one cycle after reset.

The FIFOs each assert their respective Empty flags after 512 words have been read from them and only if no write operations had occurred during this period. A read operation that is performed while an Empty flag is asserted is ignored by the FIFO, it will still however supply the last word read from the FIFO. Internal to the IDT72211 and transparent to the user is a read inhibit signal that is generated when the FIFO is empty. This signal, analogous to the write inhibit signal, is used to block out any subsequent read operations that occur, ensuring that the state of the internal read pointer is not altered and data in the FIFO's RAM array is not re-read.

TIMING REQUIREMENTS

Figure 2. and Figure 3. illustrate the timing waveforms for the write and read operations. The timing parameters for the Synchronous FIFO expansion are shown in Table 1. To operate the 15ns IDT72211 Synchronous FIFO we would need a 20R8 PAL whose maximum t_{CO} is 4ns. As these devices are not yet available in these speed grades we have to operate our 15ns FIFOs with a clock period of 17.5ns minimum. The reason for this is that the RENCNTLR asserts the read and output enable signals on the rising edge of the RCLK but with a maximum delay of 6.5ns induced by the PAL. The IDT72211 has a t_{OE} specified at 8ns maximum, this implies that 14.5ns ($t_{CO} + t_{OE} = 6.5 + 8$ns) after the rising edge of the RCLK, data will be available for sampling. Assuming the sampling side samples the output of the FIFOs also on the rising edge of the RCLK, this leaves only 0.5ns ($t_{CLK} - t_{CO} - t_{OE} = 15 - 14.5$ns) before the next arrival of a positive edge on the RCLK. This duration may be too short for the sampling side's set-up time requirements. Therefore if we use the RCLK period of 17.5ns, this would give the sampling side a data set-up time of 3ns, which is a reasonable data set-up time. The user is urged to refer to the IDT72211 and the 20R8 PAL's data sheets for more information on the timing requirements.

Parameter Symbol	Parameter Description	Min.	Max.
t_{CO}	PAL's Clock to output delay	3ns	6.5ns
t_S	PAL's Set-up time to clock	7ns	—
t_{RSS}	FIFO's reset set-up time	10ns	—
t_{ENS}	FIFO's enable set-up time	4ns	—
t_{ENH}	FIFO's enable hold time	1ns	—
t_{DS}	FIFO's data set-up time	4ns	—
t_{DH}	FIFO's data hold time	1ns	—
t_{OE}	FIFO's output enable time	—	8ns
t_{OHZ}	FIFO's output disable time	1ns	8ns
t_{CLK}	FIFO and PAL Clock period	17.5ns	—

2702 tbl 01

Table 1.

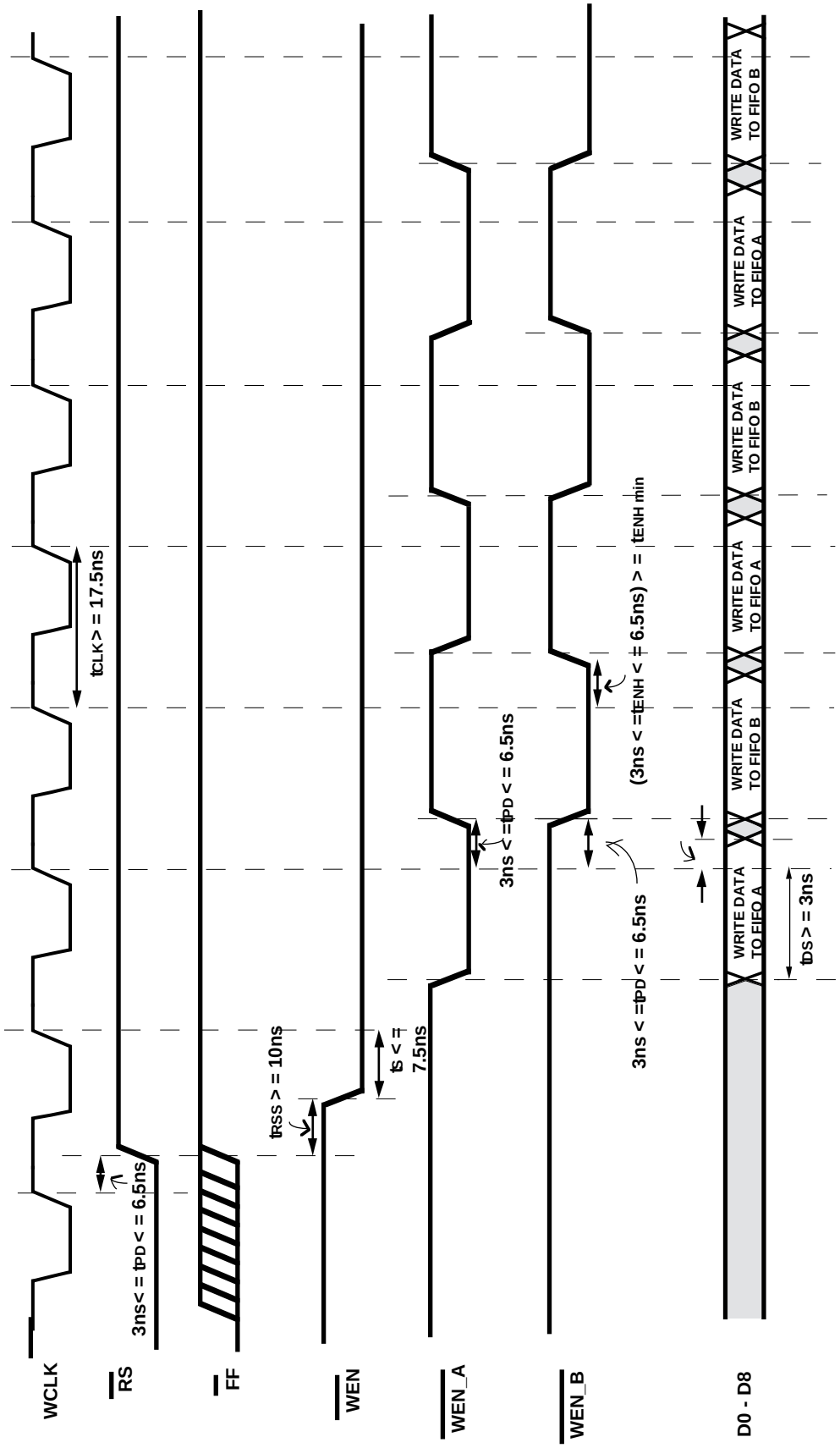

Figure 2. Write Cycle Timing

2702 drw 02

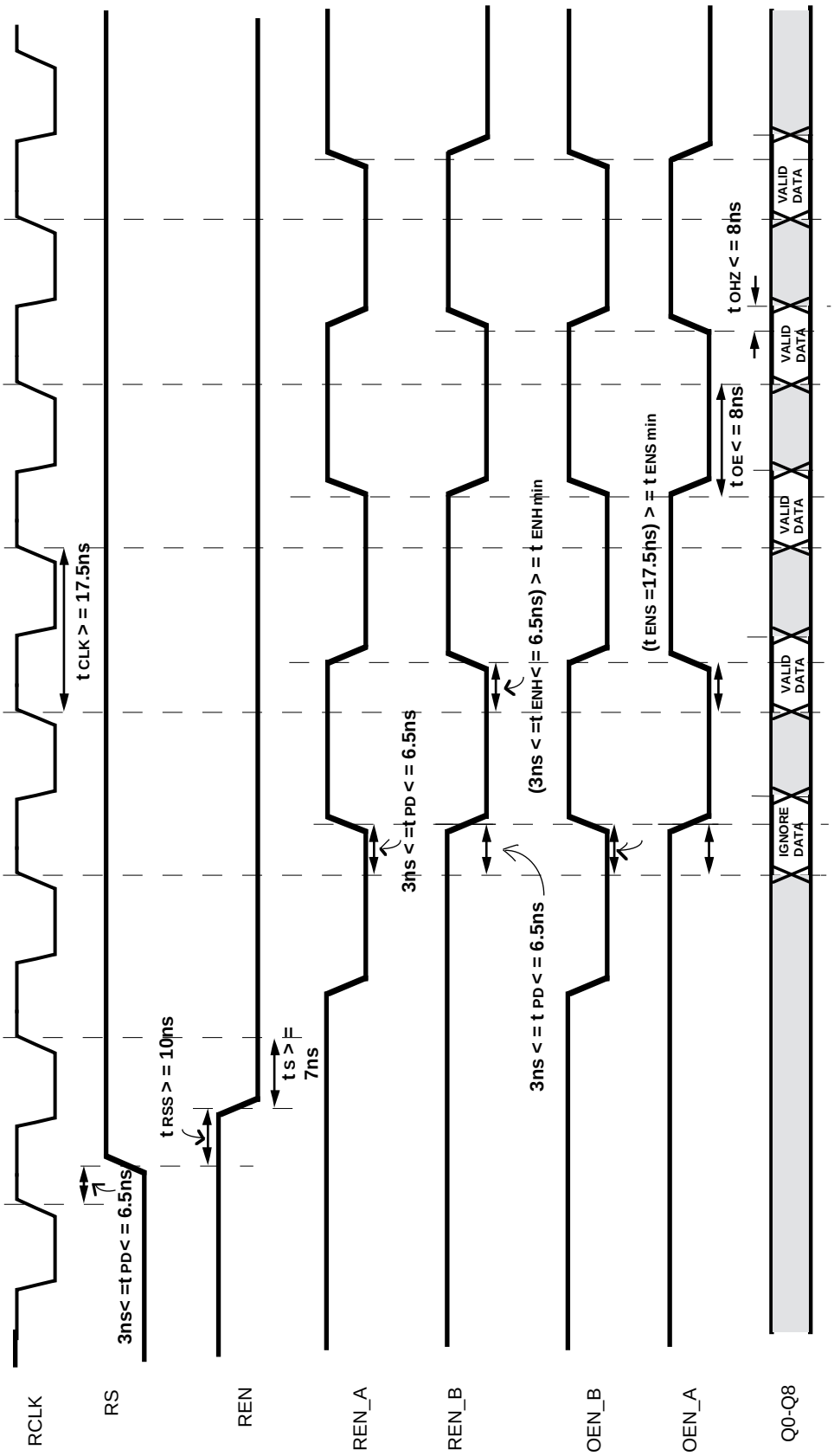

Figure 3. Read Cycle Timing

2702 drw

FOUR DEEP DEPTH EXPANSION OF THE IDT72211 SYNCFIFOS ™

The discussion on the two deep depth expansion using the ring counter approach, described earlier is also applicable in this section for a four-deep depth expansion. Figure 4 illustrates the four-deep depth expansion. Write operations will start when the user asserts the write clock (WCLK) and the active high global write enable "WEN" input to the WRENCNTLR. Write operations are performed synchronously and occur on the rising edge of the WCLK. After reset, the first write operation starts in FIFO A, the second in FIFO B, the third in FIFO C and the fourth in FIFO D. This pattern is repeated for all subsequent write operations. WEN2 inputs of FIFOs A, B, C and D are tied to the global write enable "WEN" as shown. The WRENCNTLR provides separate $\overline{\text{WEN1}}$ strobes to each FIFO. These strobes have been labelled WEN_A, WEN_B, WEN_C and WEN_D. A user can stop the write operations to the FIFO by deasserting the global write enable (WEN) input. This will ensure that the WRENCNTLR will stall the present $\overline{\text{WEN1}}$ signal and not generate the next $\overline{\text{WEN1}}$ pulse. Even if the user asserts the global write enable (WEN) input to the WRENCNTLR, the WRENCNTLR will not generate the next $\overline{\text{WEN1}}$ pulse while the composite Full flag is asserted. The WRENCNTLR thus remains in the same state until the composite Full flag is deasserted ensuring that the same write sequence is maintained. The WRENCNTLR will generate an Almost Full flag only when any one of the FIFOs asserts a Full flag. This gives the user a prior warning of three words that he is approaching the end of his data queue. The WRENCNTLR will also generate a composite Full flag when all the FIFOs assert their Full flags. All outputs generated by the WRENCNTLR occur synchronously and are asserted and deasserted on the rising edge of the WCLK.

In the four-deep depth expanded FIFO design we need to generate four read enables and four output enables to perform the read operations successfully. In addition we need to generate an Almost Empty Flag and a composite Empty Flag. The 20R8 PALs that we are using have a maximum of eight outputs only, hence this implies that we need to use at least two 20R8s to perform the read operations successfully. As the density and the functional capabilities of PALs increases it should be possible to implement the RENCNTLR using a single PAL.

Read operations start when the user enables the read clock (RCLK) and asserts the active low global read enable input ($\overline{\text{REN}}$) to the RENCNTLR, in addition the user will have to assert the global ($\overline{\text{GOEN}}$) input to the OENCNTLR. The OENCNTLR generates the separate output enable inputs to FIFOs labelled A, B, C and D. In our design read sequences are identical to write sequences. Hence after reset the first read starts in FIFO A, the second in FIFO B, the third in FIFO C and the fourth in FIFO D. This pattern is repeated for all subsequent read operations. The $\overline{\text{REN2}}$ inputs of FIFOs labelled A, B, C and D are tied to the global read enable "$\overline{\text{REN}}$" as shown. The RENCNTLR provides a separate $\overline{\text{REN1}}$ strobe to each FIFO, these strobes have been labelled REN_A, REN_B, REN_C, and REN_D. A user wishing to stop the read operations can do so by deasserting the global read enable ($\overline{\text{REN}}$) input to the RENCNTLR. This will ensure that the RENCNTLR will stall the present $\overline{\text{REN1}}$ signal and not generate the next $\overline{\text{REN1}}$ pulse. Even if the user asserts the global read enable ($\overline{\text{REN}}$) to the RENCNTLR, the RENCNTLR will not generate the next $\overline{\text{REN1}}$ pulse if the present FIFO in the cascade has its Empty flag asserted. The RENCNTLR thus remains in the same state until the composite Empty flag is deasserted thus ensuring that the same read sequence is maintained. The RENCNTLR will generate an Almost Empty flag only when one of the FIFOs asserts its Empty flag. This give the user a prior warning of three words that he is approaching the end of his data queue. The RENCNTLR also generates the composite Empty flag when all the FIFOs in the cascade assert their Empty flags. All outputs generated by the RENCNTLR occur synchronously and are asserted and deasserted on the rising edge of the RCLK. The OENCNTLR generates separate output enables to the FIFOs, these separate output enables OEN_A, OEN_B, OEN_C and OEN_D follow the read enables REN_A, REN_B, REN_C and REN_D.

A user wishing to read the same word in our expanded FIFO without incrementing the read pointer can do so by deasserting the global read enable ($\overline{\text{REN}}$) input to the RENCNTLR and asserting the global output enable ($\overline{\text{GOEN}}$). This operation stalls the read operation but will assert the output enable of one of the FIFOs which contains the data to be sampled. The user can then sample this output once every read clock cycle for multiple read cycles. Because of this one deep pipelined architecture there occurs a read latency of one cycle after reset in our four deep FIFO expansion.

SUMMARY

Expanding Synchronous FIFOs in depth can be very easily accomplished as discussed in this paper with minimum external logic. As the density, functionality and speed of industry standard PALs increases, it will be possible to expand these FIFOs to even larger depths without incurring any loss in performance.

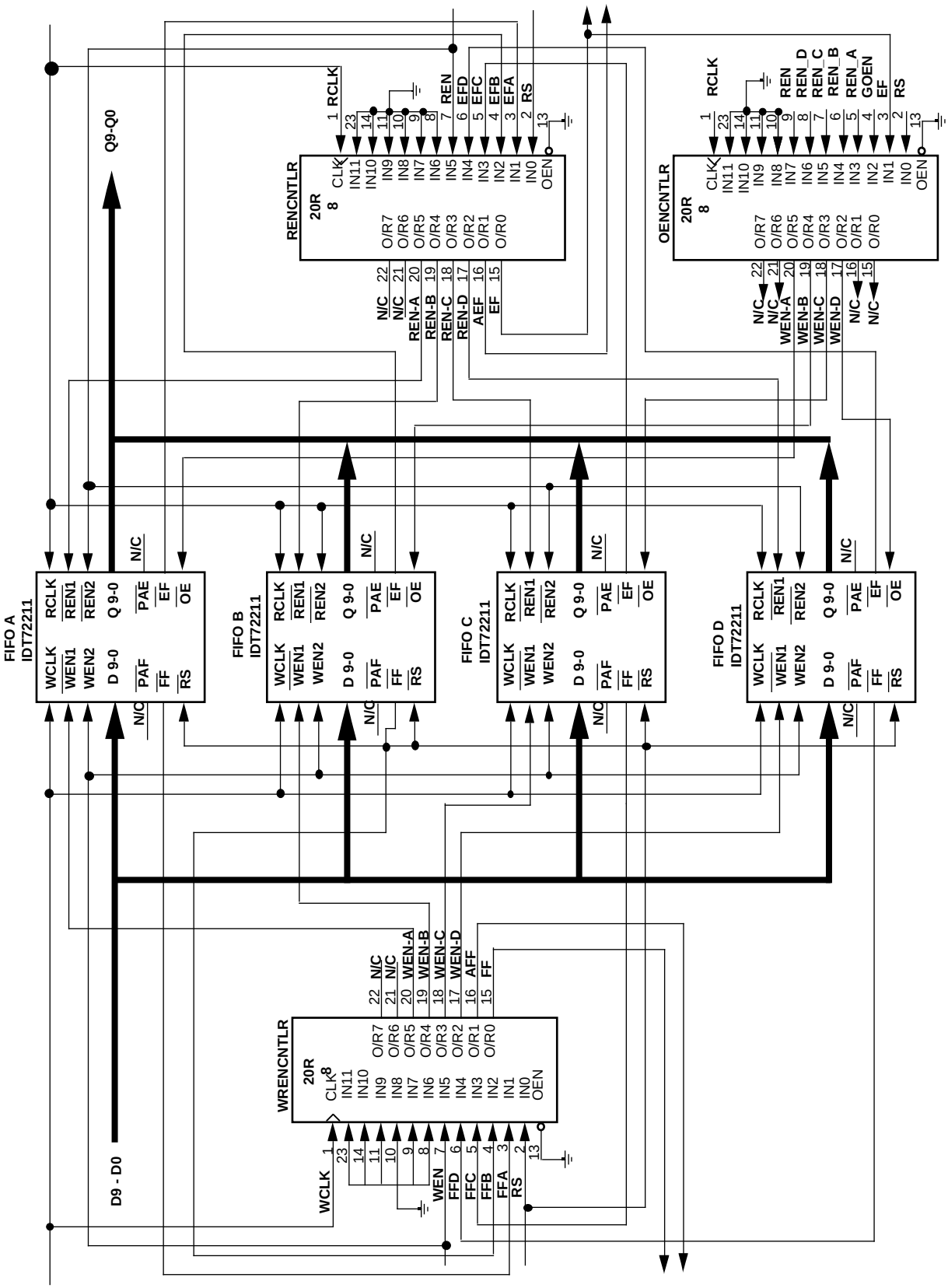

Figure 4.

2702 drw 04

```
{AUTHOR:       BHANU V. R. NANDURI
 COMPANY:      INTEGRATED DEVICE TECHNOLOGY
 DATE:         01/24/89
 }
 MODULE X2WENCNTLR;          { THIS PAL IS USED TO CONTROL THE
                               WRITE OPERATIONS IN A TWO DEEP
                               X9 OR X8 FIFO EXPANSION SCHEME }
 TITLE X2WENCNTLR;
 TYPE MMI 20R8;

 INPUTS;

 {WCLK         NODE[PIN1];}   {WCLK INPUT}
 RS            NODE[PIN2];    {SYNCHRONOUS RESET INPUT}
 FFA           NODE[PIN3];
 FFB           NODE[PIN4];
 WEN           NODE[PIN5];
 C0            NODE[PIN22];
 WEN_A         NODE[PIN20];
 WEN_B         NODE[PIN19];
 FF            NODE[PIN15];

 OUTPUTS;

 WEN_A         NODE[PIN20];   {WEN1 TO FIFO A}
 WEN_B         NODE[PIN19];   {WEN1 TO FIFO B}
 AFF           NODE[PIN16];   {ALMOST FULL FLAG FOR THE EXPANSION}
 FF            NODE[PIN15];   {FULL FLAG FOR THE EXPANSION}
 C0            NODE[PIN22];   {C0 IS BIT 0 OF THE TWO BIT COUNTER}

 TERMS;

 C0 NOT  :=    RS AND !WEN AND !WEN_A AND FF AND C0 OR
               RS AND !WEN AND !WEN_B AND !FF AND !C0 OR
               RS AND !WEN AND WEN_A AND WEN_B AND !FF AND !C0 OR
               RS AND WEN AND !WEN_B AND !C0 OR
               RS AND WEN AND WEN_A AND WEN_B AND !C0 OR
               RS AND !WEN AND WEN_A AND WEN_B AND !C0;

 WEN_A NOT:= RS AND !WEN AND WEN_B AND WEN_A AND C0 OR
               RS AND !WEN AND !WEN_B AND FF AND !C0 OR
               RS AND !WEN AND !WEN_A AND !FF AND C0;

 WEN_B NOT:= RS AND !WEN AND WEN_B AND WEN_A AND !C0 OR
               RS AND !WEN AND !WEN_A AND FF AND C0 OR
               RS AND !WEN AND !WEN_B AND !FF AND !C0;

 AFF NOT  := !FFA AND FFB OR
               FFA AND !FFB OR
               !FFA AND !FFB;

 FF NOT   := !FFA AND !FFB;
 END;
 END X2WENCNTLR.
```

```
{AUTHOR:       BHANU V. R. NANDURI
 COMPANY:      INTEGRATED DEVICE TECHNOLOGY
 DATE:         01/24/89
 }
 MODULE X2RENCNTLR;          { THIS PAL IS USED TO CONTROL THE
                               READ OPERATIONS IN A TWO DEEP
                               X9 OR X8 FIFO EXPANSION SCHEME }
 TITLE X2RENCNTLR;
 TYPE MMI 20R8;

 INPUTS;

 {WCLK         NODE[PIN1];}  {RCLK INPUT}
 RS            NODE[PIN2];   {SYNCHRONOUS RESET INPUT}
 EFA           NODE[PIN3];
 EFB           NODE[PIN4];
 REN           NODE[PIN5];
 OEN           NODE[PIN6];
 C0            NODE[PIN22];
 REN_A         NODE[PIN20];
 REN_B         NODE[PIN19];
 OEN_A         NODE[PIN18];
 OEN_B         NODE[PIN17];
 EF            NODE[PIN15];

 OUTPUTS;

 REN_A         NODE[PIN20];  {REN1 TO FIFO A}
 REN_B         NODE[PIN19];  {REN1 TO FIFO B}
 OEN_A         NODE[PIN18];  {OEN1 TO FIFO A}
 OEN_B         NODE[PIN17];  {OEN1 TO FIFO B}
 AEF           NODE[PIN16];  {ALMOST EMPTY FLAG FOR THE EXPANSION}
 EF            NODE[PIN15];  {EMPTY FLAG FOR THE EXPANSION}
 C0            NODE[PIN22];  {C0 IS BIT 0 OF THE TWO BIT COUNTER}

 TERMS;

 C0 NOT  :=    RS AND !REN AND !REN_A AND EF AND C0 OR
               RS AND !REN AND !REN_B AND !EF AND !C0 OR
               RS AND REN AND !REN_B AND !C0 OR
               RS AND REN AND REN_A AND REN_B AND !C0 OR
               RS AND !REN AND REN_A AND REN_B AND !C0;

 REN_A NOT :=  RS AND !REN AND REN_B AND REN_A AND C0 OR
               RS AND !REN AND !REN_B AND EF AND !C0 OR
               RS AND !REN AND !REN_A AND !EF AND C0;

 REN_B NOT :=  RS AND !REN AND  REN_B AND REN_A AND !C0 OR
               RS AND !REN AND !REN_A AND EF AND C0 OR
               RS AND !REN AND !REN_B AND !EF AND !C0;

 OEN_A NOT :=  RS AND !OEN AND !OEN_A AND REN_A AND REN_B AND C0 OR
               RS AND !OEN AND !OEN_B AND !REN_A AND EF AND C0 OR
               RS AND !OEN AND OEN_A AND OEN_B AND REN_A AND REN_B AND C0 OR
               RS AND !OEN AND OEN_A AND OEN_B AND !REN_A AND EF AND C0;
```

```
 OEN_B NOT := RS AND !OEN AND !OEN_B AND REN_A AND REN_B AND !C0 OR
              RS AND !OEN AND !OEN_A AND !REN_B AND EF AND !C0 OR
              RS AND !OEN AND OEN_A AND OEN_B AND REN_A AND REN_B AND !C0 OR
              RS AND !OEN AND OEN_A AND OEN_B AND !REN_B AND EF AND !C0;

 AEF NOT  := !EFA AND EFB OR
              EFA AND !EFB OR
             !EFA AND !EFB;

 EF NOT   := !EFA AND !EFB;

 END;
 END X2RENCNTLR.




{AUTHOR:       BHANU V. R. NANDURI
 COMPANY:      INTEGRATED DEVICE TECHNOLOGY
 DATE:         01/24/89
 }
 MODULE WENCNTLR;            { THIS PAL IS USED TO CONTROL THE
                               WRITE OPERATIONS IN A FOUR DEEP
                               X9 OR X8 FIFO EXPANSION SCHEME }
 TITLE WENCNTLR;
 TYPE MMI 20R8;

 INPUTS;

 {WCLK         NODE[PIN1];}   {WCLK INPUT}
 RS            NODE[PIN2];    {SYNCHRONOUS RESET INPUT}
 FFA           NODE[PIN3];
 FFB           NODE[PIN4];
 FFC           NODE[PIN5];
 FFD           NODE[PIN6];
 WEN           NODE[PIN7];
 C0            NODE[PIN22];
 C1            NODE[PIN21];
 WEN_A         NODE[PIN20];
 WEN_B         NODE[PIN19];
 WEN_C         NODE[PIN18];
 WEN_D         NODE[PIN17];
 FF            NODE[PIN15];

 OUTPUTS;

 WEN_A         NODE[PIN20];   {WEN1 TO FIFO A}
 WEN_B         NODE[PIN19];   {WEN1 TO FIFO B}
 WEN_C         NODE[PIN18];   {WEN1 TO FIFO C}
 WEN_D         NODE[PIN17];   {WEN1 TO FIFO D}
 AFF           NODE[PIN16];   {ALMOST FULL FLAG FOR THE EXPANSION}
 FF            NODE[PIN15];   {FULL FLAG FOR THE EXPANSION}
 C0            NODE[PIN22];   {C0 IS BIT 0 OF THE TWO BIT COUNTER}
 C1            NODE[PIN21];   {C1 IS BIT 1 OF THE TWO BIT COUNTER}
```

```
TERMS;

C0 NOT  :=    RS AND !WEN AND !WEN_A AND FF AND C0 OR
              RS AND !WEN AND !WEN_C AND FF AND C0 OR
              RS AND !WEN AND !WEN_B AND !FF AND !C0 OR
              RS AND !WEN AND !WEN_D AND !FF AND !C0 OR
              RS AND WEN AND !WEN_B AND !C0 OR
              RS AND WEN AND !WEN_D AND !C0 OR
              RS AND WEN AND WEN_A AND WEN_B AND WEN_C AND
              WEN_D AND !C0 OR
              RS AND !WEN AND WEN_A AND WEN_B AND WEN_C AND WEN_D AND !C0;

C1 NOT  :=    RS AND !WEN AND !WEN_B AND FF AND C1 OR
              RS AND !WEN AND !WEN_C AND FF AND !C1 OR
              RS AND !WEN AND !WEN_C AND !FF AND !C1 OR
              RS AND !WEN AND !WEN_D AND !FF AND !C1 OR
              RS AND WEN AND !WEN_C AND !C1 OR
              RS AND WEN AND !WEN_D AND !C1 OR
              RS AND WEN AND WEN_A AND WEN_B AND WEN_C AND
              WEN_D AND !C1 OR
              RS AND !WEN AND WEN_A AND WEN_B AND WEN_C AND WEN_D AND !C1;

WEN_A NOT := RS AND !WEN AND WEN_D AND WEN_C AND WEN_B AND WEN_A AND
             C0 AND C1 OR
             RS AND !WEN AND !WEN_D AND FF AND !C0 AND !C1 OR
             RS AND !WEN AND !WEN_A AND !FF AND C0 AND C1;

WEN_B NOT := RS AND !WEN AND WEN_D AND WEN_C AND WEN_B AND WEN_A AND
             !C0 AND C1 OR
             RS AND !WEN AND !WEN_A AND FF AND C0 AND C1 OR
             RS AND !WEN AND !WEN_B AND !FF AND !C0 AND C1;

WEN_C NOT := RS AND !WEN AND WEN_D AND WEN_C AND WEN_B AND WEN_A AND
             C0 AND !C1 OR
             RS AND !WEN AND !WEN_B AND FF AND !C0 AND C1 OR
             RS AND !WEN AND !WEN_C AND !FF AND C0 AND !C1;

WEN_D NOT := RS AND !WEN AND WEN_D AND WEN_C AND WEN_B AND WEN_A AND
             !C0 AND !C1 OR
             RS AND !WEN AND !WEN_C AND FF AND C0 AND !C1 OR
             RS AND !WEN AND !WEN_D AND !FF AND !C0 AND !C1;

AFF NOT  := !FFA AND FFB AND FFC AND FFD OR
            FFA AND !FFB AND FFC AND FFD OR
            FFA AND FFB AND !FFC AND FFD OR
            FFA AND FFB AND FFC AND !FFD OR
            !FFA AND !FFB AND !FFC AND !FFD;

FF NOT   := !FFA AND !FFB AND !FFC AND !FFD;
END;
END WENCNTLR.
```

```
{AUTHOR:       BHANU V. R. NANDURI
 COMPANY:      INTEGRATED DEVICE TECHNOLOGY
 DATE:         01/24/89
 }
 MODULE RENCNTLR;            { THIS PAL IS USED TO CONTROL THE
                               READ OPERATIONS IN A FOUR DEEP
                               X9 OR X8 FIFO EXPANSION SCHEME }
 TITLE RENCNTLR;
 TYPE MMI 20R8;

 INPUTS;

 {WCLK         NODE[PIN1];}   {RCLK INPUT}
 RS            NODE[PIN2];    {SYNCHRONOUS RESET INPUT}
 EFA           NODE[PIN3];
 EFB           NODE[PIN4];
 EFC           NODE[PIN5];
 EFD           NODE[PIN6];
 REN           NODE[PIN7];
 C0            NODE[PIN22];
 C1            NODE[PIN21];
 REN_A         NODE[PIN20];
 REN_B         NODE[PIN19];
 REN_C         NODE[PIN18];
 REN_D         NODE[PIN17];
 EF            NODE[PIN15];

 OUTPUTS;

 REN_A         NODE[PIN20];   {REN1 TO FIFO A}
 REN_B         NODE[PIN19];   {REN1 TO FIFO B}
 REN_C         NODE[PIN18];   {REN1 TO FIFO C}
 REN_D         NODE[PIN17];   {REN1 TO FIFO D}
 AEF           NODE[PIN16];   {ALMOST EMPTY FLAG FOR THE EXPANSION}
 EF            NODE[PIN15];   {EMPTY FLAG FOR THE EXPANSION}
 C0            NODE[PIN22];   {C0 IS BIT 0 OF THE TWO BIT COUNTER}
 C1            NODE[PIN21];   {C1 IS BIT 1 OF THE TWO BIT COUNTER}

 TERMS;

 C0 NOT  :=    RS AND !REN AND !REN_A AND EF AND C0 OR
               RS AND !REN AND !REN_C AND EF AND C0 OR
               RS AND !REN AND !REN_B AND !EF AND !C0 OR
               RS AND !REN AND !REN_D AND !EF AND !C0 OR
               RS AND REN AND !REN_B AND !C0 OR
               RS AND REN AND !REN_D AND !C0 OR
               RS AND REN AND REN_A AND REN_B AND REN_C AND
               REN_D AND !C0 OR
               RS AND !REN AND REN_A AND REN_B AND REN_C AND REN_D AND !C0;
```

```
C1 NOT  :=    RS AND !REN AND !REN_B AND EF AND C1 OR
              RS AND !REN AND !REN_C AND EF AND !C1 OR
              RS AND !REN AND !REN_C AND !EF AND !C1 OR
              RS AND !REN AND !REN_D AND !EF AND !C1 OR
              RS AND REN AND !REN_C AND !C1 OR
              RS AND REN AND !REN_D AND !C1 OR
              RS AND REN AND REN_A AND REN_B AND REN_C AND
              REN_D AND !C1 OR
              RS AND !REN AND REN_A AND REN_B AND REN_C AND REN_D AND !C1;

REN_A NOT := RS AND !REN AND REN_D AND REN_C AND REN_B AND REN_A AND
             C0 AND C1 OR
             RS AND !REN AND !REN_D AND EF AND !C0 AND !C1 OR
             RS AND !REN AND !REN_A AND !EF AND C0 AND C1;

REN_B NOT := RS AND !REN AND REN_D AND REN_C AND REN_B AND REN_A AND
             !C0 AND C1 OR
             RS AND !REN AND !REN_A AND EF AND C0 AND C1 OR
             RS AND !REN AND !REN_B AND !EF AND !C0 AND C1;

REN_C NOT := RS AND !REN AND REN_D AND REN_C AND REN_B AND REN_A AND
             C0 AND !C1 OR
             RS AND !REN AND !REN_B AND EF AND !C0 AND C1 OR
             RS AND !REN AND !REN_C AND !EF AND C0 AND !C1;

REN_D NOT := RS AND !REN AND REN_D AND REN_C AND REN_B AND REN_A AND
             !C0 AND !C1 OR
             RS AND !REN AND !REN_C AND EF AND C0 AND !C1 OR
             RS AND !REN AND !REN_D AND !EF AND !C0 AND !C1;

AEF NOT  := !EFA AND EFB AND EFC AND EFD OR
            EFA AND !EFB AND EFC AND EFD OR
            EFA AND EFB AND !EFC AND EFD OR
            EFA AND EFB AND EFC AND !EFD OR
            !EFA AND !EFB AND !EFC AND !EFD;

EF NOT   := !EFA AND !EFB AND !EFC AND !EFD;
END;
END RENCNTLR.
```

```
{AUTHOR:       BHANU V. R. NANDURI
 COMPANY:      INTEGRATED DEVICE TECHNOLOGY
 DATE:         01/24/89
 }
 MODULE OENCNTLR;            { THIS PAL IS USED TO CONTROL THE
                        READ OPERATIONS IN A FOUR DEEP
                        X9 OR X8 FIFO EXPANSION SCHEME }
 TITLE OENCNTLR;
 TYPE MMI 20R8;

 INPUTS;

 {RCLK         NODE[PIN1];}   {RCLK INPUT}
 RS         NODE[PIN2];       {SYNCHRONOUS RESET INPUT}
 EF     NODE[PIN3];
 GOEN          NODE[PIN4];
 REN_A         NODE[PIN5];
 REN_B         NODE[PIN6];
 REN_C         NODE[PIN7];
 REN_D           NODE[PIN8];
 REN             NODE[PIN9];
 C0     NODE[PIN22];
 C1     NODE[PIN21];
 OEN_A           NODE[PIN20];
 OEN_B           NODE[PIN19];
 OEN_C           NODE[PIN18];
 OEN_D           NODE[PIN17];

 OUTPUTS;

 C0     NODE[PIN22];   {C0 IS BIT 0 OF THE TWO BIT COUNTER}
 C1     NODE[PIN21];   {C1 IS BIT 1 OF THE TWO BIT COUNTER}
 OEN_A          NODE[PIN20]; {OE TO FIFO A}
 OEN_B          NODE[PIN19]; {OE TO FIFO B}
 OEN_C          NODE[PIN18]; {OE TO FIFO C}
 OEN_D          NODE[PIN17]; {OE TO FIFO D}

 TERMS;

 C0 NOT  :=    RS AND !REN AND !REN_A AND EF AND C0 OR
       RS AND !REN AND !REN_C AND EF AND C0 OR
       RS AND !REN AND !REN_B AND !EF AND !C0 OR
       RS AND !REN AND !REN_D AND !EF AND !C0 OR
       RS AND REN AND !REN_B AND !C0 OR
       RS AND REN AND !REN_D AND !C0 OR
       RS AND REN AND REN_A AND REN_B AND REN_C AND
       REN_D AND !C0 OR
       RS AND !REN AND REN_A AND REN_B AND REN_C AND REN_D AND !C0;

 C1 NOT  :=    RS AND !REN AND !REN_B AND EF AND C1 OR
       RS AND !REN AND !REN_C AND EF AND !C1 OR
       RS AND !REN AND !REN_C AND !EF AND !C1 OR
       RS AND !REN AND !REN_D AND !EF AND !C1 OR
       RS AND REN AND !REN_C AND !C1 OR
       RS AND REN AND !REN_D AND !C1 OR
       RS AND REN AND REN_A AND REN_B AND REN_C AND
       REN_D AND !C1 OR
       RS AND !REN AND REN_A AND REN_B AND REN_C AND REN_D AND !C1;

 OEN_A NOT := RS AND !GOEN AND !OEN_A AND REN_D AND REN_C AND REN_B AND REN_A AND
```

```
         C0 AND C1 OR
         RS AND !GOEN AND !OEN_D AND !REN_A AND EF AND C0 AND C1 OR
         RS AND !GOEN AND !OEN_D AND !EF AND C0 AND C1 OR
         RS AND !GOEN AND !OEN_A AND !EF AND C0 AND C1 OR
         RS AND !GOEN AND OEN_D AND OEN_C AND OEN_B AND OEN_A AND
         !REN_A AND EF AND C0 AND C1;

OEN_B NOT := RS AND !GOEN AND !OEN_B AND REN_D AND REN_C AND REN_B AND REN_A AND
              !C0 AND C1 OR
         RS AND !GOEN AND !OEN_A AND !REN_B AND EF AND !C0 AND C1 OR
         RS AND !GOEN AND !OEN_A AND !EF AND !C0 AND C1 OR
         RS AND !GOEN AND !OEN_B AND !EF AND !C0 AND C1 OR
         RS AND !GOEN AND OEN_D AND OEN_C AND OEN_B AND OEN_A AND
         !REN_B AND EF AND !C0 AND C1;

OEN_C NOT := RS AND !GOEN AND !OEN_C AND REN_D AND REN_C AND REN_B AND REN_A AND
         C0 AND !C1 OR
         RS AND !GOEN AND !OEN_B AND !REN_C AND EF AND C0 AND !C1 OR
         RS AND !GOEN AND !OEN_B AND !EF AND C0 AND !C1 OR
         RS AND !GOEN AND !OEN_C AND !EF AND C0 AND !C1 OR
         RS AND !GOEN AND OEN_D AND OEN_C AND OEN_B AND OEN_A AND
         !REN_C AND EF AND C0 AND !C1;

OEN_D NOT := RS AND !GOEN AND !OEN_D AND REN_D AND REN_C AND REN_B AND REN_A AND
         !C0 AND !C1 OR
         RS AND !GOEN AND !OEN_C AND !REN_D AND EF AND !C0 AND !C1 OR
         RS AND !GOEN AND !OEN_C AND !EF AND !C0 AND !C1 OR
         RS AND !GOEN AND !OEN_D AND !EF AND !C0 AND !C1 OR
         RS AND !GOEN AND OEN_D AND OEN_C AND OEN_B AND OEN_A AND
         !REN_D AND EF AND !C0 AND !C1;

END;
END OENCNTLR.
```