香港中文大學
The Chinese University of Hong Kong

CENG2400 Embedded System Design
Lab 04:
Keypad and LCD

Han ZHAO, Zhirui ZHANG

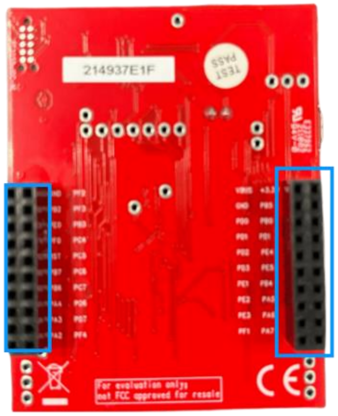
hzhao@cse.cuhk.edu.hk

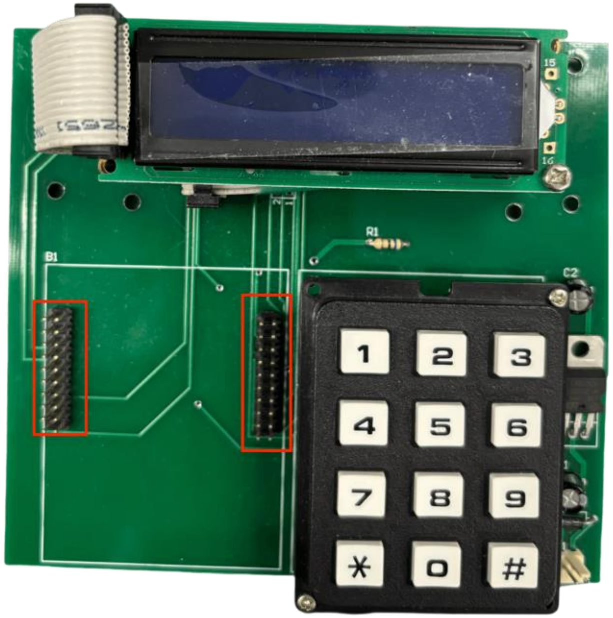
Outline

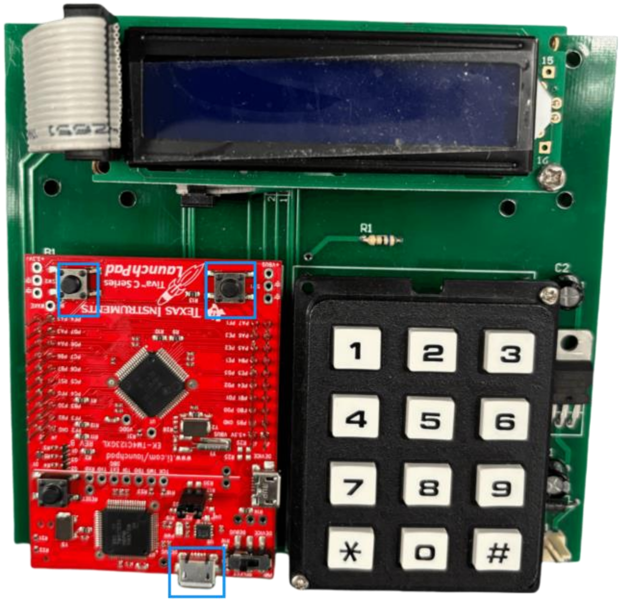
- **Part 0:** Signing in and collecting equipment
  - **Two students** share a Tiva Launchpad and a set of Keypad and LCD.
- **Part 1:** Reviewing the basics of Keypad and LCD
  - Most of the contents have been taught in Monday's lectures.
- **Part 2:** Running an example
- **Part 3:** Completing your assignment
  - Two people can work on the code as a group, but please upload your **code** and **video** on Blackboard before the next lab (**22 Oct.**) individually.

Part 1: Basics of Keypad and LCD

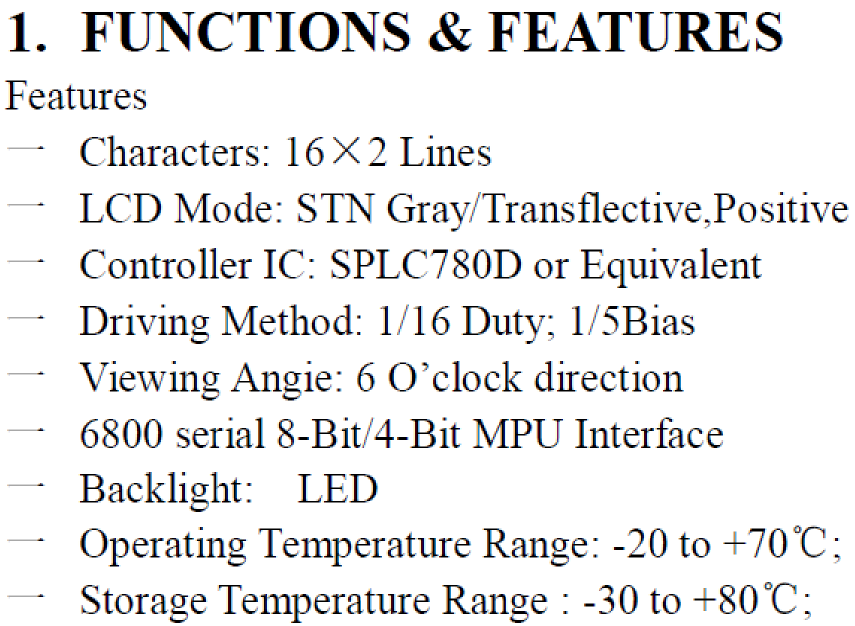
- We are going to integrate Tiva Launchpad with a **Keypad** and a **LCD display**.
  - Insert the **two rows of pins** on the board into the black sockets on the reverse side of the Tiva Launchpad.
  - The **two switches** on the Tiva Launchpad shall locate at the **top**, while the **USB connector** is at the **bottom**.







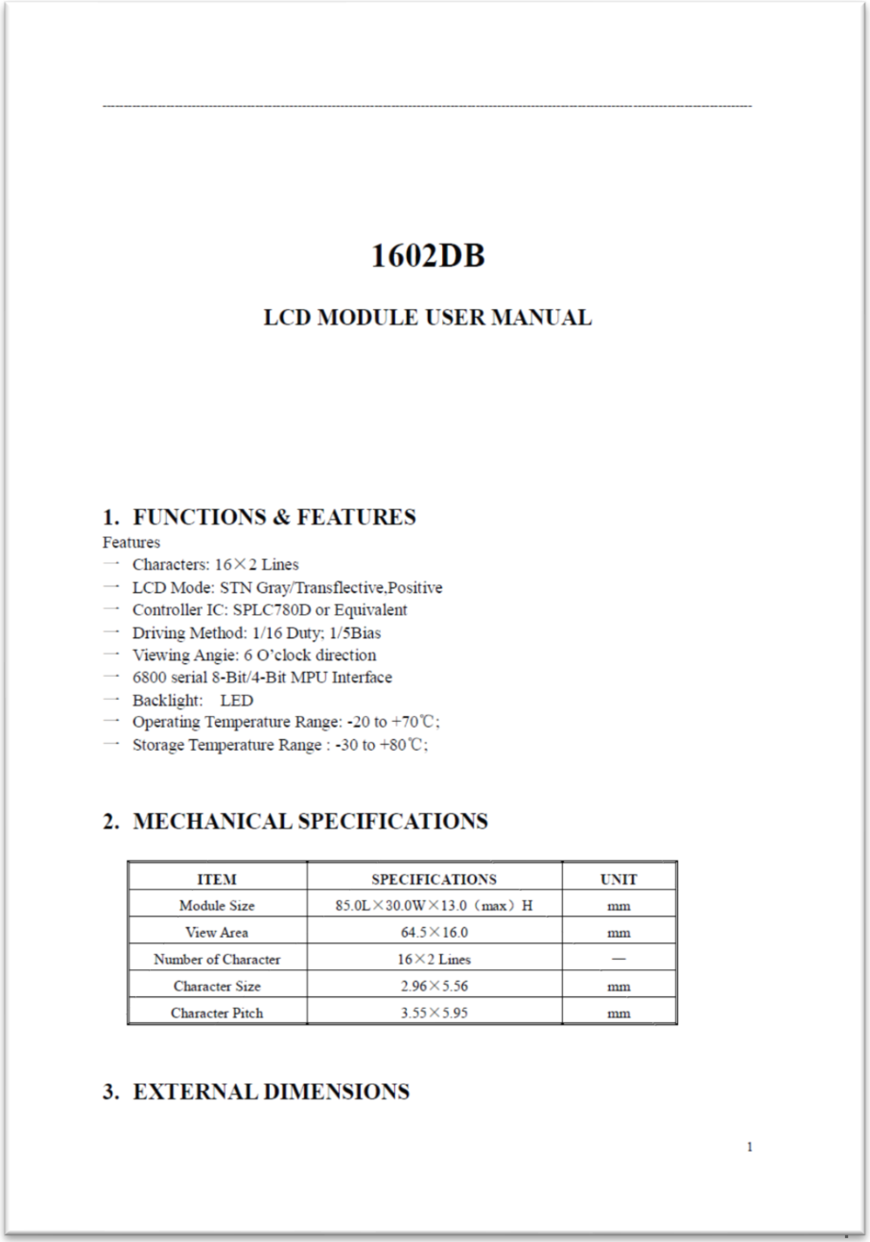
LCD Display: 1602DB LCD Module



1. FUNCTIONS & FEATURES

Features

- Characters: 16×2 Lines
- LCD Mode: STN Gray/Transflective,Positive
- Controller IC: SPLC780D or Equivalent
- Driving Method: 1/16 Duty; 1/5Bias
- Viewing Angie: 6 O'clock direction
- 6800 serial 8-Bit/4-Bit MPU Interface
- Backlight: LED
- Operating Temperature Range: -20 to +70℃;
- Storage Temperature Range : -30 to +80℃;



1602DB

LCD MODULE USER MANUAL

1. FUNCTIONS & FEATURES

Features

- Characters: 16×2 Lines
- LCD Mode: STN Gray/Transflective,Positive
- Controller IC: SPLC780D or Equivalent
- Driving Method: 1/16 Duty; 1/5Bias
- Viewing Angie: 6 O'clock direction
- 6800 serial 8-Bit/4-Bit MPU Interface
- Backlight: LED
- Operating Temperature Range: -20 to +70℃;
- Storage Temperature Range : -30 to +80℃;

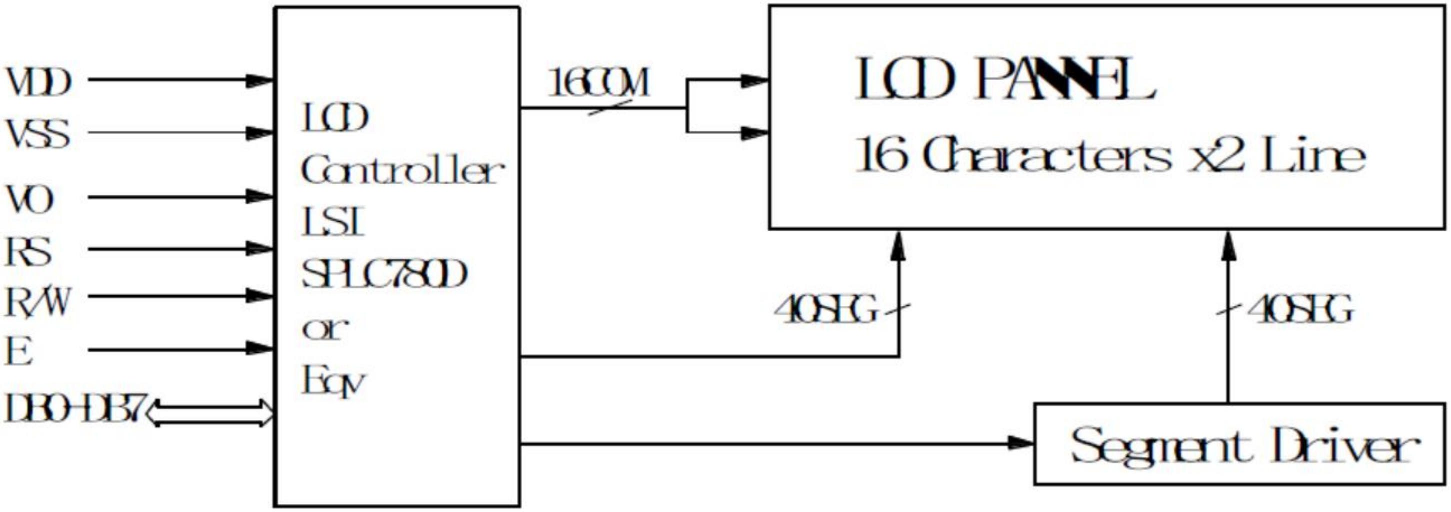
2. MECHANICAL SPECIFICATIONS

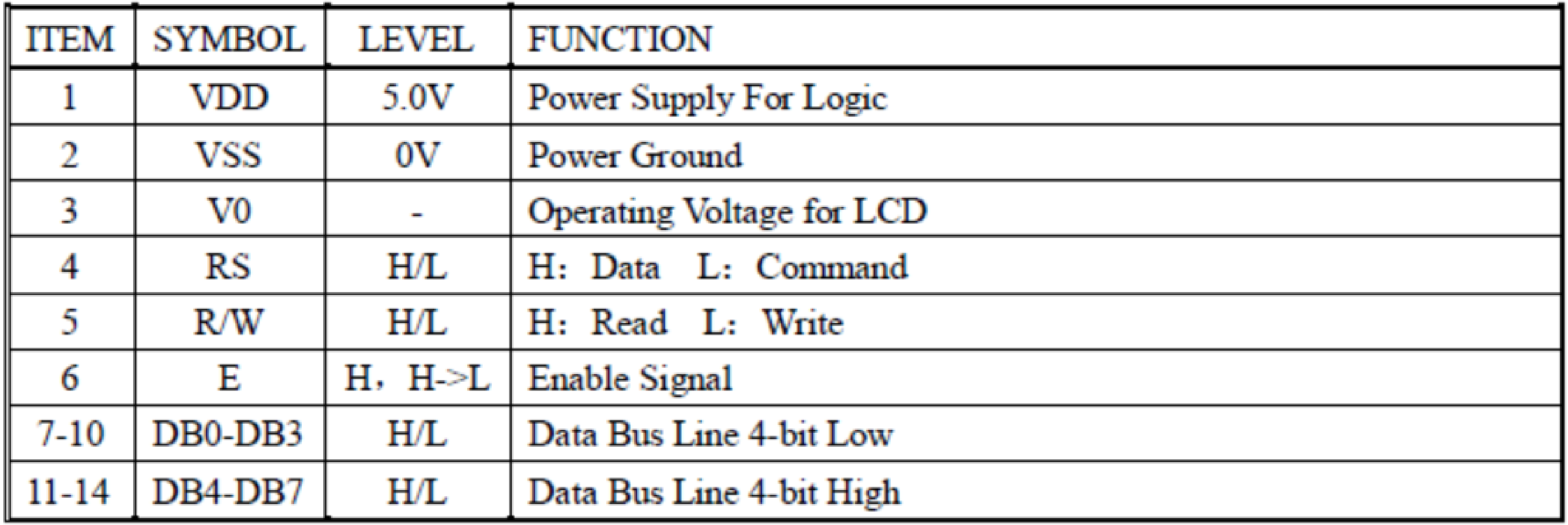
ITEM	SPECIFICATIONS	UNIT
Module Size	85.0L×30.0W×13.0（max）H	mm
View Area	64.5×16.0	mm
Number of Character	16×2 Lines	—
Character Size	2.96×5.56	mm
Character Pitch	3.55×5.95	mm

3. EXTERNAL DIMENSIONS

1

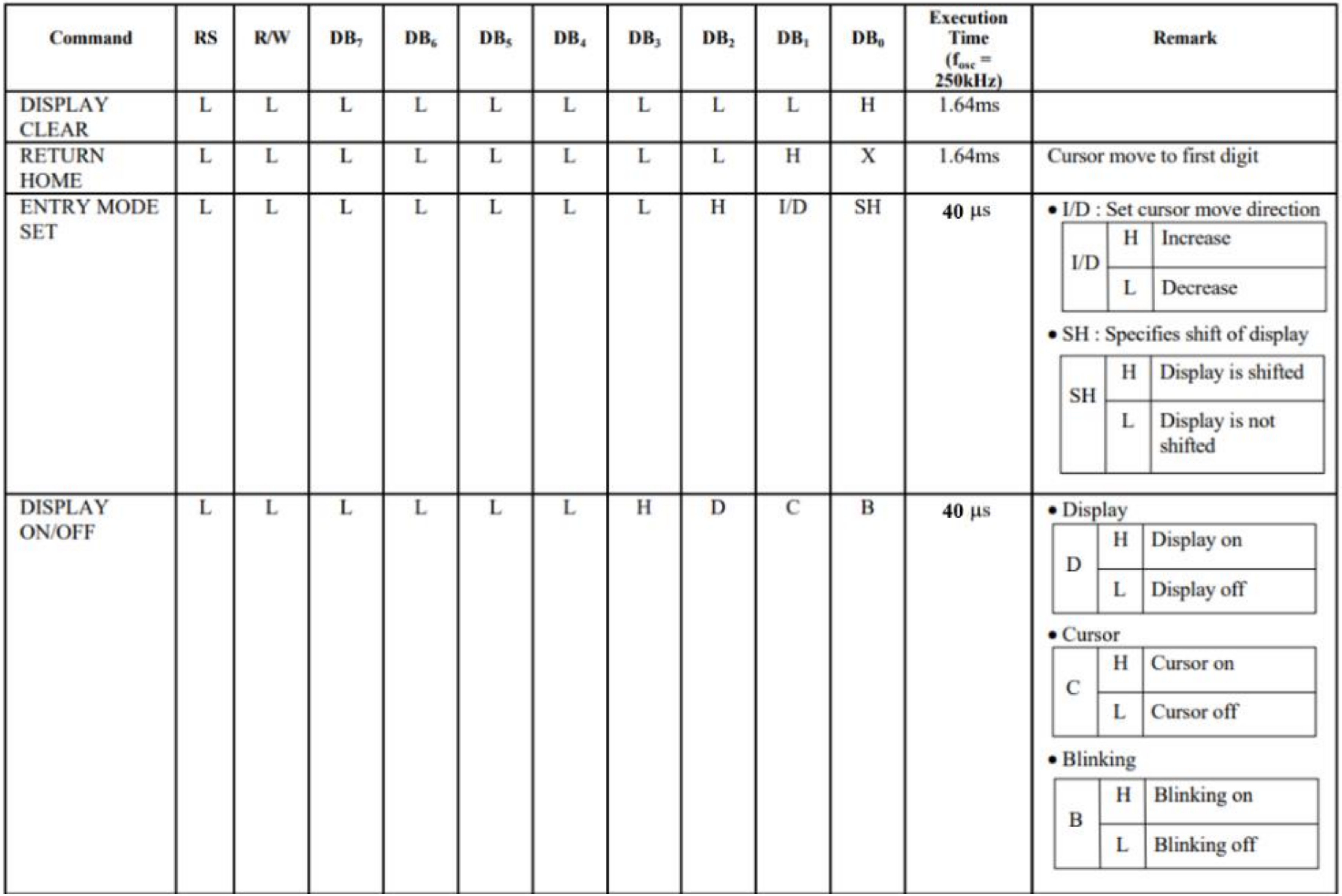
LCD Display: Block Diagram





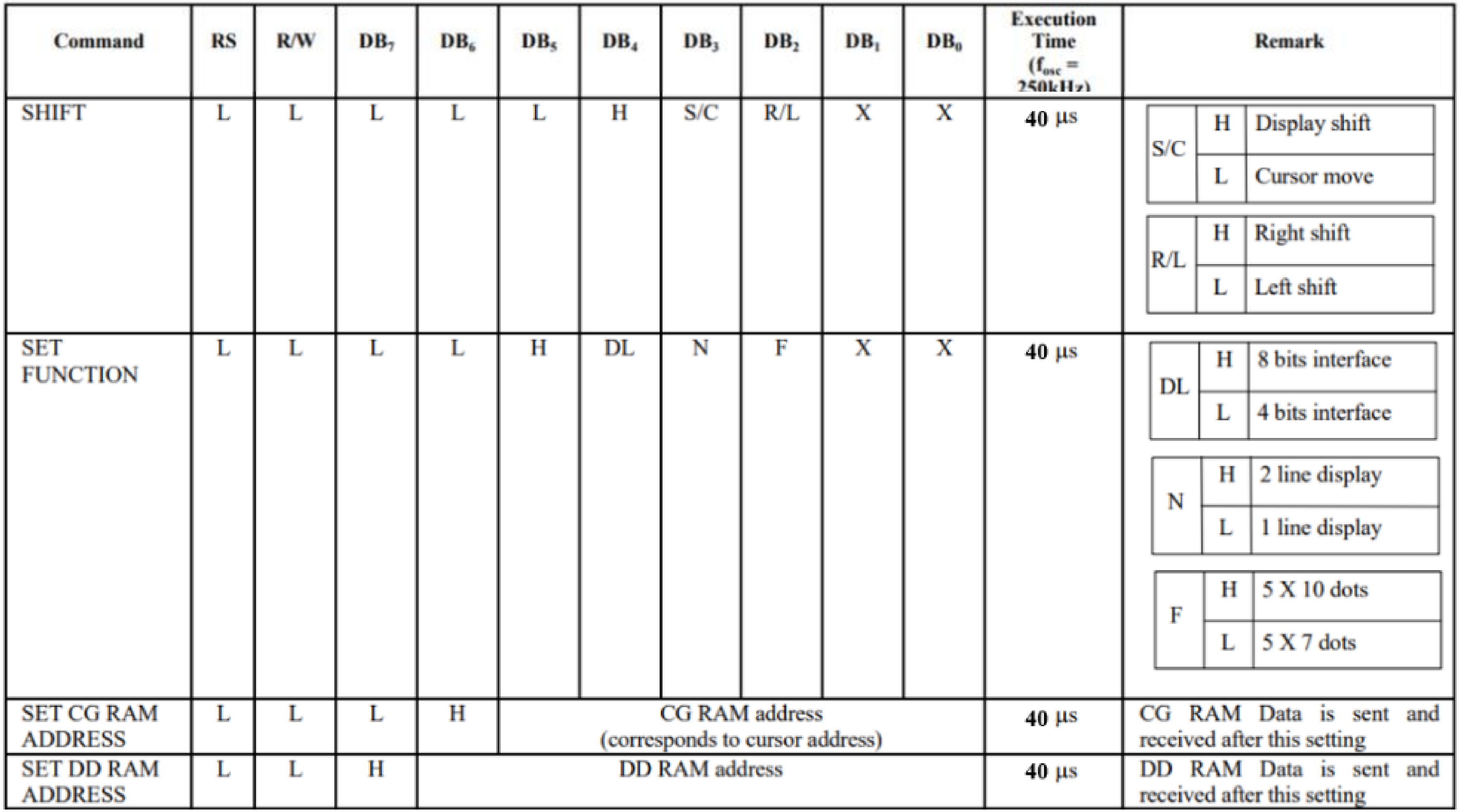
ITEM	SYMBOL	LEVEL	FUNCTION
1	VDD	5.0V	Power Supply For Logic
2	VSS	0V	Power Ground
3	V0	-	Operating Voltage for LCD
4	RS	H/L	H: Data L: Command
5	R/W	H/L	H: Read L: Write
6	E	H，H->L	Enable Signal
7-10	DB0-DB3	H/L	Data Bus Line 4-bit Low
11-14	DB4-DB7	H/L	Data Bus Line 4-bit High

LCD Display: Commands (1/3)



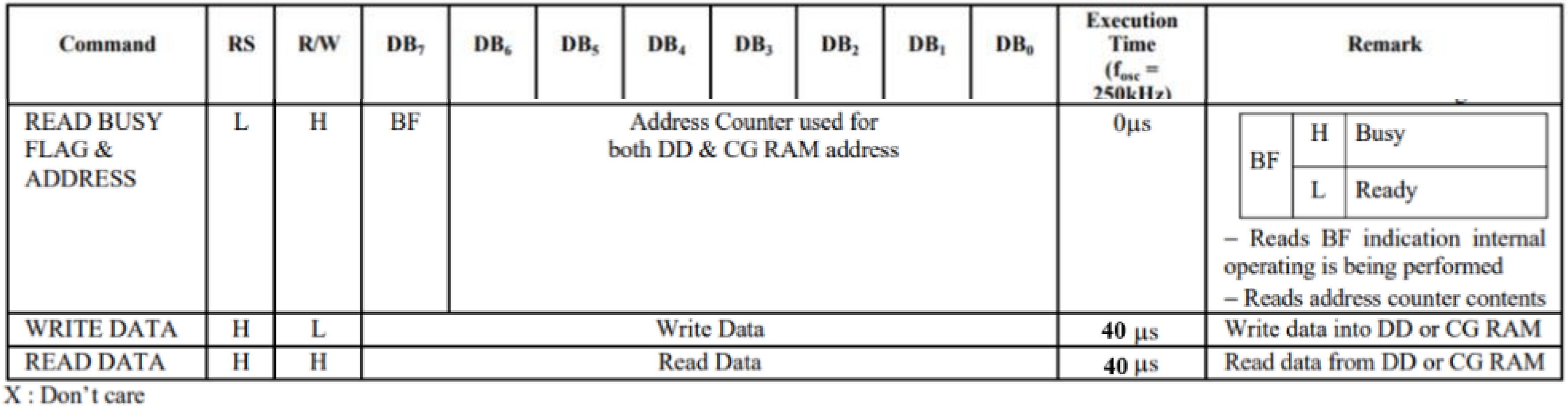
Command	RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0	Execution Time (f_{osc} = 250kHz)	Remark
DISPLAY CLEAR	L	L	L	L	L	L	L	L	L	H	1.64ms	
RETURN HOME	L	L	L	L	L	L	L	L	H	X	1.64ms	Cursor move to first digit
ENTRY MODE SET	L	L	L	L	L	L	L	H	I/D	SH	40 μs	• I/D : Set cursor move direction I/D: H – Increase; L – Decrease • SH : Specifies shift of display SH: H – Display is shifted; L – Display is not shifted
DISPLAY ON/OFF	L	L	L	L	L	L	H	D	C	B	40 μs	• Display D: H – Display on; L – Display off • Cursor C: H – Cursor on; L – Cursor off • Blinking B: H – Blinking on; L – Blinking off

LCD Display: Commands (2/3)



Command	RS	R/W	DB$_7$	DB$_6$	DB$_5$	DB$_4$	DB$_3$	DB$_2$	DB$_1$	DB$_0$	Execution Time (f$_{osc}$ = 250kHz)	Remark
SHIFT	L	L	L	L	L	H	S/C	R/L	X	X	40 µs	S/C: H Display shift S/C: L Cursor move R/L: H Right shift R/L: L Left shift
SET FUNCTION	L	L	L	L	H	DL	N	F	X	X	40 µs	DL: H 8 bits interface DL: L 4 bits interface N: H 2 line display N: L 1 line display F: H 5 X 10 dots F: L 5 X 7 dots
SET CG RAM ADDRESS	L	L	L	H	CG RAM address (corresponds to cursor address)						40 µs	CG RAM Data is sent and received after this setting
SET DD RAM ADDRESS	L	L	H	DD RAM address							40 µs	DD RAM Data is sent and received after this setting

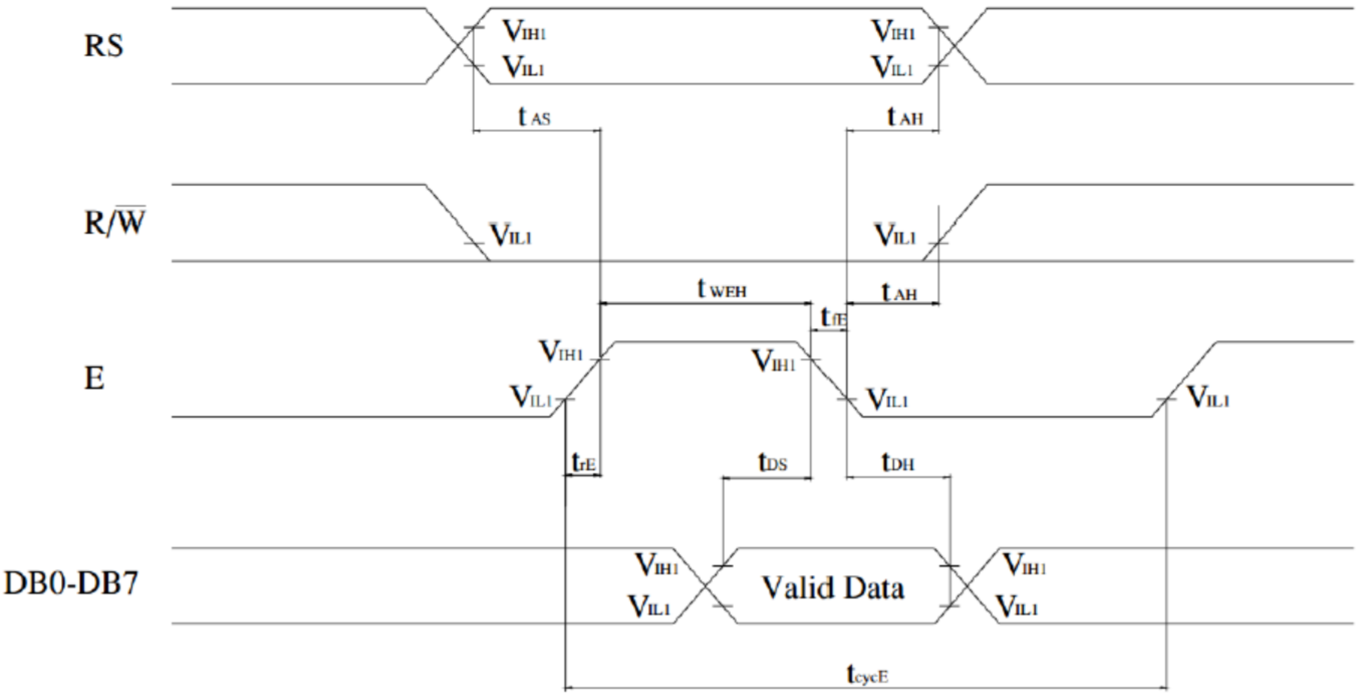
LCD Display: Commands (3/3)

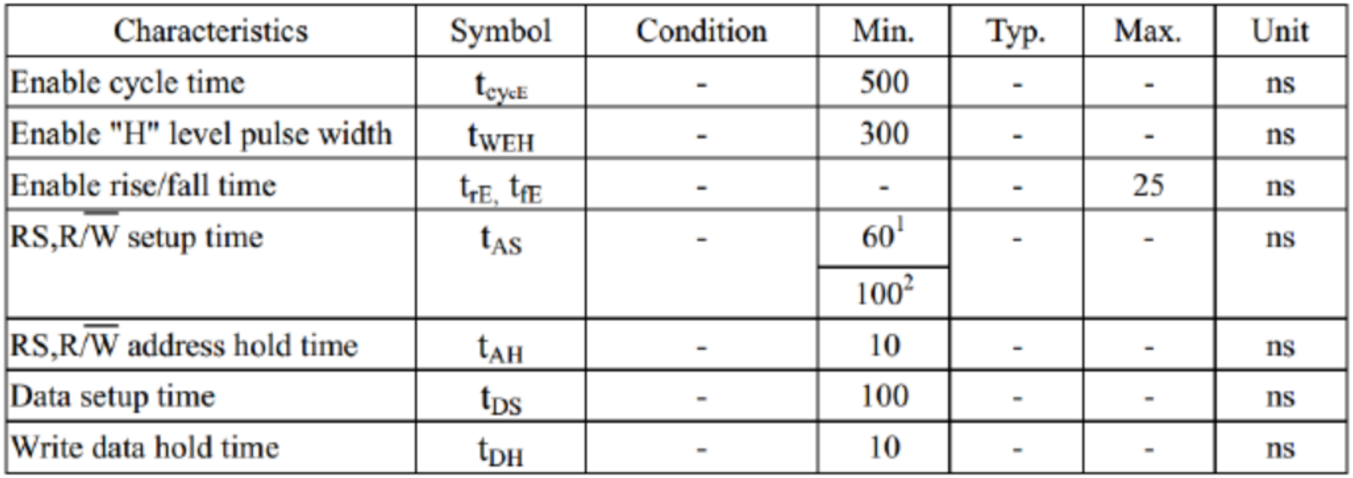


Command	RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0	Execution Time (f_{osc} = 250kHz)	Remark
READ BUSY FLAG & ADDRESS	L	H	BF	Address Counter used for both DD & CG RAM address							0µs	BF H Busy L Ready – Reads BF indication internal operating is being performed – Reads address counter contents
WRITE DATA	H	L	Write Data								40 µs	Write data into DD or CG RAM
READ DATA	H	H	Read Data								40 µs	Read data from DD or CG RAM

X : Don’t care

LCD Display: Write Timing





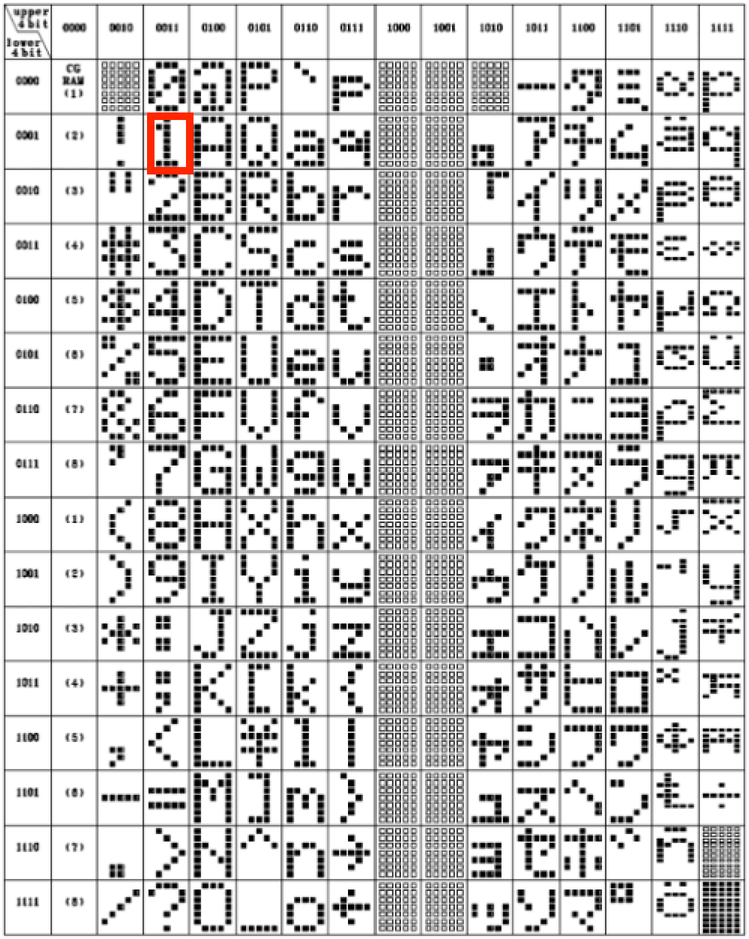
Characteristics	Symbol	Condition	Min.	Typ.	Max.	Unit
Enable cycle time	t_{cycE}	-	500	-	-	ns
Enable "H" level pulse width	t_{WEH}	-	300	-	-	ns
Enable rise/fall time	t_{rE}, t_{fE}	-	-	-	25	ns
RS,R/$\overline{W}$ setup time	t_{AS}	-	60[1] 100[2]	-	-	ns
RS,R/$\overline{W}$ address hold time	t_{AH}	-	10	-	-	ns
Data setup time	t_{DS}	-	100	-	-	ns
Write data hold time	t_{DH}	-	10	-	-	ns

LCD Display: Command Implementation

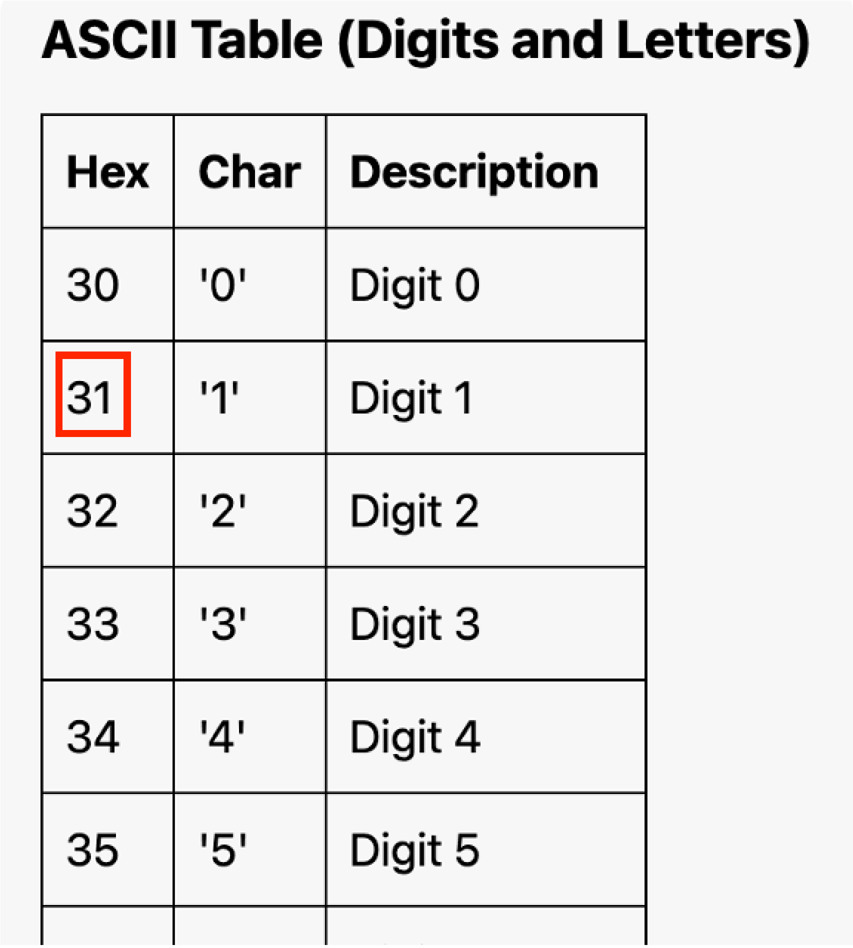
```
#define RS_PIN GPIO_PIN_5   // select pin 5 for RS
#define RW_PIN GPIO_PIN_6   // select pin 6 for RW
#define EN_PIN GPIO_PIN_7   // select pin 7 for EN
#define DB_PIN GPIO_PIN_0 | GPIO_PIN_1 | ... | GPIO_PIN_7 // select pins 0~7 for DB

void LCD_command (bool rs, bool rw, unsigned char data)
{
    if (rs == 0) // L: Command H: Data
        GPIOPinWrite(GPIO_PORTA_BASE, RS_PIN, 0x00); // set RS as L
    else
        GPIOPinWrite(GPIO_PORTA_BASE, RS_PIN, 0x20); // set RS as H
    if (rw == 0) // L: Write mode; H: Read mode
        GPIOPinWrite(GPIO_PORTA_BASE, RW_PIN, 0x00); // set RW as L
    else
        GPIOPinWrite(GPIO_PORTA_BASE, RW_PIN, 0x40); // set RW as H
    delayUs(1);
    GPIOPinWrite(GPIO_PORTA_BASE, EN_PIN, 0x80); // set H to enable signal EN
    GPIOPinWrite(GPIO_PORTB_BASE, DB_PIN, data); // assign DB0~DB7 with "data"
    delayUs(1);
    GPIOPinWrite(GPIO_PORTA_BASE, EN_PIN, 0x00); // set H->L to enable signal EN
    delayUs(1);
    if (rs == 0) // L: Command
    {
        if ((data == 0x01) | (data == 0x02) | (data == 0x03))
            delayUs(1640); // Clear Display & Display/Cursor Home take 1.64ms
        else
            delayUs(40);   // all the others commands require only 40us to execute
    }
    else
        delayUs(40); // Data Write takes 40us to execute
}
```

LCD Display: Standard Character

- We can directly feed a character in the **`LCD_command`**.
  - E.g., **`LCD_command(1, 0, ‘1’);`** `// write a ‘1’ on LCD`
  - Why? The LCD follows the standard character (ASCII code).



upper 4 bit / lower 4 bit	0000	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	CG RAM (1)		0	@	P	`	p				ー	タ	ミ	α	p
0001	(2)	!	1	A	Q	a	q			。	ア	チ	ム	ä	q
0010	(3)	"	2	B	R	b	r			「	イ	ツ	メ	β	θ
0011	(4)	#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
0100	(5)	$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
0101	(6)	%	5	E	U	e	u			・	オ	ナ	ユ	σ	ü
0110	(7)	&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
0111	(8)	'	7	G	W	g	w			ァ	キ	ヌ	ラ	g	π
1000	(1)	(	8	H	X	h	x			ィ	ク	ネ	リ	√	x̄
1001	(2)	)	9	I	Y	i	y			ゥ	ケ	ノ	ル	⁻¹	y
1010	(3)	*	:	J	Z	j	z			ェ	コ	ハ	レ	j	千
1011	(4)	+	;	K	[	k	{			ォ	サ	ヒ	ロ	ˣ	万
1100	(5)	,	<	L	¥	l	|			ャ	シ	フ	ワ	¢	円
1101	(6)	-	=	M	]	m	}			ュ	ス	ヘ	ン	£	÷
1110	(7)	.	>	N	^	n	→			ョ	セ	ホ	゛	ñ	
1111	(8)	/	?	O	_	o	←			ッ	ソ	マ	゜	ö	█



ASCII Table (Digits and Letters)

Hex	Char	Description
30	'0'	Digit 0
31	'1'	Digit 1
32	'2'	Digit 2
33	'3'	Digit 3
34	'4'	Digit 4
35	'5'	Digit 5

Class Exercise 4.1

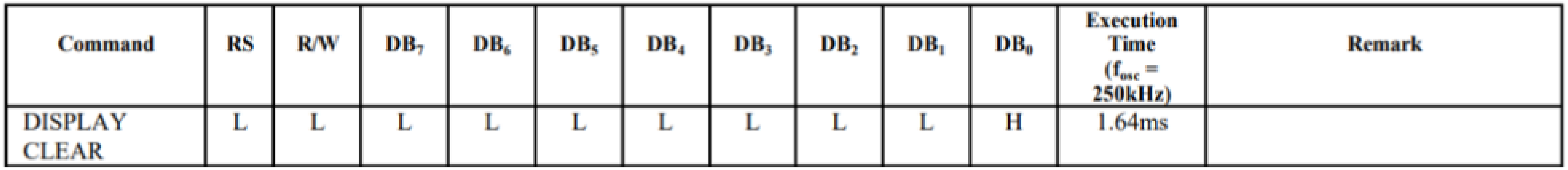
- Determine how to use the implemented **LCD_command** function to ❶ **Clear Display** and ❷ **Write a Char ‘?’** to the LCD.

```
void LCD_command (bool rs, bool rw, unsigned char data);
```

Class Exercise 4.1 (Answer)

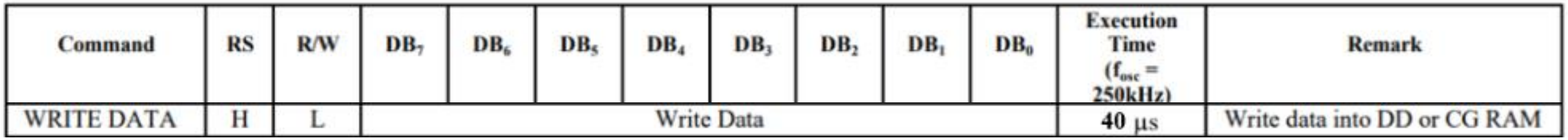
```
void LCD_command (bool rs, bool rw, unsigned char data);
```

- ❶ **Clear Display**
  - `LCD_command(0, 0, 0x01);`



Command	RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0	Execution Time (f_{osc} = 250kHz)	Remark
DISPLAY CLEAR	L	L	L	L	L	L	L	L	L	H	1.64ms	

- ❷ **Write a Char ‘?’** to the LCD
  - `LCD_command(1, 0, ‘?’);`

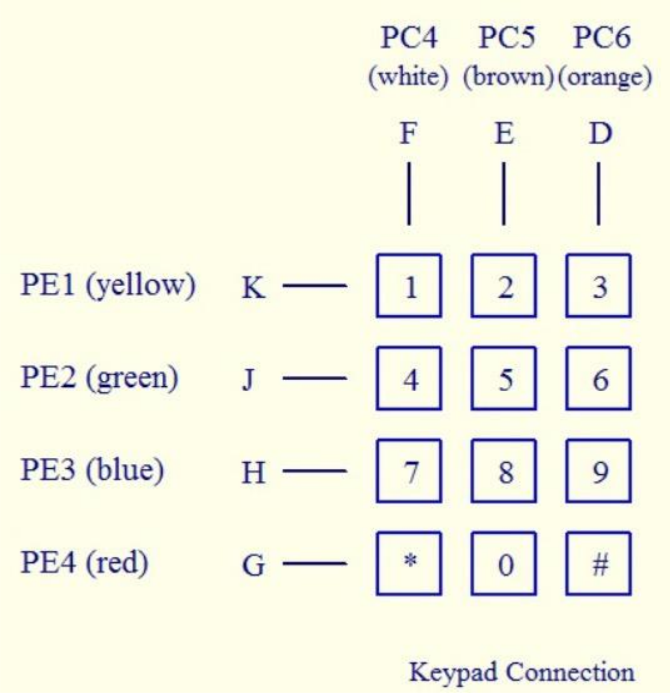


Command	RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0	Execution Time (f_{osc} = 250kHz)	Remark
WRITE DATA	H	L	Write Data								40 µs	Write data into DD or CG RAM

Keypad

- Keyboard are organized in a matrix of rows/columns.
  - When a key is pressed, a row and column make a contact;
  - Otherwise, there is no connection between rows & columns.

7 pins (4 + 3)

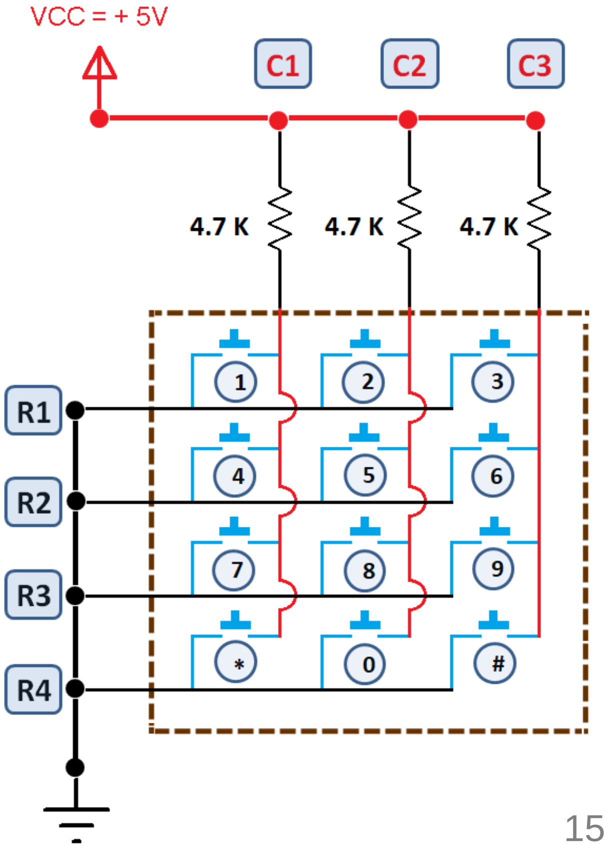


		PC4 (white)	PC5 (brown)	PC6 (orange)
		F	E	D
PE1 (yellow)	K	1	2	3
PE2 (green)	J	4	5	6
PE3 (blue)	H	7	8	9
PE4 (red)	G	*	0	#

Keypad Connection

Keypad: Matrix Control

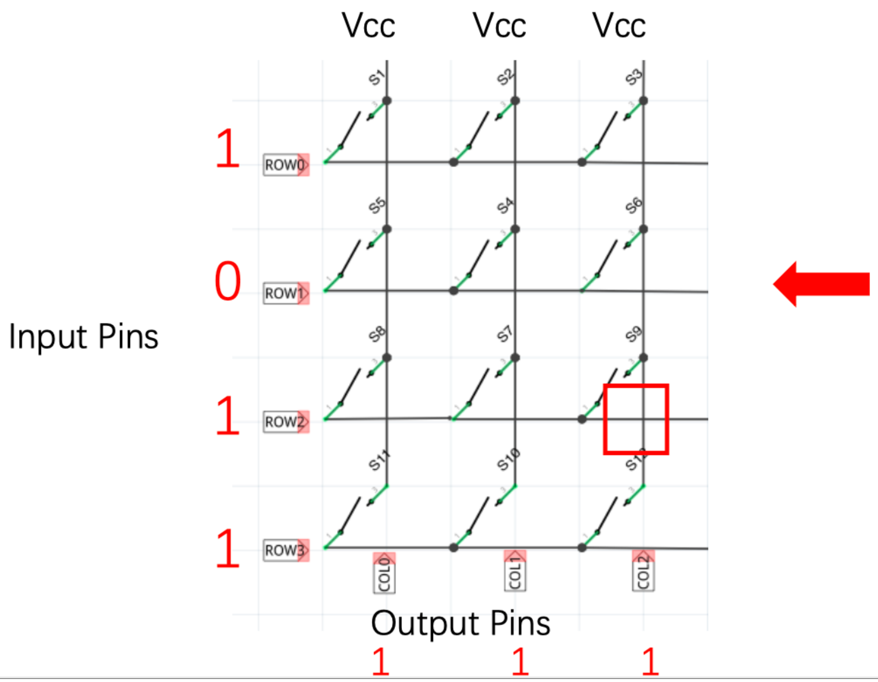
- Pull-up resistors are connected to columns to ensure that they will be in a HIGH state if no key is pressed.
- If we ground a row, pressing a key in that row will connect the associated column to ground (LOW).
- **Algorithm**: Drive one row as LOW at a time; then check the columns for a LOW to determine if any key is pressed.
  - If all ones: *no key in that row is pressed.*
  - If there is zero(s): *the key(s) on the associated column(s), intersected with that row, is pressed.*

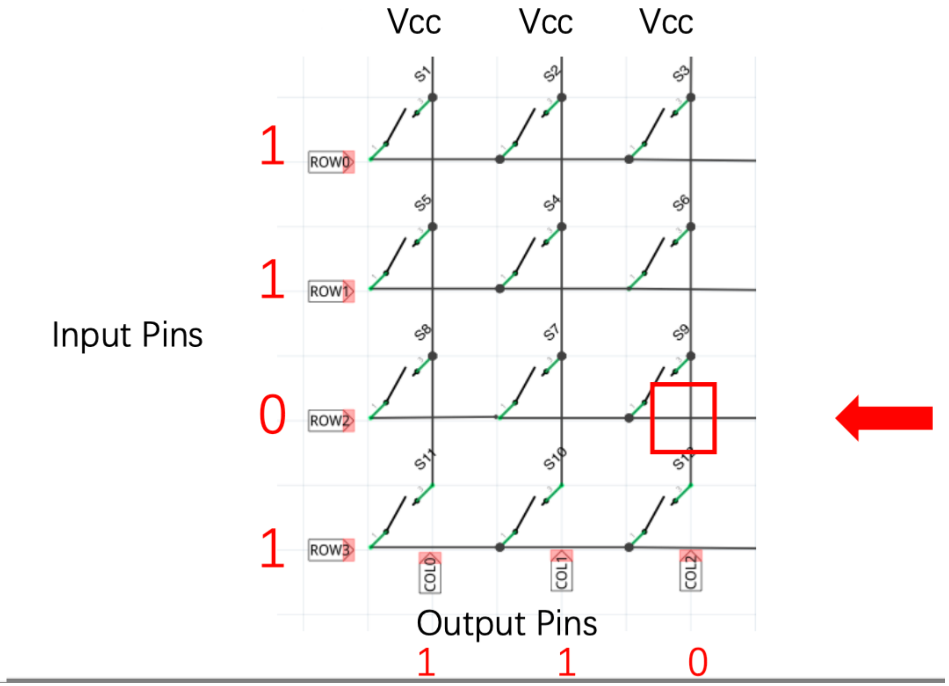


Keypad: Detection Examples

- Drive row 1 as low: 1101
- As no key pressed in row 1, we get 111 by reading columns 0~2.

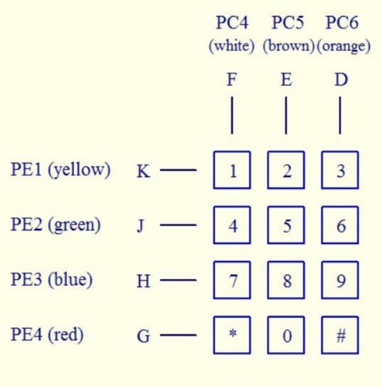
- Drive row 2 as low: 1011
- As there is key on row 2 pressed,, we get 011 by reading columns 0~2.





Keypad: Detection Implementation

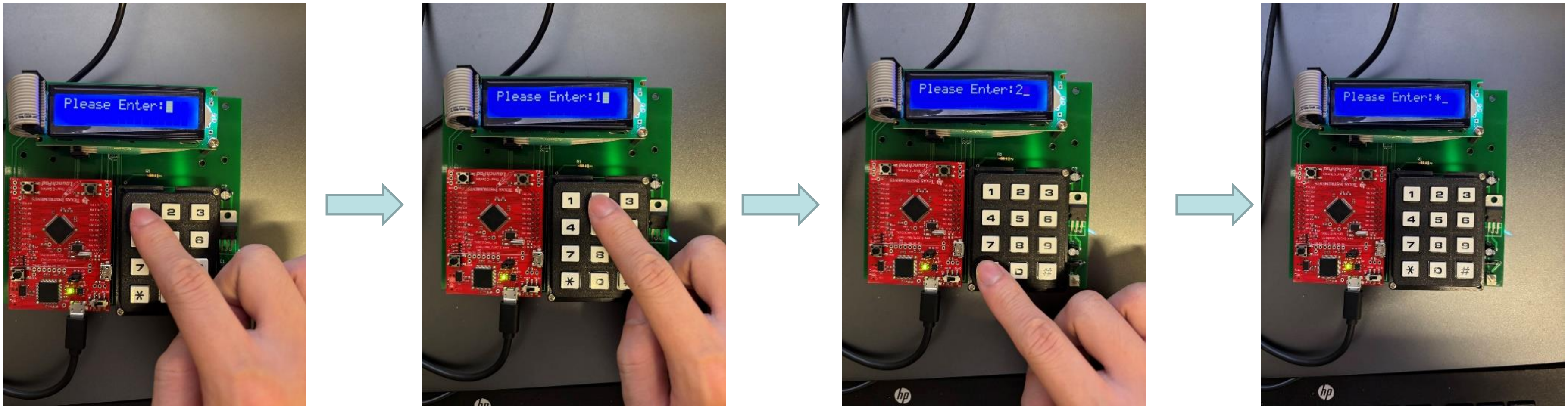
```
#define ROW GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 // Port E
#define COL GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6              // Port C

while (1)
{
    /* check the first row */
    GPIOPinWrite(GPIO_PORTE_BASE, ROW, 0x1C); // ground the first row
    if (!GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_4)) { // check the first column
        LCD_command(1, 0, '1');
        flushInput(GPIO_PORTC_BASE,GPIO_PIN_4);
    } else if (!GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_5)) { // check the second column
        LCD_command(1, 0, '2');
        flushInput(GPIO_PORTC_BASE,GPIO_PIN_5);
    } else if (!GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_6)) { // check the third column
        LCD_command(1, 0, '3');
        flushInput(GPIO_PORTC_BASE, GPIO_PIN_6);
    }
    /* check the second row */
    GPIOPinWrite(GPIO_PORTE_BASE, ROW, 0x1A); // ground the second row
    ...
    /* check the third row */
    GPIOPinWrite(GPIO_PORTE_BASE, ROW, 0x16); // ground the third row
    …
    /* check the fourth row */
    GPIOPinWrite(GPIO_PORTE_BASE, ROW, 0x0E); // ground the fourth row
    ...
}
```



Keypad Connection

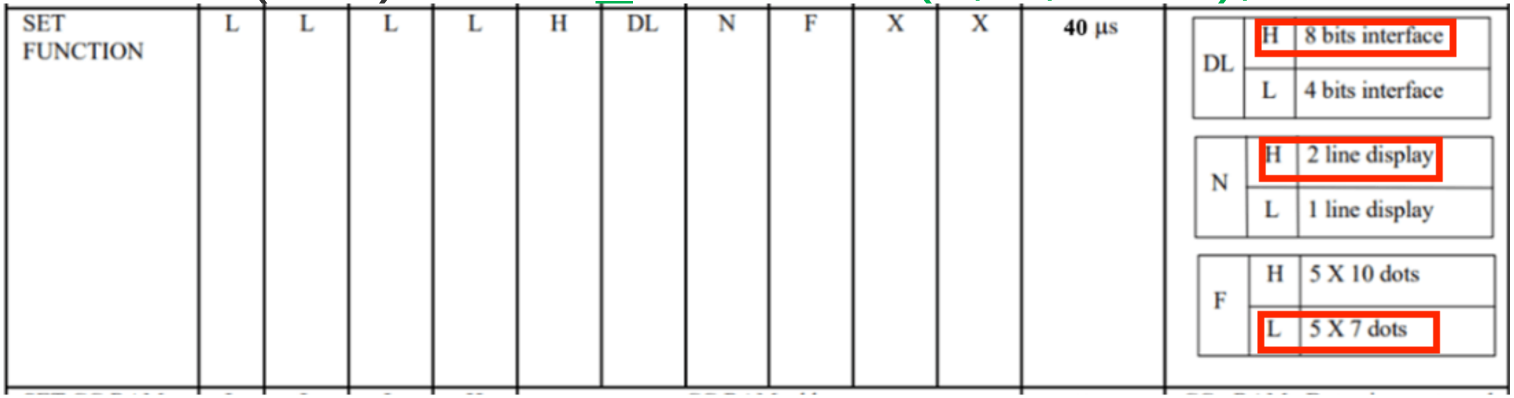
Part 2: Running an example

- Download `lab4_example.c` from blackboard and run.
- Each time you press a button on the keypad, the corresponding character will be displayed after “Please Enter:” on the first line of the LCD screen.
  - Note: The character will be displayed at the same position.



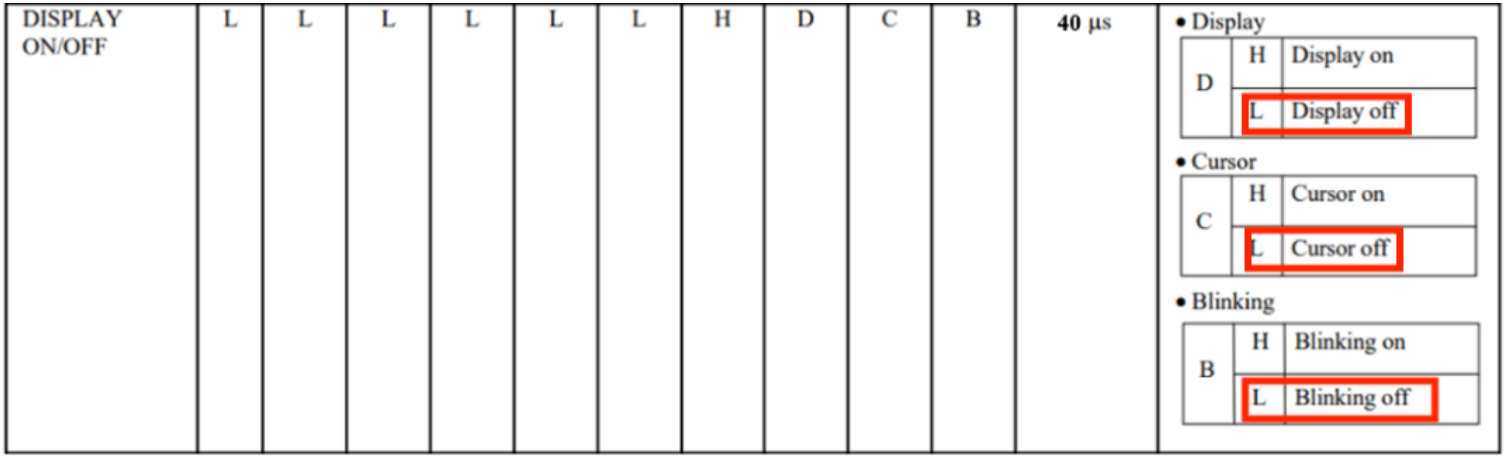
Part 2: Running an example

- LCD Configuration
  - SET FUNCTION: specify 8-bit interface, 2 line display, and 5x7 dots (font) // LCD_command(0, 0, 0x38);



SET FUNCTION	L	L	L	L	H	DL	N	F	X	X	40 µs	DL: H 8 bits interface DL: L 4 bits interface N: H 2 line display N: L 1 line display F: H 5 X 10 dots F: L 5 X 7 dots

  - DISPLAY OFF: set display off // LCD_command(0, 0, 0x08);

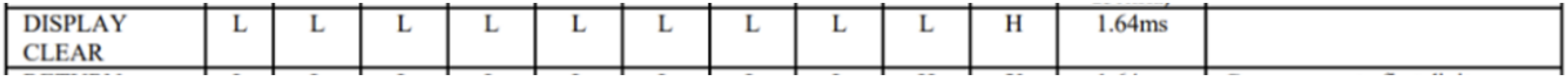


DISPLAY ON/OFF	L	L	L	L	L	L	H	D	C	B	40 µs	• Display D: H Display on D: L Display off • Cursor C: H Cursor on C: L Cursor off • Blinking B: H Blinking on B: L Blinking off

Part 2: Running an example

- LCD Configuration (cont’d)
  - DISPLAY CLEAR: set display clear

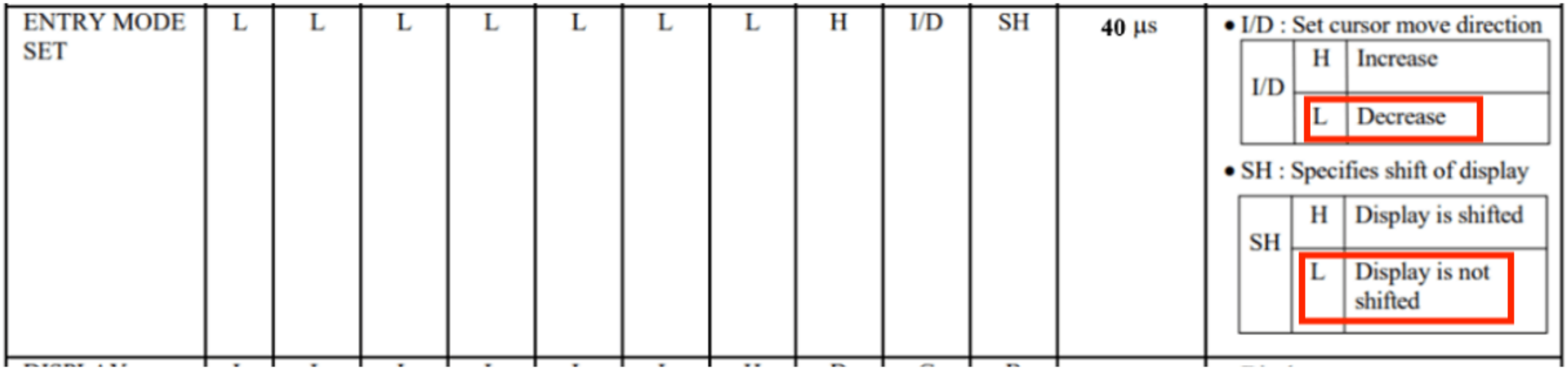
    // LCD_command(0, 0, 0x01);



DISPLAY CLEAR	L	L	L	L	L	L	L	L	L	H	1.64ms	

  - ENTRY MODE SET: set cursor move direction as decreasing & display is not shifted

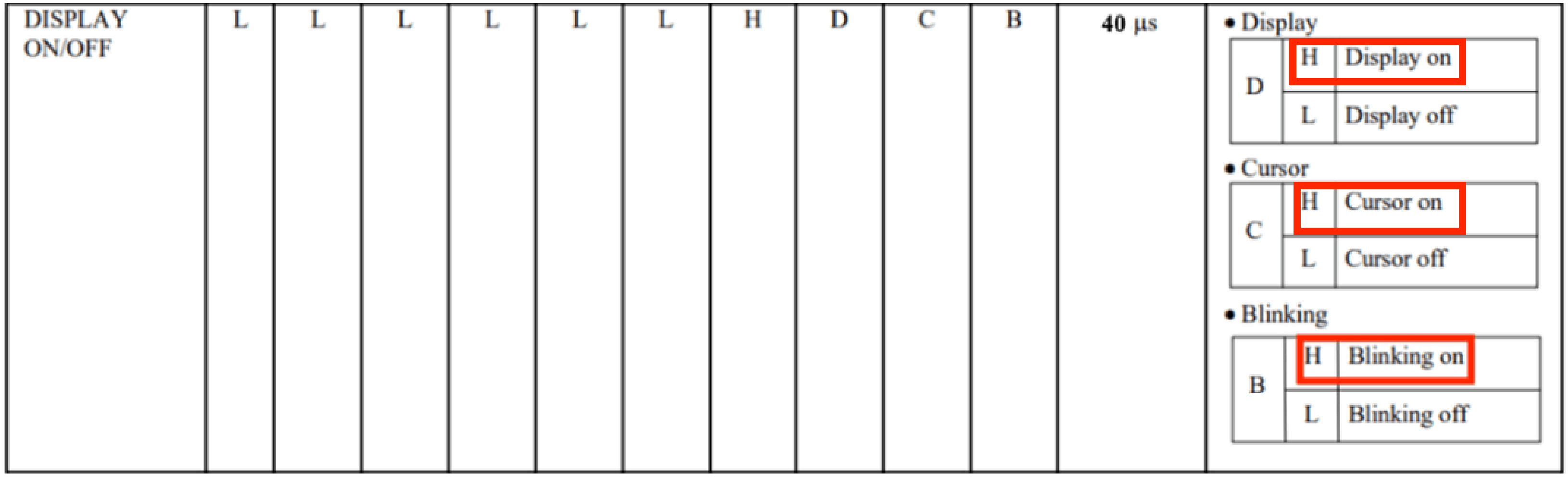
    // LCD_command(0, 0, 0x08);



ENTRY MODE SET	L	L	L	L	L	L	L	H	I/D	SH	40 µs	• I/D : Set cursor move direction I/D H Increase L Decrease • SH : Specifies shift of display SH H Display is shifted L Display is not shifted

Part 2: Running an example

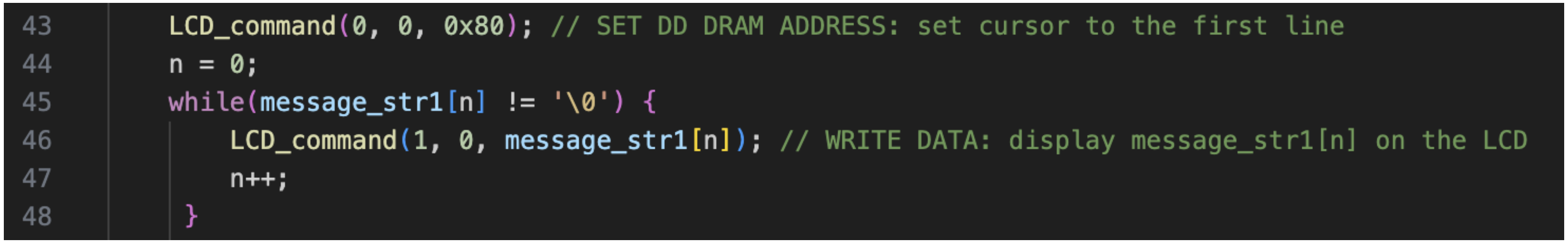
- LCD Configuration (cont’d)
  - DISPLAY ON/OFF: set display on, cursor on, & cursor blinking on // LCD_command(0, 0, 0x08);



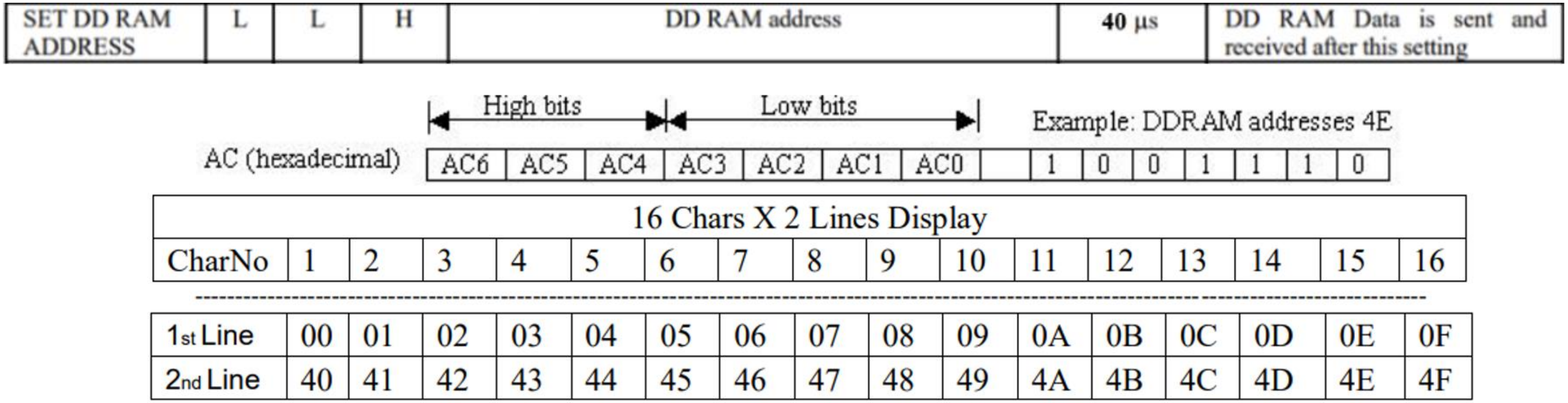
DISPLAY ON/OFF	L	L	L	L	L	L	H	D	C	B	40 µs	• Display D: H Display on D: L Display off • Cursor C: H Cursor on C: L Cursor off • Blinking B: H Blinking on B: L Blinking off

Part 2: Running an example

- LCD Cursor Position
  - In the LCD, one can move the cursor to any location in the display by issuing an address command (0x80)
  - 7th-bit is a flag for this CMD: Line 1 is 0x80; line 2 is 0xC0=0x80+0x40



```
    LCD_command(0, 0, 0x80); // SET DD DRAM ADDRESS: set cursor to the first line
    n = 0;
    while(message_str1[n] != '\0') {
        LCD_command(1, 0, message_str1[n]); // WRITE DATA: display message_str1[n] on the LCD
        n++;
    }
```



SET DD RAM ADDRESS	L	L	H	DD RAM address	40 µs	DD RAM Data is sent and received after this setting

High bits
Low bits

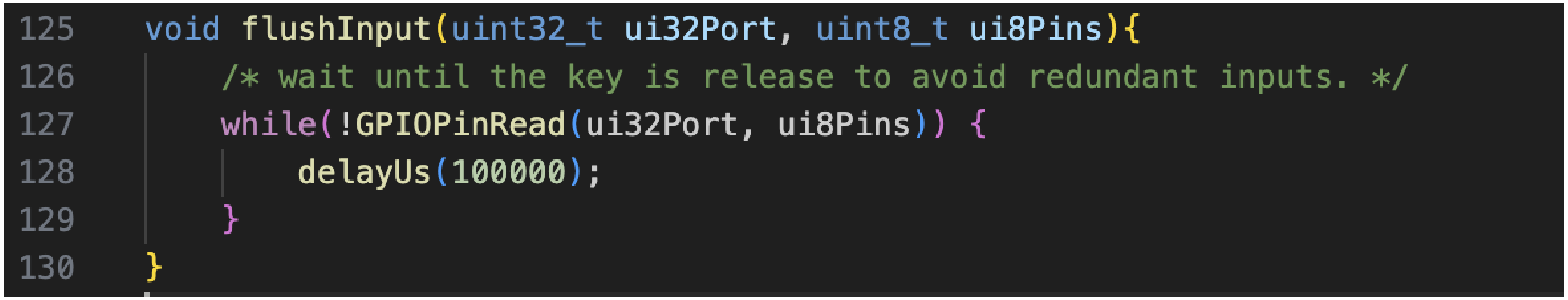
AC (hexadecimal)	AC6	AC5	AC4	AC3	AC2	AC1	AC0
Example: DDRAM addresses 4E	1	0	0	1	1	1	0

16 Chars X 2 Lines Display

CharNo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1st Line	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2nd Line	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Part 2: Running an example

- Prevent Inadvertent Key Presses
  - The purpose of this function is to eliminate key debounce and redundant inputs, ensuring that key events are not repeatedly recorded before the key is released, thereby achieving more stable key input detection.



```c
void flushInput(uint32_t ui32Port, uint8_t ui8Pins){
    /* wait until the key is release to avoid redundant inputs. */
    while(!GPIOPinRead(ui32Port, ui8Pins)) {
        delayUs(100000);
    }
}
```

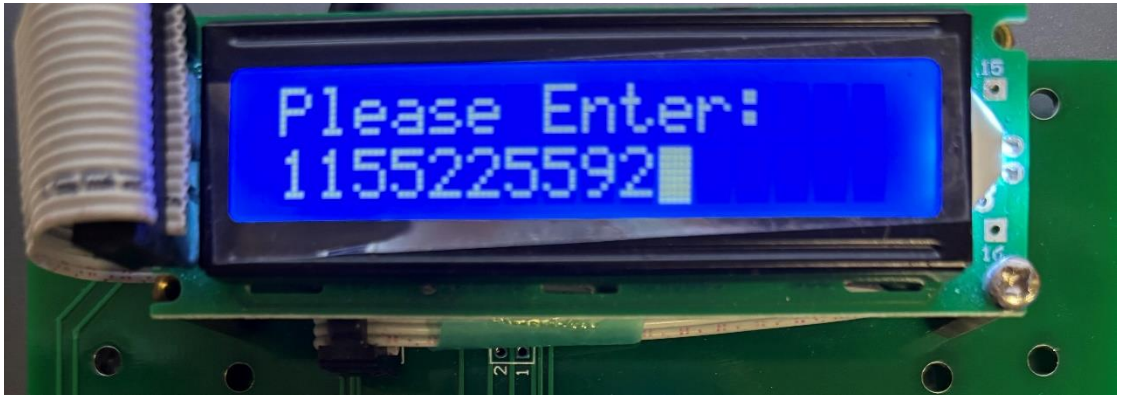
Part 3: Assignment

- Requirements

  ① The first line of the LCD displays 'Please Enter:’.

  ② When you press a **number key ('0'-'9')** on the keypad, the corresponding numbers will be continuously displayed on the second line of the LCD screen.

  - *Hint: Adjust the cursor position to the beginning of the second line by using LCD_command(false, false, ????)*

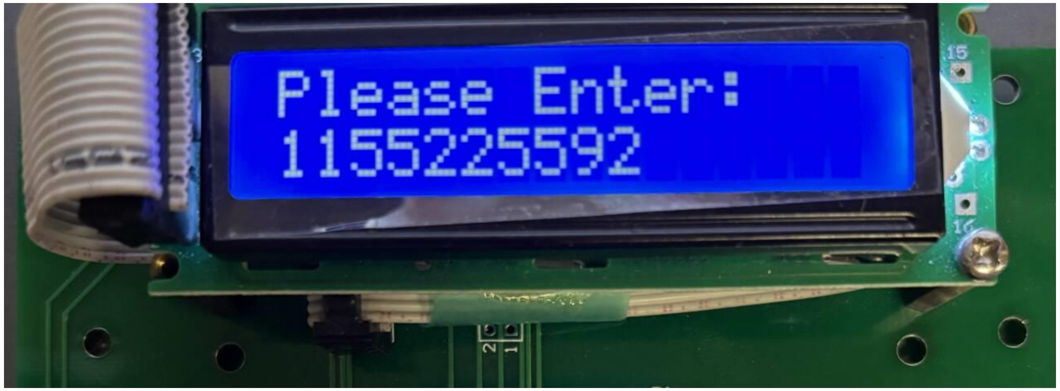


Part 3: Assignment

- Requirements (cont’d)

  ③ When you press the **'#'** key, it indicates that you have confirmed your input, at which point the cursor will **disappear**, and **no further input will be allowed** upon pressing **number keys**.

  - *Hint: Make the **'#'** and **'*'** keys change the value of a variable **flag**.*
    - *When a number key is pressed, first check the value of **flag** to determine whether the key press is valid.*



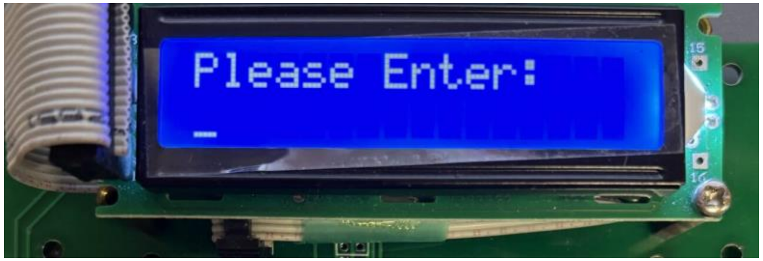
```
Please Enter:
1155225592
```

Part 3: Assignment

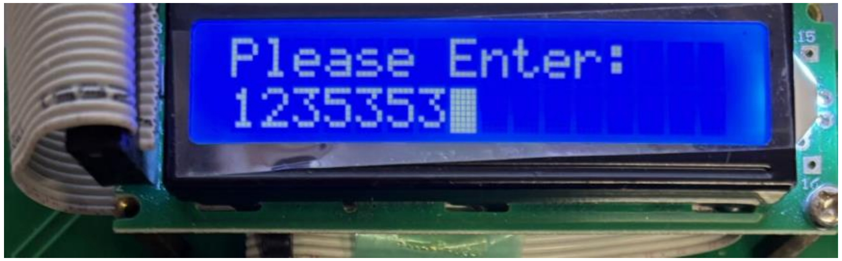
- Requirements (cont’d)

④ Input cannot resume until you press the **'*'** key, which **clears the current output** on the LCD screen, moves the cursor to the **beginning** of the second line, and allows you to **enter numbers again** by pressing the number keys.

- *Hint: Considering the variable **message_str2** in the code, what will happen if the cursor is moved to the **beginning** of the second line and this string is output?*



```
Please Enter:
_
```



```
Please Enter:
1235353
```

Part 3: Assignment

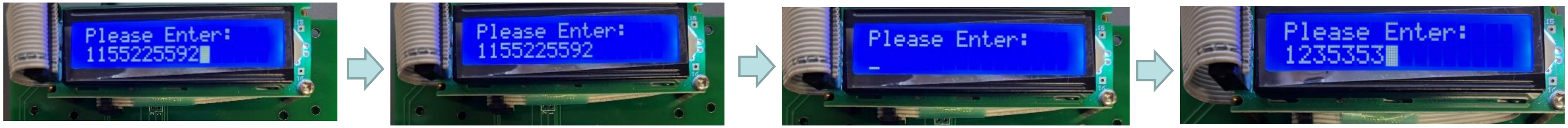
- Requirements of the demo video:

① First, enter your **student ID** on the second line.

② Then, press the **'#'** key, which should cause the cursor to **disappear** while retaining the previously entered student ID. At this point, pressing any number key will not produce any new output on the LCD screen.

③ Next, press the **'*'** key, which will **clear** the current output on the LCD screen, and the cursor will appear at the beginning of the second line.

④ At this point, pressing the number keys will allow the LCD screen to output numbers again (there are no specific requirements for the numbers displayed here).



Q&A

Thanks for listening!

Q & A