

Design of cyclic code decade counter using K-Maps,
Schematic capture and simulation of counter in Quartus-
Altera software

LAB #1 EE 2160
SEPTEMBER 20, 2004

BY:
Narayanan Krishnamurthy

Purpose:

To design and simulate a 4-bit decade counter, that implements a cyclic code encoding for the 0-9 count (see table1 for the mapping of count to cyclic code). And to use the 2 such counters to count 0-99.

Table 1 Cyclic Code Encoded Decade Count

Decade Count	Cyclic Code	
0	0000	0x0
1	0001	0x1
2	0101	0x5
3	0111	0x7
4	0110	0x6
5	1110	0xE
6	1100	0xC
7	1101	0xD
8	1001	0x9
9	1000	0x8

Procedure:

The assignment required the use of basic logic gates and flip-flops, and hence the first step was to obtain the state table based on the flip-flop implementation. T-flip-flop and D-flip-flop were chosen and two different designs were built. Then k-maps were drawn using the state table to obtain logic equations. Based on the logic equations, basic logic gates and were used to implement the schematic. The schematic was then compiled and simulated to obtain waveforms that verified the functionality of the decade counter.

Specifications:

- ❖ To design a sequential logic that implements a state machine (to take care of mapping of decade count onto the cyclic code).
- ❖ The sequential logic circuit contains storage elements – flip-flops, and combinational logic that together implements an event based state-machine.
- ❖ The event in the case of the counter is the raising edge of the clock. The next state of the storage element depends on its current state.
- ❖ The design can be implemented using any kind of storage element, and the decision to use any particular implementation rests on simplicity of design, chip area of implementation etc. Two different designs were implemented to verify the operation of the counter. The first design used T-Flip-Flop (TFF) while the second design used D-Flip-Flop (DFF). Both TFF and DFF were chosen for their simplicity in design. The operation truth table of the TFF and DFF are given in table 2 and 3 respectively. The output of the TFF toggles if its T-input is asserted, and remains unaltered if the T-input is low. The input of the DFF transfers to its output during the rising edge of the clock.

Table 2 Truth Table of a T-Flip-Flop

T (Input)	Q	
	(Cur. State)	(Next State)
0	0	0
0	1	1
1	0	1
1	1	0

Table 3 Truth Table of a D-Flip-flop

Clock	D (Input)	Q (Output)
0	0	Q
Rising	1	1
0	0	Q
Rising	0	0

Design:

The state table based on the flip-flop implementation is the first step in the design of the counter. Please see table 4 and 5 for TFF and DFF implementation.

Table 4 State table TFF implementation

Q ₄ Q ₃ Q ₂ Q ₁ (Output)		T ₄ T ₃ T ₂ T ₁ (Input)
(Cur. State) dcba	(Next State)	To trigger state chg
0000	0001	0001
0001	0101	0100
0010	XXXX	XXXX
0011	XXXX	XXXX
0100	XXXX	XXXX
0101	0111	0010
0110	1110	1000
0111	0110	0001
1000	0000	1000
1001	1000	0001
1010	XXXX	XXXX
1011	XXXX	XXXX
1100	1101	0001
1101	1001	0100
1110	1100	0010
1111	XXXX	XXXX

Table 5 State table DFF implementation

(Current State) Q ₃ Q ₂ Q ₁ Q ₀	(Next State) D ₃ D ₂ D ₁ D ₀
0000	0001
0001	0101
0101	0111
0111	0110
0110	1110
1110	1100
1100	1101
1101	1001
1001	1000
1000	0000

The crucial part of the design problem is determining a suitable combinational logic to generate an input that would effect the next-state transition on the raising edge of the clock.

K-Map for Q (1-4) also named 'dcba' current state to T-input(1-4) is given in table 5.

K-Map for D (0-4) for Q input (0-4) is given in table 6.

Table 5. K-map for T-Flip-Flop implementation.

T4		00	01	11	10
	00	0	0	X	X
	01	X	0	0	1
	11	0	0	X	0
	10	1	0	X	X
T3		00	01	11	10
	00	0	1	X	X
	01	X	0	0	0
	11	0	1	X	0
	10	0	0	X	X
T2		00	01	11	10
	00	0	0	X	X
	01	X	1	0	0
	11	0	0	X	1
	10	0	0	X	X
T1		00	01	11	10
	00	1	0	X	X
	01	X	0	1	0
	11	1	0	X	0
	10	0	1	X	X

From the adjacent elements of the 'ON' T-input bit we get the following logic equations for the T(1-4):

LOGIC EQUATIONS FOR T-INPUTS OF TFF

$$T4 = a'd'c + a'dc' = a'(d'c + dc') = a'(d \text{ xor } c)$$

$$T3 = ad'c' + adc = a(d'c' + dc) = a(D \text{ xnor } A)$$

$$T2 = d'cb' + dba'$$

$$T1 = cb'a' + cba + c'd'a' + c'da = c(b'a' + ba) + c'(d'a' + da) = c(b \text{ xnor } a) + c'(d \text{ xnor } a)$$

Table 6. K-map for D-Flip-Flop implementation.

D0		00	01	11	10
	00	1	1	X	X
	01	X	1	0	0
	11	1	1	X	0
	10	0	0	X	X
D1		00	01	11	10
	00	0	0	X	X
	01	X	1	1	1
	11	0	0	X	0
	10	0	0	X	X
D2		00	01	11	10
	00	0	1	X	X
	01	0	1	1	1
	11	1	0	X	1
	10	0	0	X	X
D3		00	01	11	10
	00	0	0	X	X
	01	X	0	0	1
	11	1	1	X	1
	10	0	1	X	X

LOGIC EQUATIONS FOR D-flip-flop

$$D0 = Q3'Q2'Q1' + Q3'Q1'Q0 + Q3Q2Q1'$$

$$D1 = Q3'Q2Q0 + Q3'Q2Q1Q0'$$

$$D2 = Q3'Q1'Q0 + Q3'Q2Q1 + Q3Q2Q0'$$

$$D3 = Q3Q2Q1' + Q3Q1'Q0 + Q2Q1Q0'$$

Schematics:

Figure 1: 4-Bit Decade Counter TFF implementation

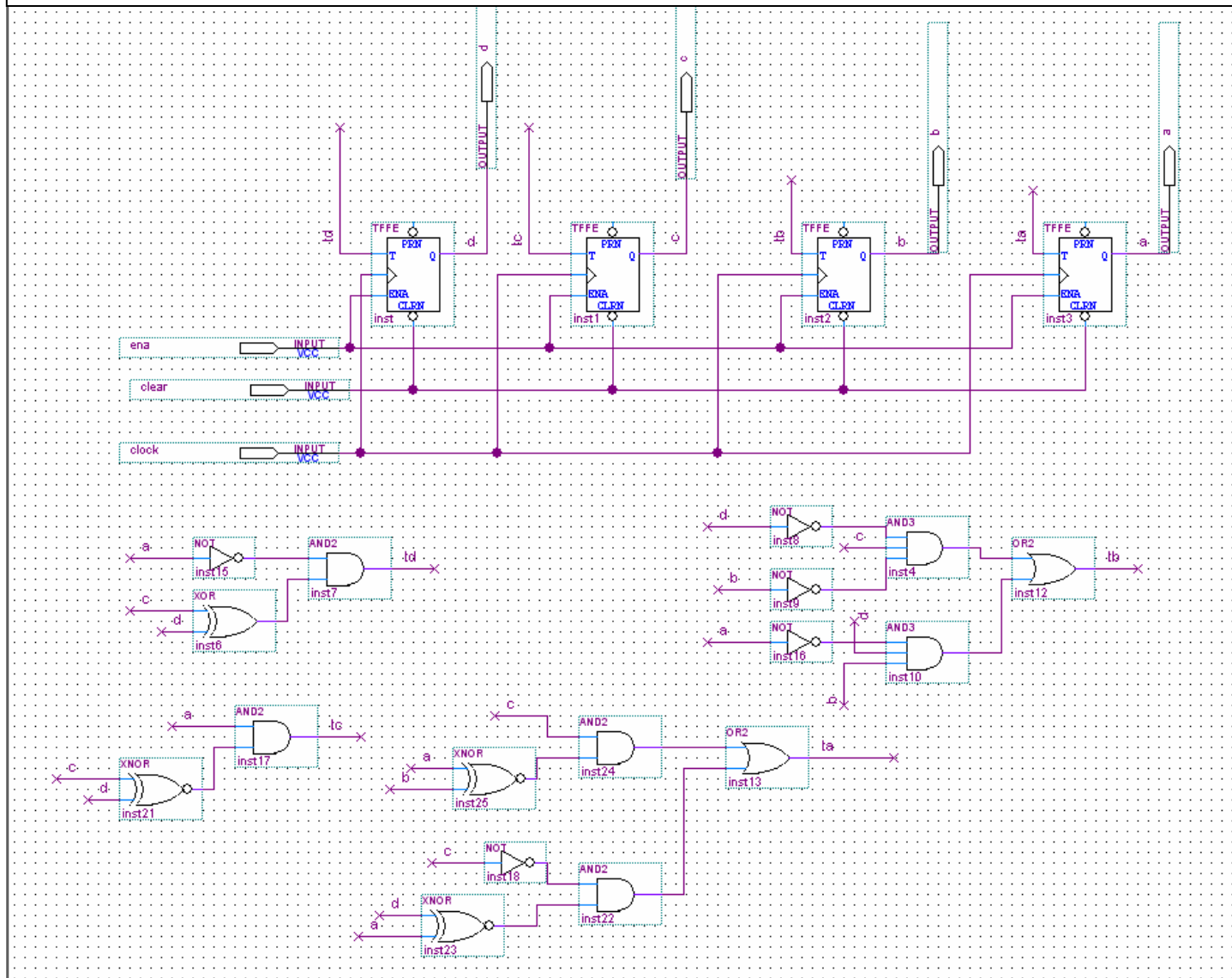
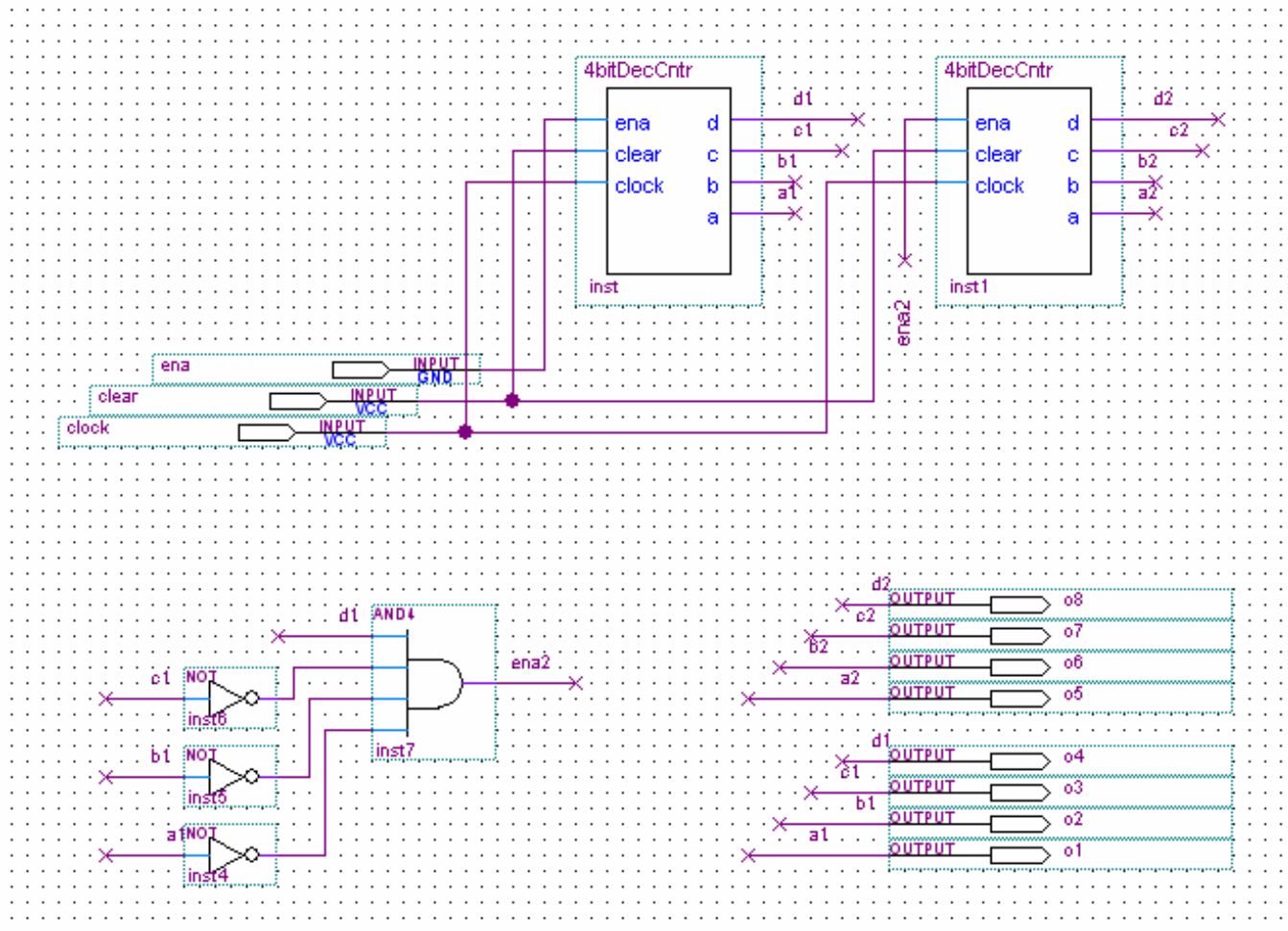


Figure 2: 4-Bit Decade Counter TFF implementation



Every nine cycles the cascaded 4-bit counters enable is asserted by the logic equation $ena = d1c1'b1'a1'$, thus the cascaded counter counts once every nine counts of first counter, implementing a 0-99 count.

Figure 3. 4-Bit Decade Counter DFF implementation

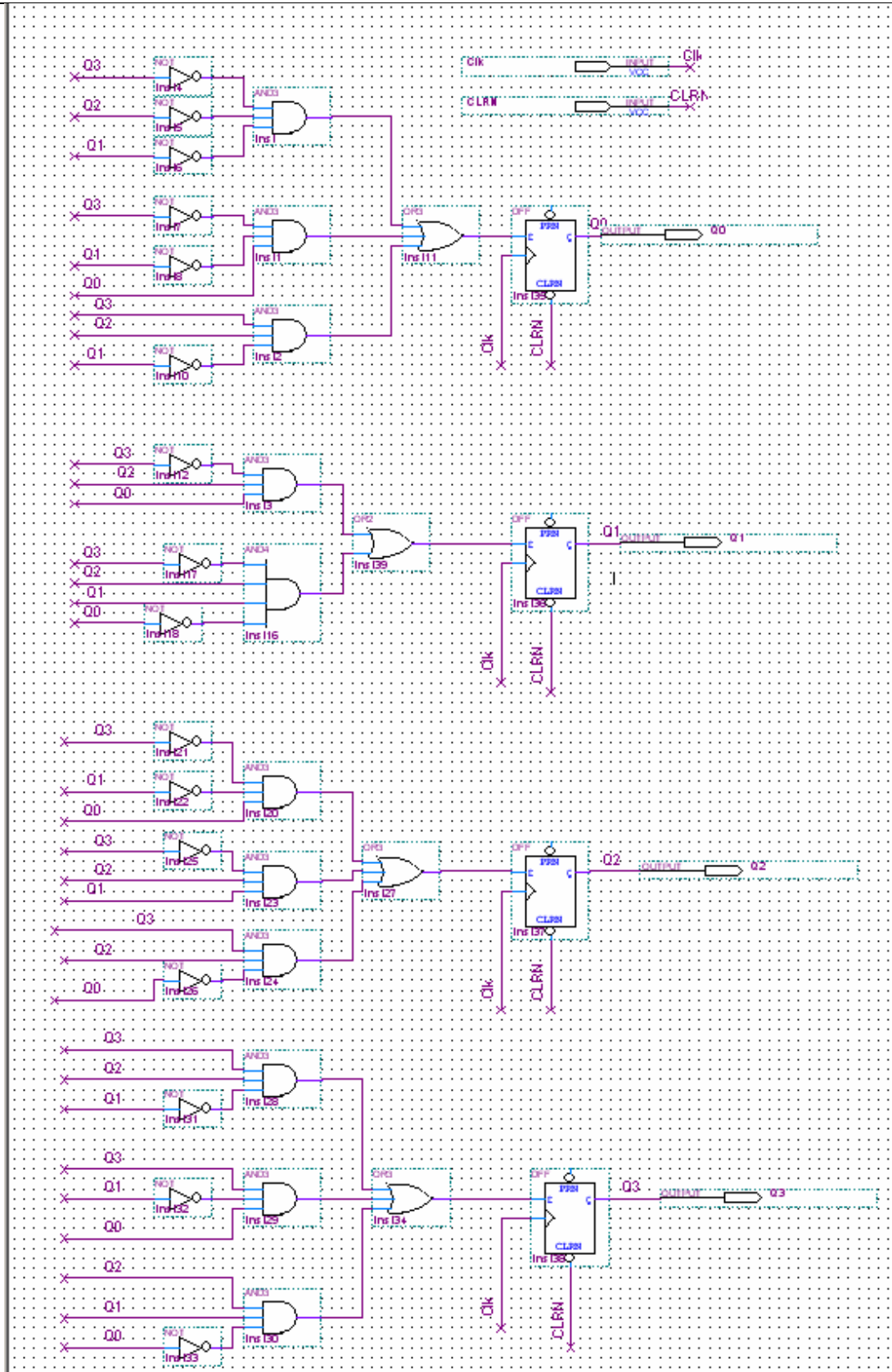
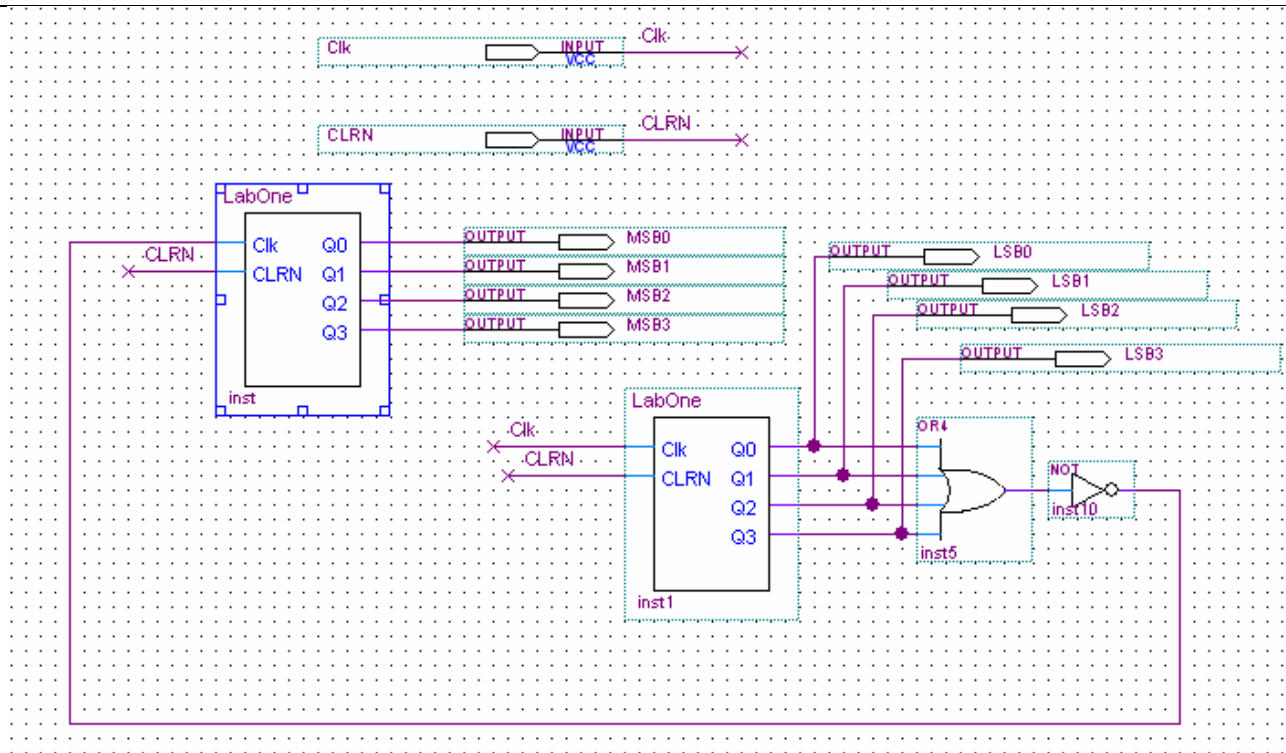


Figure 4. 4-Bit Decade Counter DFF implementation



The clock of the cascaded counter is derived from the output of the first counter, thus it counts once for every 9 cycle count of the first counter, implementing a 0-99 count.

Results

The output waveform of the 0-99 counter shows the implementation of the cyclic code (see the hex equivalent), the least significant nibble (lsn) implements a 4-bit decade counter. And as per our design the most significant nibble (msn) – the cascaded counter counts every 9 counts of the lsn, and it also implements the cyclic code. See figure 5 and 6 for waveforms from TFF and DFF implementation.

Figure 5. Waveform results for TFF implementation.

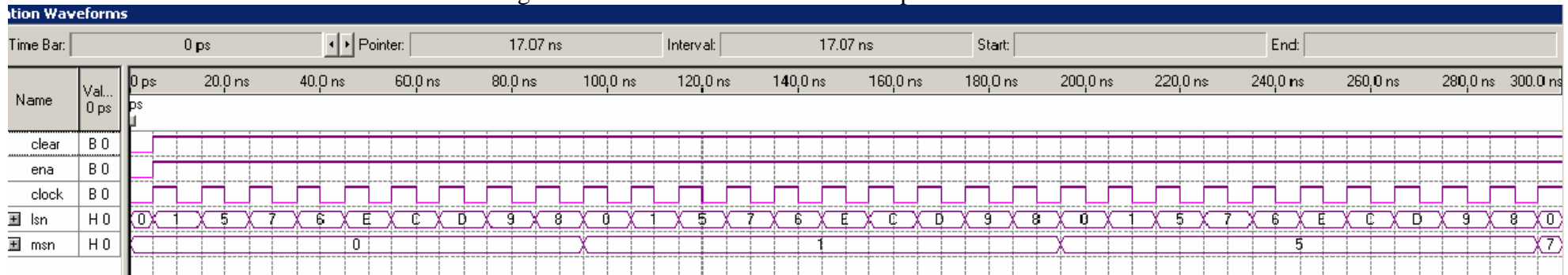
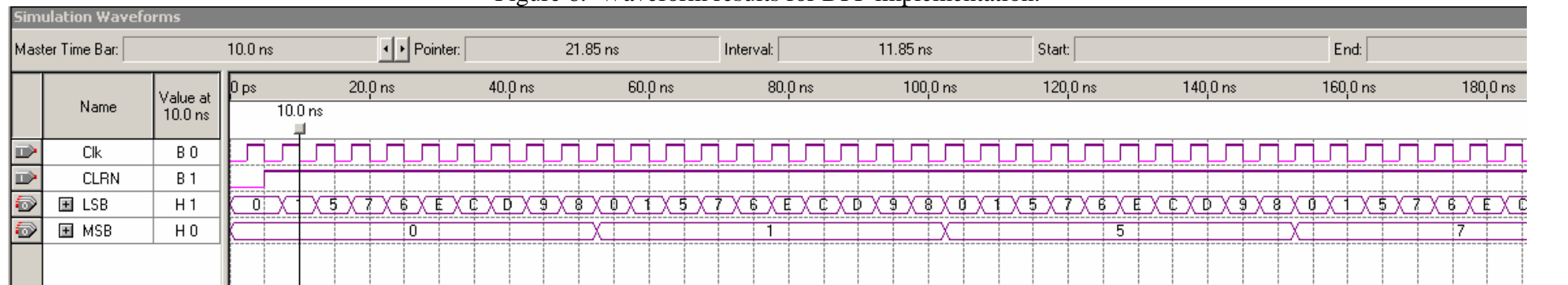


Figure 6. Waveform results for DFF implementation.



Conclusion

We verified the functionality of a cascaded decade counter that counted values from 0 through 99. We went a step further and implemented the counter using two different designs based on a TFF and a DFF. The results of both designs were the same. This assignment was very useful in teaching us how to design, compile and simulate on Quartus II software.