實驗名稱：實驗九 LCD 及 DS18B20

實驗目的：瞭解LCD的使用、瞭解DS18B20的使用

實驗步驟、結果與程式流程說明：

9.1 跑馬燈

在 LCD 上顯示自己的組別（兩位數），並且讓它從左到右依序顯示，每間格 0.3 秒到下一個動畫（請使用 Systick_Handler）。

這個實驗的主程式非常簡單，在完成各項初始化之後就只有掛在那邊而已。

```
int main()
{
    GPIO_init();
    SystemClock_Config();
    init_LCD();
    while(1);
}
```

GPIO_init 是設定各個會用到的 pin 腳，SystemClock_Config 是設定每 0.3 秒 reload 一次的 SystemTick。init_LCD 的 code 如下：

```
37 void init_LCD()
38 {
39     // LCD Register
40     WriteToLCD(0x38, 0); // Function Setting
41     WriteToLCD(0x06, 0); // Entering Mode
42     WriteToLCD(0x0C, 0); // Display on
43     WriteToLCD(0x01, 0); // Clear Screen
44     WriteToLCD(0x80, 0); // Move to top left
45 }
```

至於裡面用到的 WriteToLCD 則是如下：

```
16 int WriteToLCD(int input, int isData)
17 {
18     //TODO: Write command to LCD or let LCD di
19     int x;
20     GPIOB->ODR &= ~RW; // set RW 0
21     GPIOB->ODR |= E; // set E 1
22     delay_10ms();
23     if(isData) // check RS should be 0 or 1
24         GPIOB->ODR |= RS;
25     else
26         GPIOB->ODR &= ~RS;
27     GPIOA->ODR &= 0; // clear DB
28     x = input/4; // first 6 bits
29     input %= 4; // last 2 bits
30     GPIOA->ODR |= x<<4; // set DB first 6 bits
31     GPIOA->ODR |= input; // set DB last 2 bits
32     GPIOB->ODR &= ~E; // set E 0
33     delay_10ms();
34     return 0;
35 }
```

在 SystemTick reload 的時候要讓顯示的數字移動則是靠 SysTick_Handler，c0, c1, c2 的初始值分別為-1, 0, 1：

```
72  void SysTick_Handler(void)
73  {
74      if(c1<16) // control '0' ← 當第一個字元在第一行的時候
75          WriteToLCD(0x80+c1, 0); ← 設定要顯示的位置
76      else if(c1<32) ← 當第一個字元在第二行的時候
77          WriteToLCD(0xC0+c1-16, 0); ← 設定要顯示的位置
78      WriteToLCD(0x30, 1); ← 顯示'0'
79      if(c1<31)
80          c1++;
81      else
82          c1 = 0;
83
84      if(c2<16) // control '6' ← 當第二個字元在第一行的時候
85          WriteToLCD(0x80+c2, 0); ← 設定要顯示的位置
86      else if(c2<32) ← 當第二個字元在第二行的時候
87          WriteToLCD(0xC0+c2-16, 0); ← 設定要顯示的位置
88      WriteToLCD(0x36, 1); ← 顯示'6'
89      if(c2<31)
90          c2++;
91      else
92          c2 = 0;
93
94      if(c0<16 && c0>=0) // control ' ' ← 當要消成空白的字元在第一行的時候
95          WriteToLCD(0x80+c0, 0); ← 設定要顯示的位置
96      else if(c0<32) ← 當要消成空白的字元在第二行的時候
97          WriteToLCD(0xC0+c0-16, 0); ← 設定要顯示的位置
98      WriteToLCD(0x20, 1); ← 顯示' '
99      if(c0<31)
00          c0++;
01      else
02          c0 = 0;
03  }
```

← 調整第一個字元的位置

← 調整第二個字元的位置

← 調整要消成空白字元的位置

9.2 客製化圖形顯示與按鈕切換

實作兩種模式，並且透過板子上的按鈕（PC13）在模式之間切換（放開才反應在 LCD 上）

模式一：自製一個兩格的圖像然後讓他跑 3-1 實驗的 pattern（由左到右），並且一樣每 0.3 秒一個動畫（請使用 Systick_Handler）。

模式二：讓 LCD 可以顯示宣告好的字串（助教會改 DEMO 字串），每 0.3 秒顯示一個字元（請使用 Systick_Handler）。

這個實驗的主程式一樣很簡單，只有一直偵測 button 有沒有被按下來決定現在的顯示模式而已。

```
int main()
{
    int f[16] = {0x00, 0x00, 0x10, 0x0F, 0x07, 0x0C, 0x08, 0x04, 0x08, 0x06, 0x07, 0x1C, 0x18, 0x10, 0x10, 0x08};
    GPIO_init();
    SystemClock_Config();
    init_LCD();
    CreateFont(0x0, f);
    while(1)
    {
        if(user_press_button())
            mode = !mode;
    }
}
```

GPIO_init、SystemClock_Config、init_LCD 的 code 和 exp 1 一樣，新增的是用來在 LCD 中存入自定義字元的 CreateFont：

```
void CreateFont(int location, const int *fontArray)
{
    int i;
    WriteToLCD(0x40+location, 0); ← 設定要寫入的 CG RAM 起始位置
    for(i=0;i<16;i++)
    {
        WriteToLCD(fontArray[i], 1); ← 把自訂的字元依序傳進去
    }
}
```

SysTick_Handler 的部分因為要做的事不一樣所以也稍有不同：

```
void SysTick_Handler(void)
{
    if(mode==0) ← 如果是要顯示跑馬燈
    {
        if(prev_mode==1) ← 如果上一個 cycle 是要顯示字串
        {
            display = 0; ← 把字串顯示長度歸零
            WriteToLCD(0x01, 0); // Clear Screen
        }

        if(c1<16) // control f1
            WriteToLCD(0x80+c1, 0);
        else if(c1<32)
            WriteToLCD(0xC0+c1-16, 0);
        WriteToLCD(0x0, 1);
```

← 和 exp1 中顯示第一個字元的原理相同
只是寫入的字元 address 從 DD RAM 變成 CG RAM

```
225             if(c1<31)
226                 c1++;
227             else
228                 c1 = 0;
229
230             if(c2<16) // control f2
231                 WriteToLCD(0x80+c2, 0);
232             else if(c2<32)
233                 WriteToLCD(0xC0+c2-16, 0);
234             WriteToLCD(0x1, 1);
235             if(c2<31)
236                 c2++;
237             else
238                 c2 = 0;
239
240             if(c0<16 && c0>=0) // control '
241                 WriteToLCD(0x80+c0, 0);
242             else if(c0<32)
243                 WriteToLCD(0xC0+c0-16, 0);
244             WriteToLCD(0x20, 1);
245             if(c0<31)
246                 c0++;
247             else
248                 c0 = 0;
249             prev_mode = 0; ← 為下一個 cycle 紀錄現在這個 cycle 的模式
250         }
251         else ← 如果是要顯示字串
252         {
253             if(prev_mode==0) ← 如果上一個 cycle 是要顯示跑馬燈
254             {
255                 c0 = -1;
256                 c1 = 0;   ← 把各字元的位置初始化
257                 c2 = 1;
258                 WriteToLCD(0x01, 0); // Clear Screen
259             }
260             WriteStrToLCD(str); ← 顯示設定的字串
261             if(display<16)
262                 display++;
263             else
264             {
265                 display = 0;
266                 WriteToLCD(0x01, 0); // Clear Screen
267             }
268             prev_mode = 1; ← 為下一個 cycle 紀錄現在這個 cycle 的模式
269         }
270     }
```

← 和 exp1 中顯示第二個字元的原理相同
只是寫入的字元 address 從 DD RAM 變成 CG RAM

← 和 exp1 中顯示要消成空白的字元相同

← 如果顯示長度還沒超過範圍就加一，否則歸零並清除畫面

```
178 void WriteStrToLCD(char *str)
179 {
180     if(str[display]>0x20 && str[display]<0x7E)
181         WriteToLCD(str[display], 1);
182     else
183         WriteToLCD(0x20, 1); ← 否則顯示空白
184 }
```

← 如果字串裡的字元在 DD RAM 裡就顯示

9.3 跑馬燈與溫度計

請承接3.2並且將第二種模式改成顯示當前的溫度，並且讓溫度計擁有0.125的精度。在第一種模式的情況下，動畫仍然以0.3秒的週期往右移動，在第二種模式，則請以1秒為週期刷新溫度讀值，讀值不需要一個一個慢慢顯示，一次顯示完即可。

這個實驗的 main.c 和 exp2 幾乎一模一樣，差別只在 SysTick_Handler 顯示字串的部分變成顯示溫度，所以底下 code 只有貼這個部分。

```
429     else
430     {
431         if(prev_mode==0)
432         {
433             SysTick->LOAD = 9999999;
434             SysTick->VAL = 2999998;
435             c0 = -1;
436             c1 = 0;
437             c2 = 1;
438             WriteToLCD(0x01, 0); // Clear Screen
439         }
440         OneWire_Init(&wire, GPIOC, 0); ← 設定和溫度計傳訊息的腳位
441         DS18B20_SetResolution(&wire, resolution); ← 設定溫度精度
442         DS18B20_ConvT(&wire, resolution); ← 要溫度計讀取溫度並存到 scratch pad
443         while(DS18B20_Done(&wire)!=0); ← 等溫度轉換好
444         DS18B20_Read(&wire, &temp); ← 讀取溫度計的數值
445         WriteToLCD(0x01, 0); // Clear Screen
446         t1 &= temp;
447         t1 >>= 4;          ← 取得整數部份
448         t2 &= temp; ← 取得小數部分
449         t3 += (t2&0x1)*625;
450         t3 += ((t2&0x2)>>1)*1250;
451         t3 += ((t2&0x4)>>2)*2500;
452         t3 += ((t2&0x8)>>3)*5000;   ← 計算小數部分
453         while(mask<=t1)
454             mask *= 10;             ← 計算整數位數
455         mask /= 10;
456         if(t1==0)
457             WriteToLCD(0x30, 1);
458         else
459         {
460             while(mask>0)
461             {                                   ← 顯示整數部分
462                 i = t1/mask;
463                 WriteToLCD(0x30+i, 1);
464                 t1 -= mask*i;
465                 mask /= 10;
466             }
467         }
468         WriteToLCD(0x2E, 1); ← 顯示小數點
469         mask = 1;
470         while(mask<=t3)
471         {
472             mask *= 10;
473             j--;                    ← 計算小數位數與要補幾個零
474         }
475         mask /= 10;
```

```
476             while(j>0)
477             {
478                 WriteToLCD(0x30, 1);  ← 顯示補零
479                 j--;
480             }
481             while(mask>0)
482             {
483                 i = t3/mask;
484                 WriteToLCD(0x30+i, 1);  ← 顯示小數部分
485                 t3 -= mask*i;
486                 mask /= 10;
487             }
488             prev_mode = 1;  ← 為下一個 cycle 紀錄現在這個 cycle 的模式
489         }
490 }
```

除了主程式以外，要寫的還有控制溫度計訊號傳輸的 onewire.c 和 ds18b20.c，首先是 onewire.c 裡的 code 如下：

OneWire_Init 用來設定要用的 GPIO pin 腳是哪一個。

```
 6 /* Init OneWire Struct with GPIO information
12 void OneWire_Init(OneWire_t* OneWireStruct, GPIO_TypeDef* GPIOx, uint32_t GPIO_Pin)
13 {
14     // TODO
15     OneWireStruct->GPIOx = GPIOx;
16     OneWireStruct->GPIO_Pin = GPIO_Pin;  ← set pin number
17     return;
18 }
```

OneWire_Reset 是實作 one wire protocol 裡的 reset，告訴溫度計要傳指令了。

```
20 /* Send reset through OneWireStruct
28 uint8_t OneWire_Reset(OneWire_t* OneWireStruct)
29 {
30     // TODO
31     uint8_t i=1;
32     /* Line low, and wait 500us */
33     OneWireStruct->GPIOx->MODER &= 0x00<<(2*OneWireStruct->GPIO_Pin); // input mode
34     OneWireStruct->GPIOx->BRR |= 0x1<<OneWireStruct->GPIO_Pin; // output low
35     OneWireStruct->GPIOx->MODER |= 0x01<<(2*OneWireStruct->GPIO_Pin); // output mode
36     delay_2_5us(200);
37     /* Release line and wait for 70us */
38     OneWireStruct->GPIOx->MODER &= 0x00<<(2*OneWireStruct->GPIO_Pin); // input mode
39     delay_2_5us(28);
40     /* Check bit value */
41     do
42     {
43         i = OneWireStruct->GPIOx->IDR & 0x1<<OneWireStruct->GPIO_Pin;  ← check if initialize is success
44     }while(i==0);
45     /* Delay for 430 us */
46     delay_2_5us(172);
47     OneWireStruct->GPIOx->MODER &= 0x00<<(2*OneWireStruct->GPIO_Pin); // input mode
48     return i;
49 }
```

OneWire_WriteBit 也是實作 one wire protocol 來寫入 bit 給溫度計。

```
51⊕/* Write 1 bit through OneWireStruct
57⊖void OneWire_WriteBit(OneWire_t* OneWireStruct, uint8_t bit)
58 {
59     // TODO
60     OneWireStruct->GPIOx->MODER &= 0x0<<(2*OneWireStruct->GPIO_Pin); // input mode
61     if (bit) ← if the bit to write is 1
62     {
63         OneWireStruct->GPIOx->BRR |= 0x1<<OneWireStruct->GPIO_Pin; // output low
64         OneWireStruct->GPIOx->MODER |= 0x01<<(2*OneWireStruct->GPIO_Pin); // output mode
65         delay_2_5us(4); // 10
66         OneWireStruct->GPIOx->MODER &= 0x00<<(2*OneWireStruct->GPIO_Pin); // input mode
67         delay_2_5us(28); // 70
68     }
69     else ← if the bit to write is 0
70     {
71         OneWireStruct->GPIOx->BRR |= 0x1<<OneWireStruct->GPIO_Pin; // output low
72         OneWireStruct->GPIOx->MODER |= 0x01<<(2*OneWireStruct->GPIO_Pin); // output mode
73         delay_2_5us(28); // 70
74         OneWireStruct->GPIOx->MODER &= 0x00<<(2*OneWireStruct->GPIO_Pin); // input mode
75         delay_2_5us(4); // 10
76     }
77     OneWireStruct->GPIOx->MODER &= 0x00<<(2*OneWireStruct->GPIO_Pin); // input mode
78     return;
79 }
```

OneWire_ReadBit 同樣是實作 one wire protocol 讀取溫度計傳來的 bit。

```
81⊕/* Read 1 bit through OneWireStruct
86⊖uint8_t OneWire_ReadBit(OneWire_t* OneWireStruct) {
87     // TODO
88     uint8_t bit;
89     OneWireStruct->GPIOx->BRR |= 0x1<<OneWireStruct->GPIO_Pin; // output low
90     OneWireStruct->GPIOx->MODER |= 0x01<<(2*OneWireStruct->GPIO_Pin); // output mode
91     delay_2_5us(1); // 2.5
92     OneWireStruct->GPIOx->MODER &= 0x00<<(2*OneWireStruct->GPIO_Pin); // input mode
93     delay_2_5us(3); // 7.5
94     bit = OneWireStruct->GPIOx->IDR & 0x1<<OneWireStruct->GPIO_Pin; ← get the bit
95     delay_2_5us(20); // 50
96     OneWireStruct->GPIOx->MODER &= 0x00<<(2*OneWireStruct->GPIO_Pin); // input mode
97     return bit;
98 }
```

OneWire_WriteByte 是由要傳輸的 byte 的 LSB 開始連續寫入 8 個 bit 給溫度計。

```
100⊕/* A convenient API to write 1 byte through OneWireStruct
106⊖void OneWire_WriteByte(OneWire_t* OneWireStruct, uint8_t byte)
107 {
108     // TODO
109     uint8_t i = 8;
110     while (i>0)
111     {
112         OneWire_WriteBit(OneWireStruct, byte & 0x1);
113         byte >>= 1;
114         i--;
115     }
116     return;
117 }
```

OneWire_ReadByte 是由 LSB 開始連續讀取 8 個 bit 溫度計回傳的 bit 寫入 byte 裡。

```
119⊕/* A convenient API to read 1 byte through OneWireStruct
124⊖uint8_t OneWire_ReadByte(OneWire_t* OneWireStruct)
125 {
126     // TODO
127     uint8_t i = 0, byte = 0;
128         while(i<8)
129         {
130             byte |= OneWire_ReadBit(OneWireStruct)<<i;
131             i++;
132         }
133         return byte;
134 }
```

OneWire_SkipROM 是不特別指定資料要傳給哪個溫度計，反正線上只有一個。

```
136⊕/* Send ROM Command, Skip ROM, through OneWireStruct
139⊖void OneWire_SkipROM(OneWire_t* OneWireStruct)
140 {
141     // TODO
142     OneWire_WriteByte(OneWireStruct, 0xCC); ← 根據 data sheet，指令碼為 0xCC
143     return;
144 }
```

ds18b20.c 裡的 code 如下：

DS18B20_ConvT 是用來告訴溫度計轉換溫度並把數字存到 scratch pad 裡。

```
 3⊕/* Send ConvT through OneWire with resolution
11⊖int DS18B20_ConvT(OneWire_t* OneWire, DS18B20_Resolution_t resolution)
12 {
13     // TODO
14     OneWire_Reset(OneWire); ← 告訴溫度計要傳指令了
15     OneWire_SkipROM(OneWire); ← 不指定 ROM
16     OneWire_WriteByte(OneWire, 0x44); ← 傳送指令碼
17     return 0;
18 }
```

DS18B20_Read 是用來告訴溫度計回傳 scratch pad 裡的資料。

```
20⊕/* Read temperature from OneWire
28⊖uint8_t DS18B20_Read(OneWire_t* OneWire, int *destination)
29 {
30     // TODO
31     int temp=0;
32     uint8_t t[2];
33     OneWire_Reset(OneWire); ← 告訴溫度計要傳指令了
34     OneWire_SkipROM(OneWire); ← 不指定 ROM
35     OneWire_WriteByte(OneWire, 0xBE); ← 傳送指令碼
36     t[0] = OneWire_ReadByte(OneWire); ← 接收溫度 least significant 的 8 個 bit
37     t[1] = OneWire_ReadByte(OneWire); ← 接收溫度 most significant 的 8 個 bit
38     OneWire_Reset(OneWire); ← 只需要讀前兩個 byte 就夠了所以用 reset 終止溫度計的資料傳輸
39     temp |= t[1];
40     temp <<= 8;
41     temp |= t[0];
42     *destination = temp;
43     return 0;
44 }
```

← 將讀到的兩個 byte 整合並回傳

DS18B20_SetResolution 是用來設定溫度計的精度。

```
48 /* Set resolution of the DS18B20
56 uint8_t DS18B20_SetResolution(OneWire_t* OneWire, DS18B20_Resolution_t resolution)
57 {
58     uint8_t res = 0;
59     res = resolution-9;
60     res <<= 5;              ← 設定要傳給溫度計的 byte 值
61     res |= 0x1F;
62     OneWire_Reset(OneWire); ← 告訴溫度計要傳指令了
63     OneWire_SkipROM(OneWire); ← 不指定 ROM
64     OneWire_WriteByte(OneWire, 0x4E); ← 傳送指令碼
65     OneWire_WriteByte(OneWire, 0x64); ← 設定高溫界線
66     OneWire_WriteByte(OneWire, 0x00); ← 設定低溫界線
67     OneWire_WriteByte(OneWire, res); ← 設定精度
68     return 0;
69 }
```

DS18B20_Read 是用來檢查溫度計是不是完成溫度轉換了。

```
71 /* Check if the temperature conversion is done or not
78 uint8_t DS18B20_Done(OneWire_t* OneWire)
79 {
80     return OneWire_ReadBit(OneWire); ← 讀取一個 bit 來檢查
81 }
```

心得討論與應用聯想：

這次的實驗花的時間比上個實驗還要久，幾乎整個星期的晚上都砸在這上面了。其實知道這次只要跟著 protocol 做，並不是什麼太複雜的東西，但是問題老是出在硬體上。做實驗一的時候 LCD 一直沒有反應，再檢查過不知道多少次 code 應該沒什麼問題之後我還用馬達充當電表跑 debug mode 檢查了每個腳位的電位高低是不是正確，最後是認真看了英文的原始 data sheet 之後才發現是誤會了作業投影片上 V0 要接什麼的意思。而 LCD 可以顯示之後我就很快的解決了實驗一、二，來到實驗三繼續卡關。

實驗三也是檢查了好幾遍 code 找不出問題但溫度計始終沒有動靜。我甚至都先把讀值進來之後要怎麼換算、模式切換等等框架通通寫好了，也手動設數字測試過確認只要溫度計有反應，這個實驗就完成了，可是它依舊只讀得到 0。後來在聽同學說他調整了接線問題就可以正常運作之後我決定再來檢查硬體，抱著姑且試試無視投影片上說 pull-up mode 就可以達到效果的心情接上上拉電阻之後就發現一切居然都不一樣了。

是說這次實驗做完之後，盒子裡沒用過的元件應該就只剩下一條傳輸線而已了，有種成就達成的感覺。one wire protocol 雖然沒反應的時候完全沒辦法檢查是哪裡的問題實在很討厭，但是想想其實是個很聰明的設計啊。

本次實驗最大教訓：data sheet 還是乖乖看原文的比較好，接線一多之後硬體出問題的機率會大幅提高。