**Table 1. Truth table for 7-segment display decoder**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Hexadecimal Digit** | **Inputs** | | | | **Outputs** | | | | | | | | |
|  |  |  |  | **PORTD** | | | | **PORTB** | | | | |
| **PB3** | **PB2** | **PB1** | **PB0** | **Sg** | **Sf** | **Se** | **(in hex)** | **Sd** | **Sc** | **Sb** | **Sa** | **(in hex)** |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 06 | 1 | 1 | 1 | 1 | 0F |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 00 | 0 | 1 | 1 | 0 | 06 |
| 2 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0A | 1 | 0 | 1 | 1 | 0B |
| 3 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 08 | 1 | 1 | 1 | 1 | 0F |
| 4 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0C | 0 | 1 | 1 | 0 | 06 |
| 5 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0C | 1 | 1 | 0 | 1 | 0D |
| 6 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | OE | 1 | 1 | 0 | 1 | 0D |
| 7 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 00 | 0 | 1 | 1 | 1 | 07 |
| 8 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0E | 1 | 1 | 1 | 1 | 0F |
| 9 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0C | 1 | 1 | 1 | 1 | 0F |
| A | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0E | 0 | 1 | 1 | 1 | 07 |
| B | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0E | 1 | 1 | 0 | 0 | 0C |
| C | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 06 | 1 | 0 | 0 | 1 | 09 |
| D | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0A | 1 | 1 | 1 | 0 | 0E |
| E | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0E | 1 | 0 | 0 | 1 | 09 |
| F | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0E | 0 | 0 | 0 | 1 | 01 |

**Code**

#include <avr/io.h>

#include <util/delay.h>

#define *F\_CPU* 16000000UL

void BinaryCounter\_SSD() {

while (1) {

switch (PINC) {

case 0x00:

PORTD = 0x06;

PORTB = 0x0F;

break;

case 0x01:

PORTD = 0x00;

PORTB = 0x06;

break;

case 0x02:

PORTD = 0x0A;

PORTB = 0x0B;

break;

case 0x03:

PORTD = 0x08;

PORTB = 0x0F;

break;

case 0x04:

PORTD = 0x0C;

PORTB = 0x06;

break;

case 0x05:

PORTD = 0x0C;

PORTB = 0x0D;

break;

case 0x06:

PORTD = 0x0E;

PORTB = 0x0D;

break;

case 0x07:

PORTD = 0x00;

PORTB = 0x07;

break;

case 0x08:

PORTD = 0x0E;

PORTB = 0x0F;

break;

case 0x09:

PORTD = 0x0C;

PORTB = 0x0F;

break;

case 0x0A:

PORTD = 0x0E;

PORTB = 0x07;

break;

case 0x0B:

PORTD = 0x0E;

PORTB = 0x0C;

break;

case 0x0C:

PORTD = 0x06;

PORTB = 0x09;

break;

case 0x0D:

PORTD = 0x0A;

PORTB = 0x0E;

break;

case 0x0E:

PORTD = 0x0E;

PORTB = 0x09;

break;

case 0x0F:

PORTD = 0x0E;

PORTB = 0x01;

break;

default:

break;

}

}

}

int main() {

DDRB = 0xFF;

DDRC = 0x0F;

DDRD = 0xFF;

PORTB = 0x0F;

PORTC = 0x0F;

PORTD = 0x0E;

BinaryCounter\_SSD();

return 0;

}

//Explanation sa code

The provided code is designed to interface with and control a common anode 7-segment display using an AVR microcontroller programmed in the C language. The primary objective is to display a binary counter ranging from 0 to 15 (0x0 to 0xF) continuously on the 7-segment display

#include <avr/io.h>

#include <util/delay.h>

#define *F\_CPU* 16000000UL

Header Files and Clock Frequency: The code starts by including necessary header files for AVR programming and delay functions. It defines the clock frequency F\_CPU, which must match the actual clock frequency of the microcontroller.

BinaryCounter\_SSD Function

void BinaryCounter\_SSD() {

while (1) {

switch (PINC) {

case 0x00:

PORTD = 0x06;

PORTB = 0x0F;

break;

case 0x01:

PORTD = 0x00;

PORTB = 0x06;

break;

case 0x02:

PORTD = 0x0A;

PORTB = 0x0B;

break;

case 0x03:

PORTD = 0x08;

PORTB = 0x0F;

break;

case 0x04:

PORTD = 0x0c;

PORTB = 0x06;

break;

case 0x05:

PORTD = 0x0C;

PORTB = 0x0D;

break;

case 0x06:

PORTD = 0x0E;

PORTB = 0x0D;

break;

case 0x07:

PORTD = 0x00;

PORTB = 0x07;

break;

case 0x08:

PORTD = 0x0E;

PORTB = 0x0F;

break;

case 0x09:

PORTD = 0x0C;

PORTB = 0x0F;

break;

case 0x0A:

PORTD = 0x0E;

PORTB = 0x07;

break;

case 0x0B:

PORTD = 0x0E;

PORTB = 0x0C;

break;

case 0x0C:

PORTD = 0x06;

PORTB = 0x09;

break;

case 0x0D:

PORTD = 0x0A;

PORTB = 0x0E;

break;

case 0x0E:

PORTD = 0x0E;

PORTB = 0x09;

break;

case 0x0F:

PORTD = 0x0E;

PORTB = 0x01;

break;

default:

break;

}

}

}

BinaryCounter\_SSD Function is the core of the design. It uses a switch statement to map the input from Port C (PINC) to specific binary values corresponding to decimal numbers 0 through 15.

Main Function

int main() {

DDRB = 0xFF;

DDRC = 0x0F;

DDRD = 0xFF;

PORTB = 0x0F;

PORTC = 0x0F;

PORTD = 0x0E;

BinaryCounter\_SSD();

return 0;

}

The main function initializes the data direction registers (DDRB, DDRC, DDRD) to define the I/O direction of the ports:

DDRB and DDRD are set to output mode to control the segments and common anode pins. DDRC is set to input mode, indicating that Port C will be used for reading input values (digit selection).Initial values are set for PORTB, PORTC, and PORTD to configure the common anode pin,.The program is designed to run continuously. It enters an infinite loop (while(1)) within the BinaryCounter\_SSD function. This ensures that the binary counting and display process continuously repeats, displaying numbers 0 through 15 in a loop on the 7-segment display.

//Explanation design and testing

Designing a seven-segment display using an Arduino Uno microcontroller involves both hardware and software considerations. The hardware aspect encompasses the physical connections between the Arduino and the components, which, in this case, include push buttons and the seven-segment display. Specifically, we connect four push buttons to analog pins A0 to A3, with each button representing a binary input state (pressed or not pressed). Simultaneously, the seven-segment display is interfaced with the Arduino via digital pins on PORTD to control the individual segments and another digital pin on PORTB to manage the common cathode or anode connection.

In parallel with the hardware setup, we need a truth table to define how the seven-segment display should behave based on the states of the push buttons. The truth table serves as a critical reference for the software logic. Each row in the table represents a unique combination of button states, and the corresponding output specifies which segments of the seven-segment display should be turned on or off to display a particular digit. When all buttons are not pressed (0, 0, 0, 0), the seven-segment display should exhibit the digit 0, and the binary representation 0b00111111 tells us which segments should be illuminated to achieve this. When Button 1 alone is pressed (0, 0, 0, 1), the display should show 1 (0b00000110). This process continues for all possible button combinations.

\The testing phase is where we ensure that our hardware and software work together correctly. Here's how we do it:

We start by putting the code we've written onto the Arduino Uno using the Microchip Studio.We double-check that all our wires and components are properly connected and that the power supply is set up correctly. We then follow a systematic plan. We press each button one by one and look at the display. We need to make sure that what's shown matches what we expect based on our plan. We do this for all possible button combinations to make sure everything works as it should.Improvement and Documentation: After everything works well we write down all the important information about our project, like how it's wired, the code we used, and how to use it. This way, we have a record and can share it with others if needed.