Practice Assignment 3 – EMB 2

Software State Machines

Assignment 3

Submission deadline: **21 March 2025**

Make a software implementation of the state-machine shown in figure 7-43 on page 551 in "Digital Design – Wakerly (4th edition)". Two programs are expected:

- program 1, based on transition equations (parts A, B)
- program 2, based on the state diagram (part C)

- Use VSCode and the Arduino nano microcontroller as target.
- Use Button 3 and Button 4 for the X and Y inputs
- Use the serial monitor to show the current state, and I/O pins for X, Y, Z1 and Z2.
- The clock signal will be a function that will make the state transition when called, as well as shortly turn on PB5 for 100 ms.
- Make a loop where you call the transition function periodically (each 1 second for example).
- Document your C-codes and the software tests.

Hand in **a single pdf file*** with

- your code,
- explanations for the functionality of the program and
- test cases.

Circuit Diagram

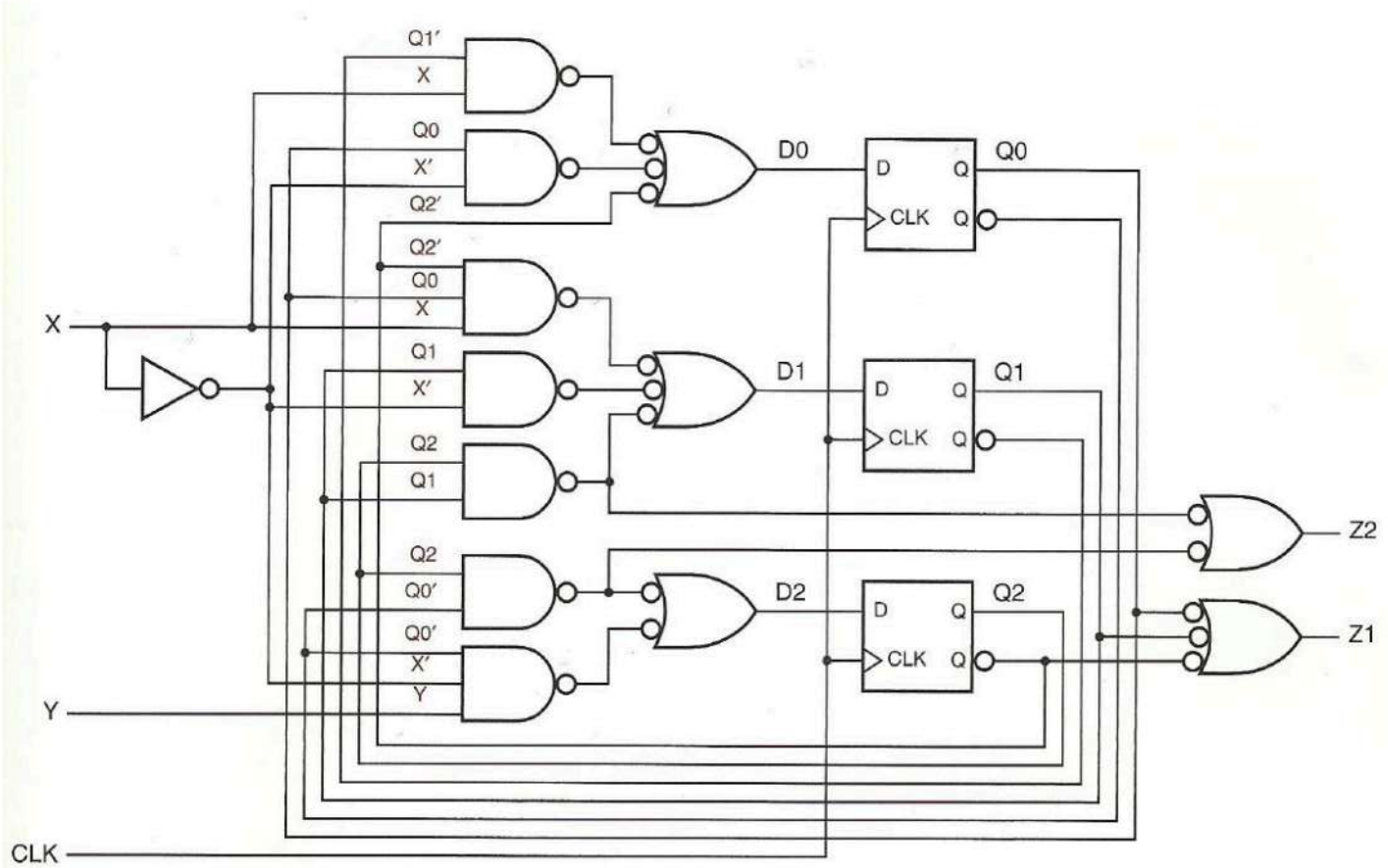


Figure 7-43 A clocked synchronous state machine with three flip-flops and eight states.

Transition/Output table

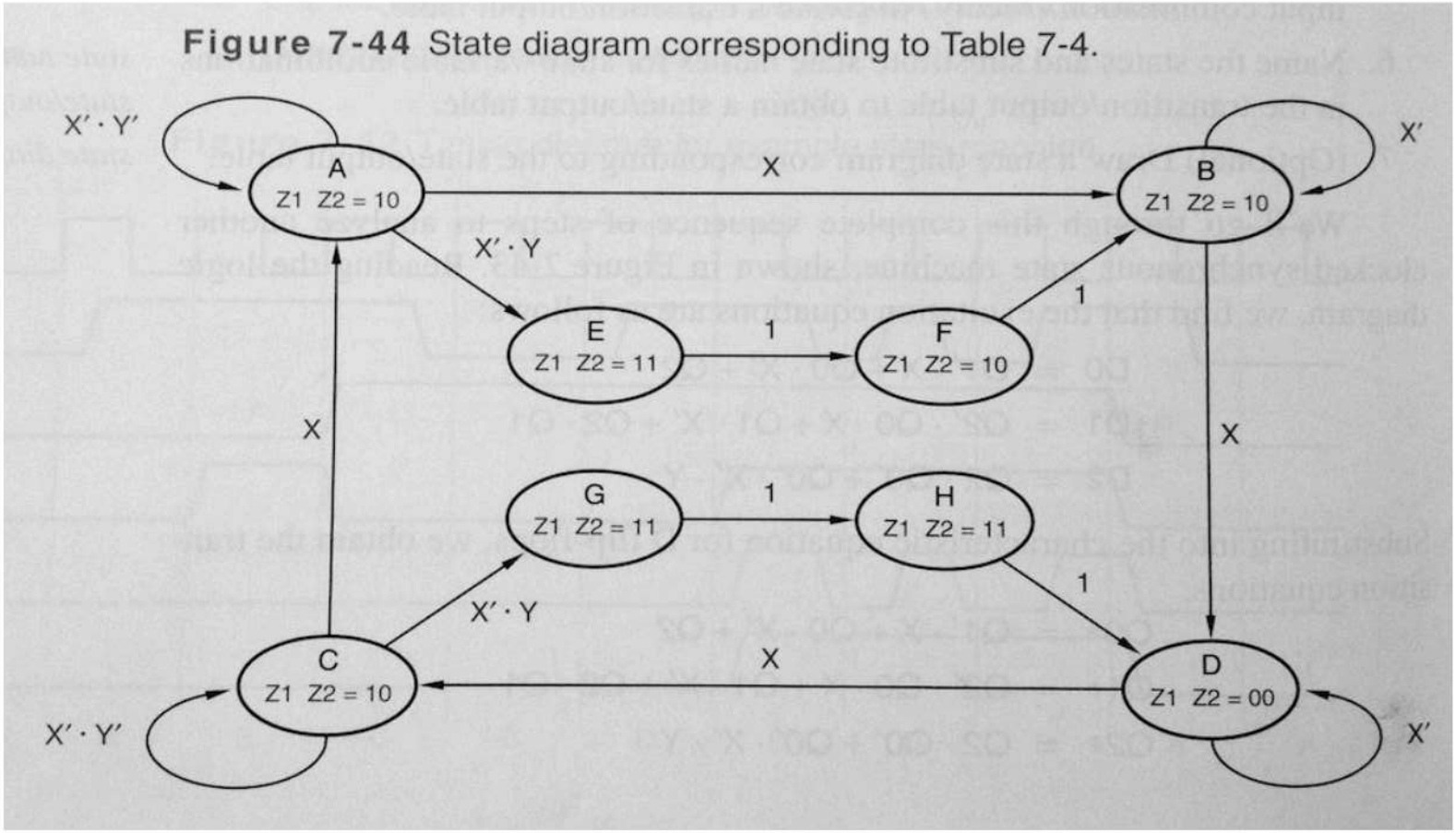
(a)

	XY				
Q2 Q1 Q0	00	01	10	11	Z1 Z2
000	000	100	001	001	10
001	001	001	011	011	10
010	010	110	000	000	10
011	011	011	010	010	00
100	101	101	101	101	11
101	001	001	001	001	10
110	111	111	111	111	11
111	011	011	011	011	11
	Q2* Q1* Q0*				

(b)

	XY				
S	00	01	10	11	Z1 Z2
A	A	E	B	B	10
B	B	B	D	D	10
C	C	G	A	A	10
D	D	D	C	C	00
E	F	F	F	F	11
F	B	B	B	B	10
G	H	H	H	H	11
H	D	D	D	D	11
	S*				

State Diagram



Part A (Lecture)

- Write the excitation + transition equations for the given circuit (on a piece of paper)

- Use the skeleton provided below to create a program in VS Code (start from the Hello World template, EMB1)
- Create all needed functions (empty for now)
- Implement the `read_xy()` function that monitors the state of the buttons 3 and 4. The function should update the global variables **X** and **Y** to 1/0 according to if the buttons are pressed/not pressed.
- Implement the *`show_output()`* function that will display variables **X** and **Y** for now.
- Implement the *`state_transition()`* function that will turn on for now PB5 for 100ms, then turn it OFF

Skeleton of the program (parts A and B)

```
#include <stdio.h> // + delay, + usart, + io



unsigned char x, y, z1, z2; // inputs and outputs
unsigned char q0, q1, q2, q0_next, q1_next, q2_next;

void read_xy_values(void); //checking which button is pressed
void show_output(void); //showing current state + state variables on the screen
void state_transition(void ); //advancing to the new state by implementing transition equations

main()
{
      q0 = 0;
      q1 = 0;
      q2 = 0;
      .
      .

      while(1)
      {
            read_xy_values();
            state_transition();
            show_output();
            _delay_ms(1000);
      }
}
// … function definitions…
```

Part B (Lab)

- You must implement the transition equations in code
  - the tuple q2, q1, q0 gives the current state
  - the tuple q2_next, q1_next, q0_next gives the next state
- How to optimally implement the transition equations? Which arithmetic/logic operators to use?
- We improve the button reading: How to check if a button is pressed ? With or without other buttons pressed at the same time? hint: bitwise operators
- Print all variables in the `show_output()` function
- How can we display the output on the screen as a state name? (consider using ASCII characters) and calculate the current state as 0, 1, 2, 3 … then represent it as A, B, C, D… hint: convert tuple q2, q1, q0 into a decimal number

Part C (Homework)

- Use enumerated types to represent the states and implement directly the state transition table
- Use the skeleton on next page
- Use the following state machine with 4 states example as inspiration

Skeleton of the program part C:

```
#include <stdio.h> // + delay, + usart, + io

// another way: use state names
typedef enum {
            A_state,
            B_state,
            C_state,
            D_state,
            E_state,
            F_state,
            G_state,
            H_state
}state;

main()
{
      state current_state, next_state;

      current_state = A_state;
      .
      .

      while(1)
      {
            read_xy_values();
            next_state = state_transition(current_state, x, y);
            current_state = next_state;
            show_output(current_state);
            _delay_ms(1000);
      }
}
//…
```

Example with enumerated types

Enumerated type: a data type whose list of values is specified by the programmer in the type declaration. Example:

```
typedef enum {
          A_STATE,
          B_STATE,
          C_STATE,
          D_STATE
          }state;
```

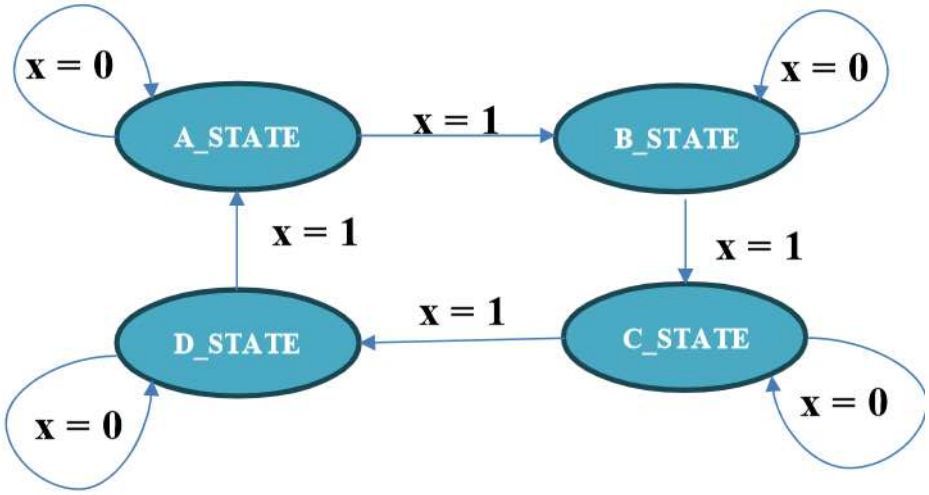
Defining type ***state*** as shown causes the enumeration constant **A_STATE** to be represented as the integer **0**, constant **B_STATE** to be represented as integer **1**, and so on.

Below, variable *current_state* can be manipulated just as one would handle any other integers.

Example:

```
state current_state, next_state;
current_state = A_STATE;
next_state = B_STATE;
current_state = next_state;
```

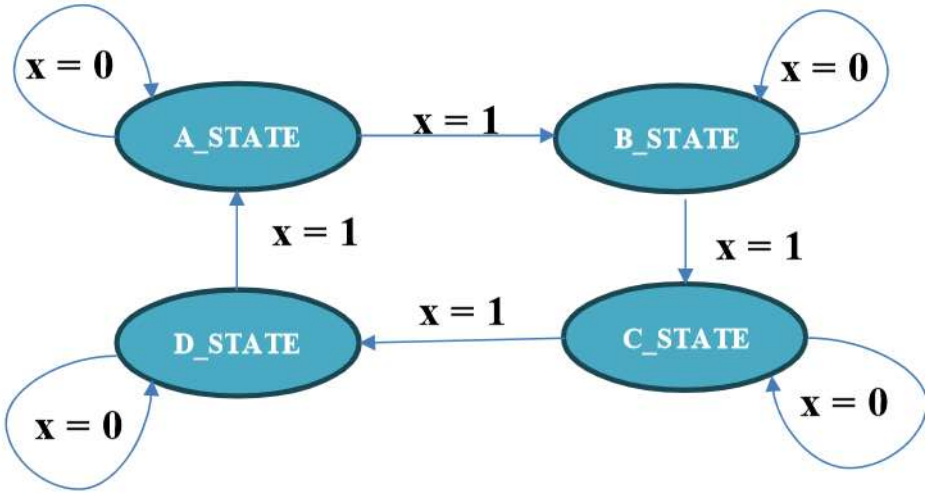
Here is how this state diagram might be implemented:




```
#include <stdio.h> // + delay, + usart, + io
typedef enum {
            A_STATE,
            B_STATE,
            C_STATE,
            D_STATE
}state;
// + function prototypes
main()
{
      state current_state, next_state;
      current_state = A_STATE; // initial state
      int x = 0; // initial input
      // initialize DDRC, DDRD, PORTC, PORTD ...
      while(1)
      {
            x = read_x(); // will be 0 or 1, according to a button pressed or not
            next_state = state_transition(current_state, x);
            current_state = next_state ;
            print_state(current_state);
            _delay_ms(1000);
      }
}

void print_state (state state_to_print){
            switch(state_to_print){
                        case A_STATE: printf("A_STATE \n"); break;
                        case B_STATE: printf("B_STATE \n"); break;
                        case C_STATE: printf("C_STATE \n"); break;
                        case D_STATE: printf("D_STATE \n"); break;
            }
}
```

Here is how this state diagram might be implemented:




```
// ... continues from previous slide

state state_transition(state current_state, int input){
            if (input){ // if x = 1, then change states
               switch (current_state){
                        case (A_STATE):
                                    return B_STATE;
                        case (B_STATE):
                                    return C_STATE;
                        case (C_STATE):
                                    return D_STATE;
                        case (D_STATE):
                                    return A_STATE;
                }
            }
            return current_state; // if the program reaches this point, then x = 0, so keep current state
}

int read_x(void){
// if button pressed return 1 else return 0

}
```