Digital Design Through ESP32-Devikit-v1

G. V. V. Sharma

About this book

This book provides a simple introduction to digital design using the Arduino framework, assembly and embedded C. It is suitable for students ranging from primary school to college. The content is sufficient for industry jobs. There is no copyright, so readers are free to print and share.

November 15, 2024

First manual appeared in 2015

Contents

1 Installation

1.1 *Termux*

1. On your android device, follow the instructions in

   ```
   https://github.com/gadepall/fwc−1
   ```

   to setup and install Debian on Termux.

1.2 *Platformio*

1. Install Packages

   ```
   apt install avra avrdude gcc−avr avr−libc
   ```

2. Follow the instructions in

   ```
   https://docs.platformio.org/en/latest/core/installation/methods/installer−script.html #
         super−quick−macos−linux
   ```

   to install platformio.
3. Execute the following on debian

   ```
   cd installation/codes
   pio run
   ```

4. Connect your ESP32-devkit-v1 to the laptop/rpi and type

   ```
   pio run −t nobuild −t upload
   ```

5. The LED beside pin 13 will start blinking

1.3 *Arduino Droid*

1. Install ArduinoDroid from apkpure
2. Open ArduinoDroid and grant all permissions
3. Connect the ESP32-devkit-v1-devkit-v1 to your phone via USB-OTG
4. For flashing the bin files, in ArduinoDroid,

   ```
   Actions−>Upload−>Upload Precompiled
   ```

   then go to your working directory and select

   ```
   .pio/build/esp32doit−devkit−v1/firmware.bin
   ```

   for uploading bin file to the ESP32-Devkit-v1.
5. The LED beside pin RX2 will start blinking

2 Seven Segment Display

We show how to control a seven segment display.

2.1 Components

Component	Value	Quantity
Resistor	220 Ohm	1
ESP32-devkit-v1		1
Seven Segment Display		1
Decoder	7447	1
Flip Flop	7474	2
Jumper Wires		20

TABLE 2.1: Components

1. Breadboard:

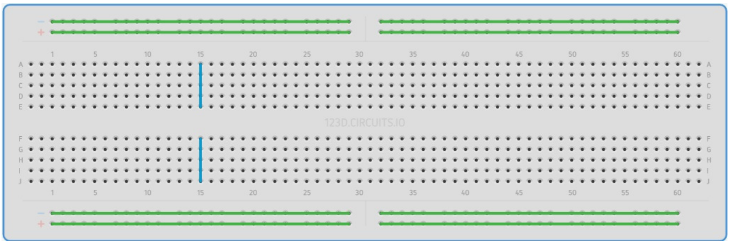

Fig. 2.1: Bread board connnections

   The breadboard can be divided into 5 segments. In each of the green segements, the pins are internally connected so as to have the same voltage. Similarly, in the central segments, the pins in each column are internally connected in the same fashion as the blue columns.
2. Seven Segment Display: The seven segment display in Fig. 2.2 has eight pins, a, b, c, d, e, f, g and *dot* that take an active LOW input, i.e. the LED will glow only if the input is connected to ground. Each of these pins is connected to an LED segment. The *dot* pin is reserved for the $\cdot$ LED.

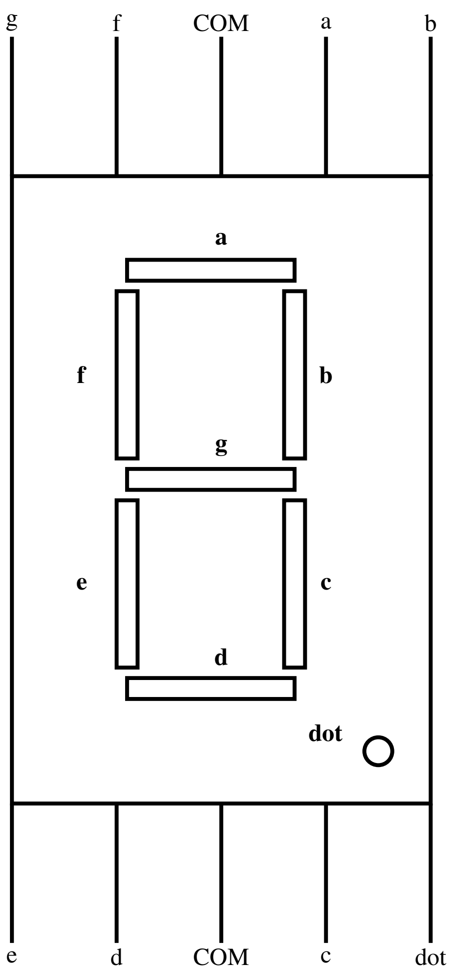

Fig. 2.2: Seven Segment pins

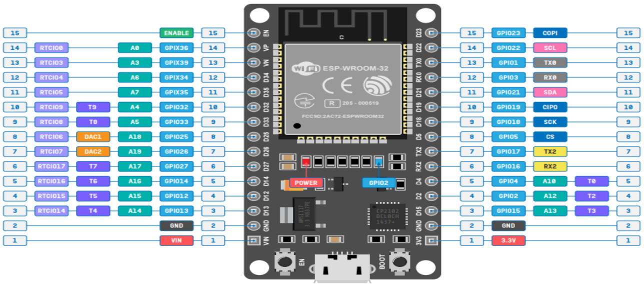

Fig. 2.3: ESP32-Devkit-v1

3. ESP32-devkit-v1: The ESP32-devkit-v1 in Fig 2.3 has some ground pins, ADC(*AnalogtoDigitalConverter*) input pins D2, D4, D12-D15, D25-D27, D32 and D33 that can be used for both input as well as output. It also has two power pin that can generate 3.3V. In the following exercises, only the GND, 3.3V and digital pins will be used.

2.2 *Display Control through Hardware*

2.2.1 *Powering the Display:*

1. Plug the display to the breadboard in Fig. 2.1 and make the connections in Table 2.2. Henceforth, all 3.3V and GND connections will be made from the breadboard.

ESP32-devkit-v1	**Breadboard**
3.3V	Top Green
GND	Bottom Green

TABLE 2.2: Supply for Bread board

2. Make the connections in Table 2.3.

Breadboard		**Display**
3.3V	Resistor	COM
GND		DOT

TABLE 2.3: Connecting Seven segment display on Bread board

3. Connect the ESP32-devkit-v1 to the computer. The DOT led should glow.

2.2.2 Controlling the Display: Fig. 2.4 explains how to get decimal digits using the seven segment display. GND=0.

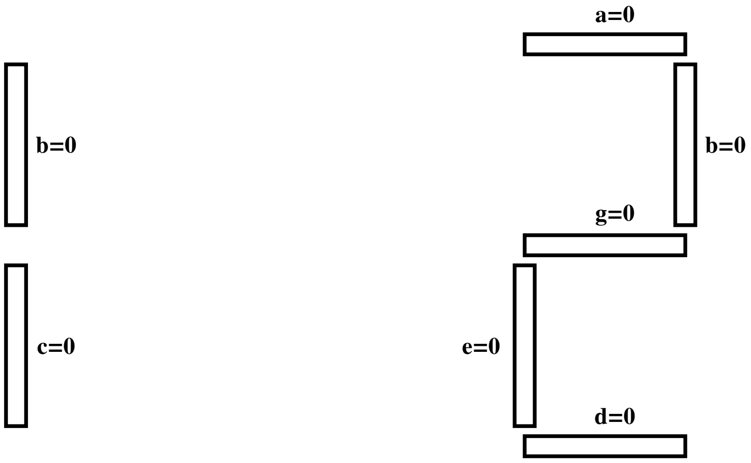

Fig. 2.4: Seven Segment connections

ESP32	13	12	14	27	26	25	33
Display	a	b	c	d	e	f	g

TABLE 2.5

1. Generate the number 1 on the display by connecting only the pins *b* and *c* to GND (=0). This corresponds to the first row of 2.4. 1 means not connecting to GND.
2. Repeat the above exercise to generate the number 2 on the display.
3. Draw the numbers 0-9 as in Fig. 2.4 and complete Table 2.4

a	**b**	**c**	**d**	**e**	**f**	**g**	**decimal**
0	0	0	0	0	0	1	0

TABLE 2.4

2.3 Display Control through Software

1. Make connections according to Table 2.5
2. Download the following code using the arduinodroid and execute

```
cd ide/sevenseg/codes/sevenseg/sevenseg.cpp
pio run
```

3. Now generate the numbers 0-9 by modifying the above program.

3 7447

Here we show how to use the 7447 BCD-Seven Segment Display decoder to learn Boolean logic.

3.1 *Hardware*

1. Make connections between the seven segment display in Fig. 2.2 and the 7447 IC in Fig. 3.1 as shown in Table 3.1

7447	$\bar{a}$	$\bar{b}$	$\bar{c}$	$\bar{d}$	$\bar{e}$	$\bar{f}$	$\bar{g}$
Display	a	b	c	d	e	f	g

TABLE 3.1

2. Make connections to the lower pins of the 7447 according to Table 3.2 and connect VIN = 3.3V. You should see the number 0 displayed for 0000 and 1 for 0001.

D	**C**	**B**	**A**	**Decimal**
0	0	0	0	0
0	0	0	1	1

TABLE 3.2

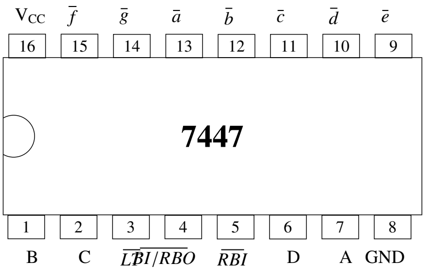

Fig. 3.1: 7447 IC

3. Complete Table 3.2 by generating all numbers between 0-9.

3.2 *Software*

1. Now make the connections as per Table 3.3 and execute the following program

```
ide/7447/codes/gvv_ard_7447/gvv_ard_7447.cpp
```

7447	D	C	B	A
ESP32-Devkit-v1	27	14	12	13

TABLE 3.3

Z	Y	X	W	**D**	**C**	**B**	**A**
0	0	0	0	**0**	**0**	**0**	**1**
0	0	0	1	**0**	**0**	**1**	**0**
0	0	1	0	**0**	**0**	**1**	**1**
0	0	1	1	**0**	**1**	**0**	**0**
0	1	0	0	**0**	**1**	**0**	**1**
0	1	0	1	**0**	**1**	**1**	**0**
0	1	1	0	**0**	**1**	**1**	**1**
0	1	1	1	**1**	**0**	**0**	**0**
1	0	0	0	**1**	**0**	**0**	**1**
1	0	0	1	**0**	**0**	**0**	**0**

TABLE 3.4: Truth table for incrementing Decoder.

In the truth table in Table 3.4, W, X, Y, Z are the inputs and A, B, C, D are the outputs. This table represents the system that increments the numbers 0-8 by 1 and resets the number 9 to 0 Note that $D = 1$ for the inputs 0111 and 1000. Using *boolean* logic,

$$D = WXYZ' + W'X'Y'Z \tag{3.1}$$

Note that 0111 results in the expression $WXYZ'$ and 1000 yields $W'X'Y'Z$.

2. The code below realizes the Boolean logic for B, C and D in Table 3.4. Write the logic for A and verify.

```
ide/7447/codes/inc_dec/inc_dec.ino
```

3. Now make additional connections as shown in Table 3.5 and execute the following code. Comment.

```
ide/7447/codes/ip_inc_dec/ip_inc_dec.cpp
```

Solution: In this exercise, we are taking the number 5 as input to the ESP32-devkit-v1 and displaying it on the seven segment display using the 7447 IC.

	Z	Y	X	W
Input	0	1	0	1
ESP32	32	33	25	26

TABLE 3.5

4. Verify the above code for all inputs from 0-9.
5. Now write a program where
   a) the binary inputs are given by connecting to 0 and 1 on the breadboard
   b) incremented by 1 using Table 3.4 and
   c) the incremented value is displayed on the seven segment display.

6. Write the truth table for the 7447 IC and obtain the corresponding boolean logic equations.
7. Implement the 7447 logic in the arudino. Verify that your ESP32-devkit-v1 now behaves like the 7447 IC.

4 Karnaugh Map

4.1 *Incrementing Decoder*

We explain Karnaugh maps (K-map) by finding the logic functions for the incrementing decoder

1. The incrementing decoder takes the numbers $0, , \dots, 9$ in binary as inputs and generates the consecutive number as output. The corresponding truth table is available in Table 3.4
2. Using Boolean logic, output A in Table 3.4 can be expressed in terms of the inputs W, X, Y, Z as

$$A = W'X'Y'Z' + W'XY'Z' + W'X'YZ' + W'XYZ' + W'X'Y'Z \tag{4.1}$$

3. K-Map for A: The expression in (4.1) can be minimized using the K-map in Fig. 4.1

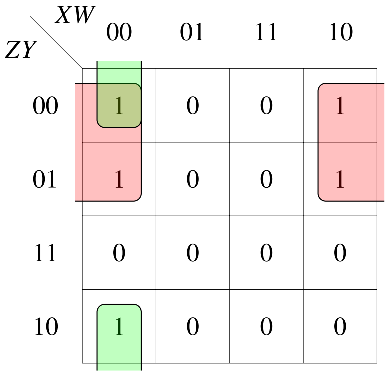

Fig. 4.1: K-map for A

In Fig. 4.1, the *implicants* in boxes 0,2,4,6 result in $W'Z'$. The implicants in boxes 0,8 result in $W'X'Y'$. Thus, after minimization using Fig. 4.1, (4.1) can be expressed as

$$A = W'Z' + W'X'Y' \tag{4.2}$$

Using the fact that

$$\begin{aligned} X + X' &= 1 \\ XX' &= 0, \end{aligned} \tag{4.3}$$

derive (4.2) from (4.1) algebraically

4. K-Map for B: From Table 3.4, using boolean logic,

$$B = WX'Y'Z' + W'XY'Z' + WX'YZ' + W'XYZ' \tag{4.4}$$

Show that (4.4) can be reduced to

$$B = WX'Z' + W'XZ' \tag{4.5}$$

using Fig 4.2

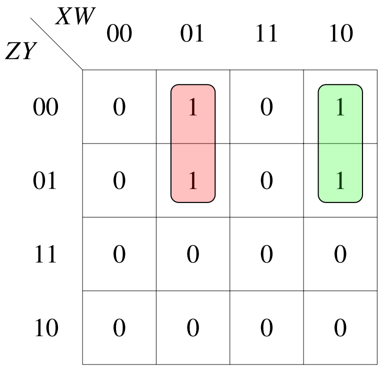

Fig. 4.2: K-map for B

5. Derive (4.5) from (4.4) algebraically using (4.3)
6. K-Map for C: From Table 3.4, using boolean logic,

$$C = WXY'Z' + W'X'YZ' + WX'YZ' + W'XYZ' \tag{4.6}$$

Show that (4.6) can be reduced to

$$C = WXY'Z' + X'YZ' + W'YZ' \tag{4.7}$$

using Fig. 4.3.

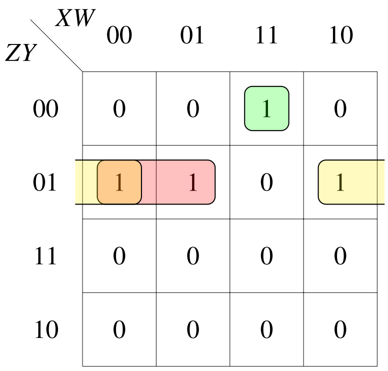

Fig. 4.3: K-map for C

7. Derive (4.7) from (4.6) algebraically using (4.3)

8. K-Map for D: From Table 3.4, using boolean logic,

$$D = WXYZ' + W'X'Y'Z \tag{4.8}$$

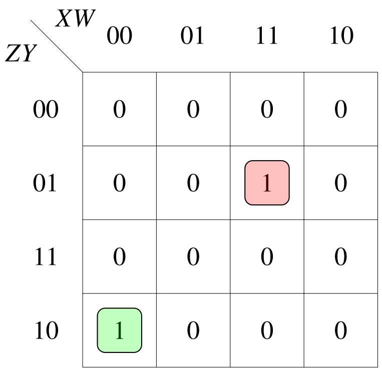

Fig. 4.4: K-map for D

9. Minimize (4.8) using Fig 4.4
10. Execute the code in

```
ide/7447/codes/inc_dec/inc_dec.cpp
```

and modify it using the K-Map equations for A, B, C and D. Execute and verify for each case.
11. Display Decoder: Table 4.1 is the truth table for the display decoder in Fig. 3.1. Use K-maps to obtain the minimized expressions for a, b, c, d, e, f, g in terms of A, B, C, D.

D	C	B	A	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**Decimal**
0	0	0	0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	0
0	0	0	1	**1**	**0**	**0**	**1**	**1**	**1**	**1**	1
0	0	1	0	**0**	**0**	**1**	**0**	**0**	**1**	**0**	2
0	0	1	1	**0**	**0**	**0**	**0**	**1**	**1**	**0**	3
0	1	0	0	**1**	**0**	**0**	**1**	**1**	**0**	**0**	4
0	1	0	1	**0**	**1**	**0**	**0**	**1**	**0**	**0**	5
0	1	1	0	**0**	**1**	**0**	**0**	**0**	**0**	**0**	6
0	1	1	1	**0**	**0**	**0**	**1**	**1**	**1**	**1**	7
1	0	0	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	8
1	0	0	1	**0**	**0**	**0**	**1**	**1**	**0**	**0**	9

TABLE 4.1: Truth table for display decoder.

4.2 Dont Care

We explain Karnaugh maps (K-map) using don't care conditions

1. Don't Care Conditions: 4 binary digits are used in the incrementing decoder in Table 4.1. However, only the numbers from 0-9 are used as input/output in the decoder and we *don't care* about the numbers from 10-15. This phenomenon can be addressed by revising the truth table in Table 4.1 to obtain Table 4.2.

Z	Y	X	W	**D**	**C**	**B**	**A**
0	0	0	0	**0**	**0**	**0**	**1**
0	0	0	1	**0**	**0**	**1**	**0**
0	0	1	0	**0**	**0**	**1**	**1**
0	0	1	1	**0**	**1**	**0**	**0**
0	1	0	0	**0**	**1**	**0**	**1**
0	1	0	1	**0**	**1**	**1**	**0**
0	1	1	0	**0**	**1**	**1**	**1**
0	1	1	1	**1**	**0**	**0**	**0**
1	0	0	0	**1**	**0**	**0**	**1**
1	0	0	1	**0**	**0**	**0**	**0**
1	0	1	0	**-**	**-**	**-**	**-**
1	0	1	1	**-**	**-**	**-**	**-**
1	1	0	0	**-**	**-**	**-**	**-**
1	1	0	1	**-**	**-**	**-**	**-**
1	1	1	0	**-**	**-**	**-**	**-**
1	1	1	1	**-**	**-**	**-**	**-**

TABLE 4.2

2. The revised K-map for A is available in Fig 4.5. Show that

$$A = W' \tag{4.9}$$

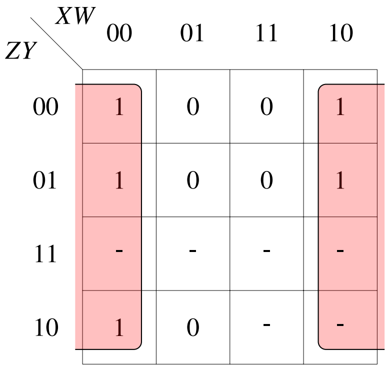

Fig. 4.5: K-map for A with don't cares

3. The revised K-map for B is available in Fig 4.6. Show that

$$B = WX'Z' + W'X \tag{4.10}$$

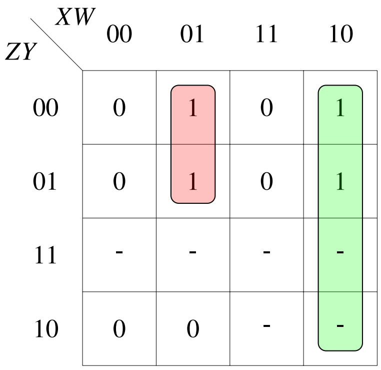

Fig. 4.6: K-map for B with don't cares

4. The revised K-map for C is available in Fig 4.7. Show that

$$C = X'Y + W'Y + WXY' \tag{4.11}$$

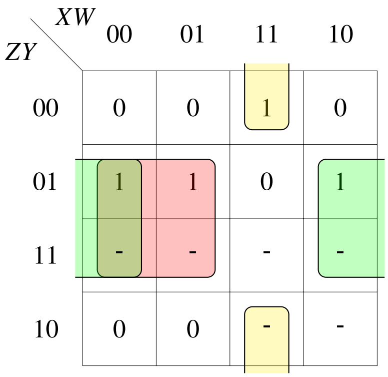

Fig. 4.7: K-map for C with don't cares

5. The revised K-map for D is available in Fig 4.8. Show that

$$D = W'Z + WXY \tag{4.12}$$

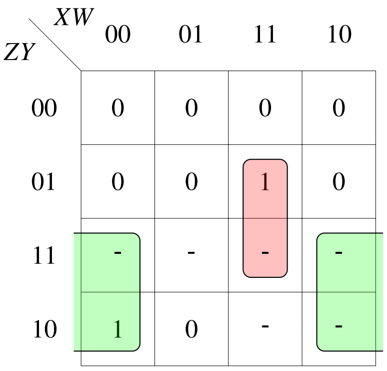

Fig. 4.8: K-map for D with don't cares

6. Verify the incrementing decoder with don't care conditions using the arduino.
7. Display Decoder: In Table 4.1, use K-maps to obtain the minimized expressions for a, b, c, d, e, f, g in terms of A, B, C, D with don't care conditions. Verify using ESP32-devkit-v1.

5 7474

We show how to use the 7474 D-Flip Flop ICs in a sequential circuit to realize a decade counter.

1. Generate the CLOCK signal using the **blink** program in the arduino.
2. Connect the Arduino, 7447 and the two 7474 ICs according to Table 5.1 and Fig. 5.2. The pin diagram for 7474 is available in Fig. 5.1

	INPUT				OUTPUT				CLOCK		3.3V			
	W	X	Y	Z	A	B	C	D						
ESP32	D32	D33	D25	D26	D13	D12	D14	D27	D2					
7474	5	9			2	12			CLK1	CLK2	1	4	10	13
7474			5	9			2	12	CLK1	CLK2	1	4	10	13
7447					7	1	2	6			16			

TABLE 5.1

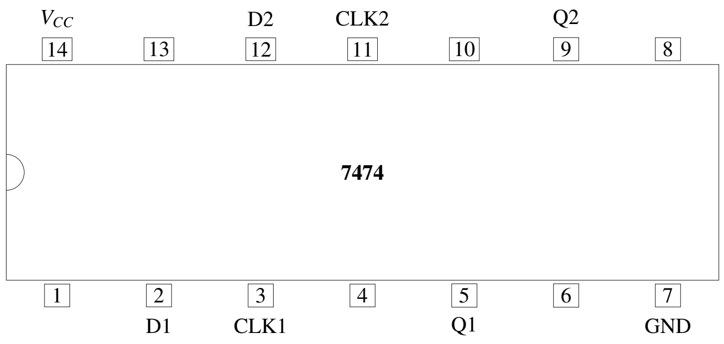

Fig. 5.1

3. Intelligently use the codes in

```
ide/7447/codes/inc_dec/inc_dec.ino
```

and

```
ide/7447/codes/inc_dec/ip_inc_dec.ino
```

to realize the decade counter in Fig. 5.2.

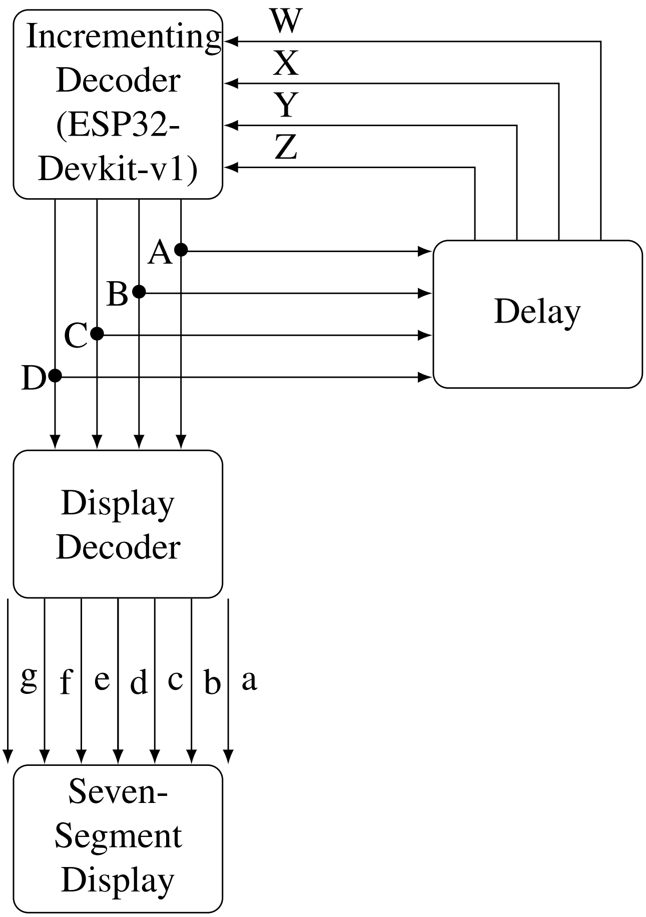
Incrementing Decoder (ESP32-Devkit-v1)
W
X
Y
Z
A
B
C
D
Delay
Display Decoder
g
f
e
d
c
b
a
Seven-Segment Display

Fig. 5.2

6 Finite State Machine

We explain a state machine by deconstructing the decade counter

The block diagram of a decade counter (repeatedly counts up from 0 to 9) is available in Fig. 5.2. The *incrementing* decoder and *display* decoder are part of *combinational* logic, while the *delay* is part of *sequential* logic.

1. Fig. 6.1 shows a *finite state machine* (FSM) diagram for the decade counter in Fig 5.2 s_0 is the state when the input to the incrementing decoder is 0. The *state transition table* for the FSM is Table 3.4, where the present state is denoted by the variables W, X, Y, Z and the next state by A, B, C, D.

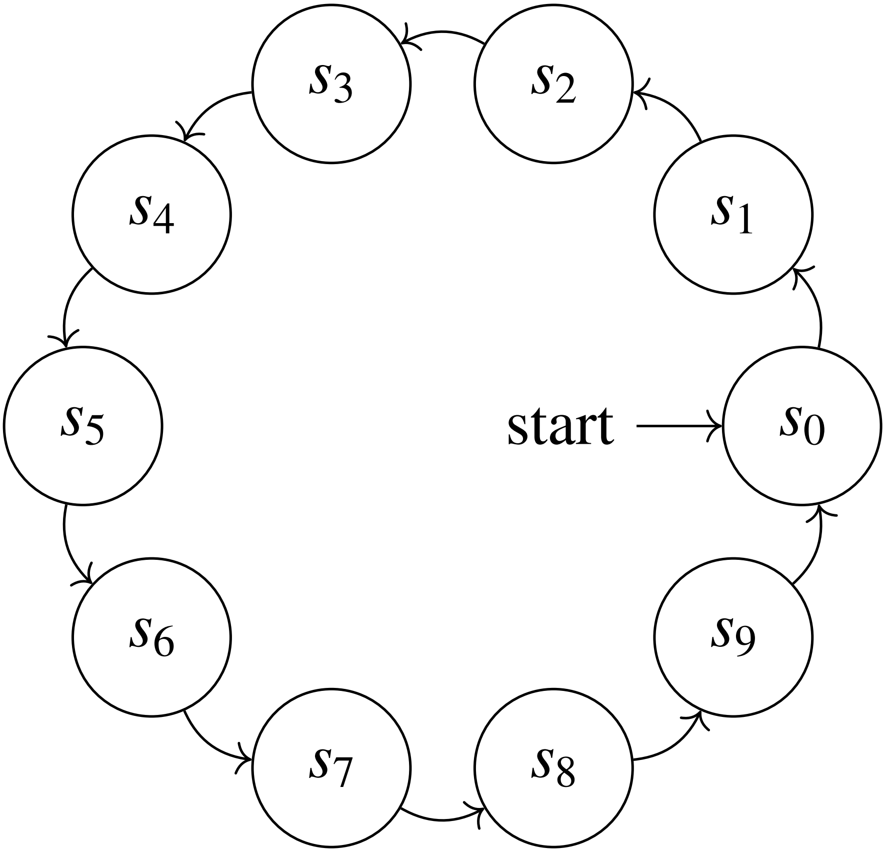

Fig. 6.1: FSM for the decade counter

2. The FSM implementation is available in Fig 6.2 The *flip-flops* hold the input for the time that is given by the *clock* This is nothing but the implementation of the *Delay* block in Fig 5.2

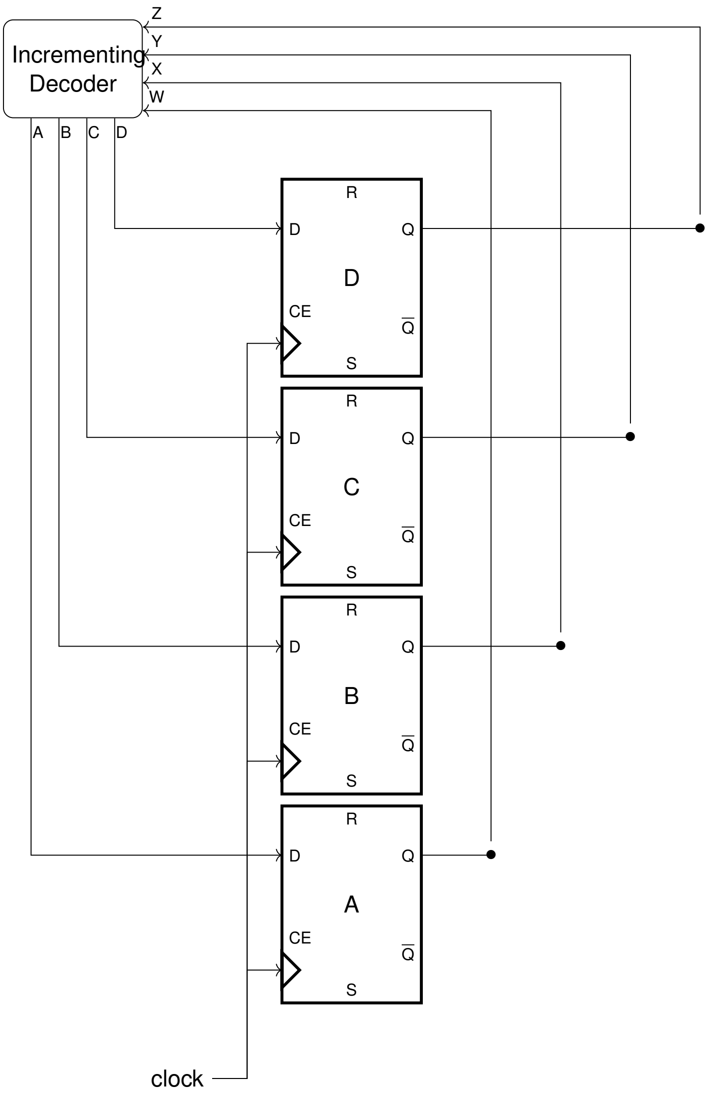

Fig. 6.2: Decade counter FSM implementation using D-Flip Flops

3. The hardware cost of the system is given by

$$\text{No of D Flip-Flops} = \lceil \log_2(\text{No of States}) \rceil \tag{6.1}$$

   For the FSM in Fig 6.1, the number of states is 9, hence the number of flip flops required = 4
4. Draw the state transition diagram for a decade down counter (counts from 9 to 0 repeatedly) using an FSM.
5. Write the state transition table for the down counter.

6. Obtain the state transition equations with and without don’t cares.
7. Verify your design using an ESP32-Devkit-v1.

7 OTA

Here we show how to program ESP32 through OTA using platformio. The below link guides to the codes required for implementation of ESP32 through OTA

```
https://github.com/pradyumna963/afw/tree/main/ide/ota
```

1. Follow the steps 1 and 2 given in section 1.2.
2. Open the main.cpp file by excecting the below command in termux and change the SSID and password mentioned in the main code.

```
nvim /ide/ota/blink/src/main
```

3. In termux excecte the following to generate the bin file:

```
cd ide/ota/blink
pio run
```

4. Create a folder in AruinoDroid/precompiled dirctory and copy paste the bin file to this folder by excecting the following commands:

```
mkdir /sdcard/Arduinodroid/precompiled/blink_ota
cp ls .pio/build/esp32doit-devkit-v1/firmware.bin /sdcard/Arduinodroid/
      precompiled/blink_ota
```

5. For flashing the bin files, open ArduinoDroid

```
Actions->Upload->Upload Precompiled
```

   then select

```
blink_ota->firmware.bin
```

   for uploading the bin bile to the ESP32.
6. After the uploading is finished you will get the following in the terminal.

```
Error: open failed: ENOENT (No such file or directory)
```

   Disconnect the power supply and reconnect it, then the inbuilt led will start blink. The ESP32 will also be connected to the wifi network that is mntioned in the code.
7. For the next program that needs to be flashed you can upload the code through OTA by excecuting the following commands in termux following commands

```
cd -
cd ide/ota/sevenseg/static
pio run -t upload --upload-port 192.168.0.0 #use the IP address of the ESP32
```