گزارش دستورکار شماره 4

فربد فولادی-98243045

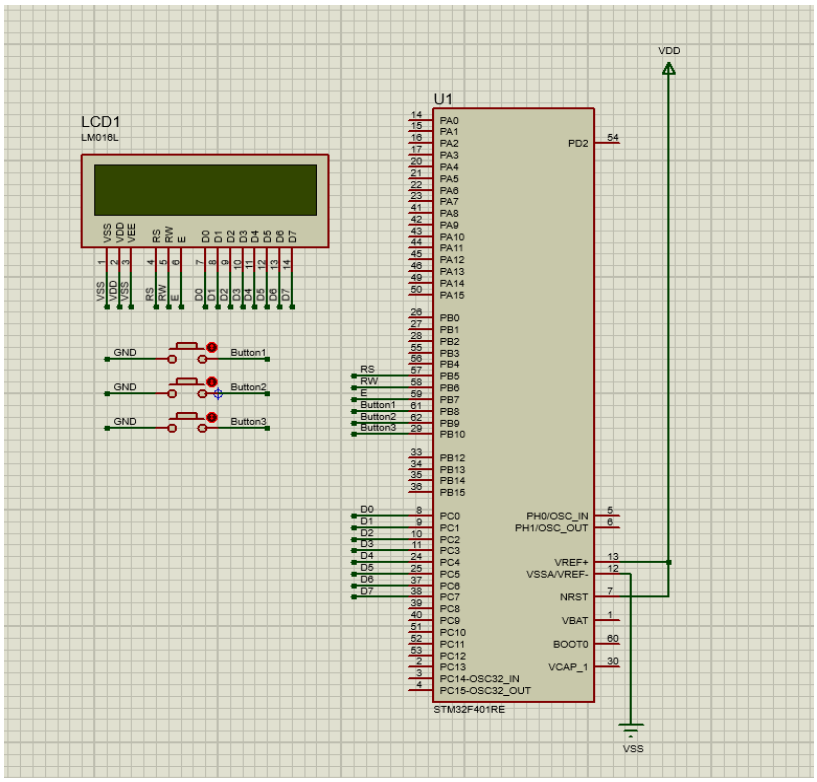
عرفان رفیعی اسکوئی-98243027

در این دستور کار از ما طراحی یک کرنومتر دیجیتال با دقت یک هزارم ثانیه به کمک تایمر در میکروکنترلر STM32F401RE که حاصل باید بر روی یک LCD کاراکتری 2*16 نمایش داده شود خواسته شده بود.

مدل نمایش این شمارنده دیجیتال باید به شکل زیر باشد:

MM:SS:mmm که mmm از 0 تا 999 را باید با فاصله زمانی یک هزارم ثانیه (یک میلی ثانیه) بشمارد و پس از آن به مقدار SS باید 1 واحد اضافه شود. هنگامیکه SS به مقدار 59 رسید یک دقیقه میشود و به MM باید 1 واحد اضافه شود.

در این کرنومتر ما سه کلید داریم که در عکس زیر اورده شده است :



ما در ابتدا RS و RW و EN را define کرده و مقادیر offset مورد نظر را به آنها میدهیم و در ادامه در تابع main بعد از روشن شدن شمارنده، عبارت welcome را چاپ میکنیم و منتظر دریافت input مورد نظر هستیم .

```
static enum state chronometer_state = stopped;

static int MM = 0, SS = 0, MMM = 0, first, second, is_falling = 0;

int main(void)
{
  LCD_init();
  LCD_data('W');
  LCD_data('e');
  LCD_data('l');
  LCD_data('c');
  LCD_data('o');
  LCD_data('m');
  LCD_data('e');
  while (1)
  {
    if (chronometer_state == counting)
    {
      show_time();
      delayMs(75);
    }
    else if (chronometer_state == off)
    {
      break;
    }
  }
```

حال اگر chronometer_state ما به حالت counting درآمد یعنی input ای دریافت شده و از صفحه welcome خارج میشویم و وارد حالت show_time میشویم که تابع آن نیز در انتهای کد تعریف شده است که در ادامه عکس آنرا نیز اورده ایم :

```
229  void show_time(void)
230  {
231    LCD_command(1);
232    char print[16];
233    sprintf(print, "%.2d:%.2d:%.3d", MM, SS, MMM);
234    for (int i = 0; i < 16 && print[i] != '\0'; i++)
235      LCD_data(print[i]);
236  }
237
```

این تابع در هر لحظه زمان مورد نظر ما را در نمایشگر نشان میدهد (به عبارتی چاپ میکند).

در ابتدا RS و RW و EN را با مقادیر اولیه تعریف میکنیم. سپس توابع لازم را تعریف می کنیم که در ادامه توضیح داده ایم.

```
#define RS 0x20 /* PB5 mask for reg select */
#define RW 0x40 /* PB6 mask for read/write */
#define EN 0x80 /* PB7 mask for enable */
```

delayMs(int n برای ایجاد delay است.

```
/* delay n milliseconds (16 MHz CPU clock) */
void delayMs(int n)
{
  int i;
  for (; n > 0; n--)
    for (i = 0; i < 3195; i++)
      __NOP();
}
```

تابع LCD_command یک کامند میگیره و در ابتدا RS و RW را صفر می کنیم و کامند را در ODR قرار می دهیم و EN را high میکنیم. و command های 1 و 2 به 1.64ms delay نیاز دارند و مابقی به 40ms .

```
void LCD_command(unsigned char command)
{
  GPIOB->BSRR = (RS | RW) << 16; /* RS = 0, R/W = 0 */
  GPIOC->ODR &= ~0x00FF;        /* clear data bus */
  GPIOC->ODR |= command;        /* put command on data bus */
  GPIOB->BSRR = EN;         /* pulse E high */
  delayMs(0);
  GPIOB->BSRR = EN << 16; /* clear E */

  if (command < 4)
    delayMs(2); /* command 1 and 2 needs up to 1.64ms */
  else
    delayMs(1); /* all others 40 us */
}
```

در تابع LCD_DATA مقدار rs را 1 و rw را 0 میکنیم و همچنین ODR مان را clear میکنیم تا data ورودی را در آن قرار دهیم و EN را high می کنیم و در آخر با شیفت دادن به چپ به اندازه 16 بیت clear میکنیم.

```
void LCD_data(char data)
{
  GPIOB->BSRR = RS;   /* RS = 1 */
  GPIOB->BSRR = RW << 16; /* R/W = 0 */
  GPIOC->ODR &= ~0x00FF;  /* clear data bus */
  GPIOC->ODR |= data;   /* put data on data bus */
  GPIOB->BSRR = EN;   /* pulse E high */
  GPIOB->BSRR = EN;   /* pulse E high */
  delayMs(0);
  GPIOB->BSRR = EN << 16; /* clear E */
  delayMs(1);
}
```

در تابع LCD_INIT صرفا initialize های اولیه را انجام میدهیم با command هایی که در دستور کار به ما داده شده بود و یک تابع port_init هم صدا زده میشود که پورت های بورد را متناسب با توضیحاتی که در دستور کار بود به دکمه ها و lcd متصل میکند.

```
/* initialize port pins then initialize LCD controller */
void LCD_init(void)
{
  PORTS_init();

  delayMs(30); /* initialization sequence */
  LCD_command(0x30);
  delayMs(10);
  LCD_command(0x30);
  delayMs(1);
  LCD_command(0x30);

  LCD_command(0x38); /* set 8-bit data, 2-line, 5x7 font */
  LCD_command(0x06); /* move cursor right after each char */
  LCD_command(0x01); /* clear screen, move cursor to home */
  LCD_command(0x0F); /* turn on display, cursor blinking */
}
```

در PORTS_INIT علاوه بر enable کردن clock و set کردن pin ها (برای LCD از port های B5, B6, B7 استفاده میکنیم که به ترتیب برای R\S و R\W و EN هستند و برای D0-D7 در LCD از پورت های C0-C7 استفاده میکنیم.)

همچنین TIM5 و TIM2 نیز داریم که به ترتیب برای 3 ثانیه تاخیر دکمه سوم و تاخیر ثانیه شمار به کار میروند.

در آخر هم EXTI9 و EXTI15 زا enable میکنیم و handler آن ها را نیز صدا میزنیم.

```
void PORTS_init(void)
{
  RCC->AHB1ENR |= 0x06; /* enable GPIO B/C clock */
  RCC->APB2ENR |= 0x4000; /* enable SysConfig clock */

  /* PB5 for LCD R/S */
  /* PB6 for LCD R/W */
  /* PB7 for LCD EN */
  GPIOB->MODER = 0x00005400; /* set pin output mode */
  GPIOB->PUPDR = 0x0150000;  /* set pin output mode */
  GPIOB->BSRR = 0x00C00000;  /* turn off EN and R/W */

  /* PC0-PC7 for LCD D0-D7, respectively. */
  GPIOC->MODER = 0x00005555; /* set pin output mode */

  RCC->APB1ENR |= RCC_APB1ENR_TIM5EN; /* enable TIM5 clock */
  TIM5->PSC = 16000 - 1;        /* divided by 16000 */
  TIM5->ARR = 10000 - 1;        /* divided by 10000 */
  TIM5->CR1 = TIM_CR1_CEN;      /* enable counter */

  RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
  TIM2->PSC = 160;
  TIM2->ARR = 10 - 1;
  TIM2->DIER |= TIM_DIER_UIE;
  NVIC_EnableIRQ(TIM2_IRQn); /* enable interrupt in NVIC */

  SYSCFG->EXTICR[2] = 0x0111;
  EXTI->IMR = 0x0700;
  EXTI->FTSR = 0x0700;
  EXTI->RTSR = 0x0400;
  NVIC_EnableIRQ(EXTI9_5_IRQn);
  NVIC_EnableIRQ(EXTI15_10_IRQn);
  __enable_irq();
}
```

تابع EXTI15_10_IRQHandler برای بررسی این است که دکمه سوم به مدت 3 ثانیه نگه داشته شده است یا خیر که متناسب با آن عملیات مربوطه را انجام دهیم.

به این صورت کار میکند که وقتی یک بار فشار داده میشود ابتدا counting را disable میکنیم و وارد استیت stop میشویم و همه مقادیر صفر می شوند و با کمک تابع show_time مقادیری که صفر شده اند را نمایش می دهیم. حال چک میکنیم که آیا مقدار is_falling برابر با 0 است یا 1. اگر 0 باشد یعنی فشار داده شده است پس TIM-> CNT را در first قرار می دهیم و مقدار is_falling را 1 میکنیم و اگر 1 باشد یعنی فشار داده نشده است پس ابتدا مقداش را 0 میکنیم و TIM5-> CNTرا در second قرار می دهیم و حک میکنیم که بیشتر از 30 است یا نه که بیانگر این است که 3 ثانیه نگه داشته شده است که در این صورت وارو استیت off می شویم که در main گفته ایم که اگر استیت برابر با counting است show_time را انجام دهد و اگر برابر با off بود از حلقه break کند و وارد حلقه دیگری شود که عبارت turn_off را روی LCD نمایش می دهد.

```
void EXTI15_10_IRQHandler(void)
{
  chronometer_state = stopped;
  TIM2->CR1 &= ~TIM_CR1_CEN;
  MMM = 0;
  SS = 0;
  MM = 0;
  show_time();
  if (is_falling == 0)
  {
    first = TIM5->CNT;
    is_falling = 1;
  }
  else
  {
    is_falling = 0;
    second = TIM5->CNT;
    if (second - first > 300)
    {
      __disable_irq();
      LCD_command(1);
      chronometer_state = off;
    }
  }

  EXTI->PR |= 0x0400;
  NVIC_ClearPendingIRQ(EXTI15_10_IRQn);
}
```

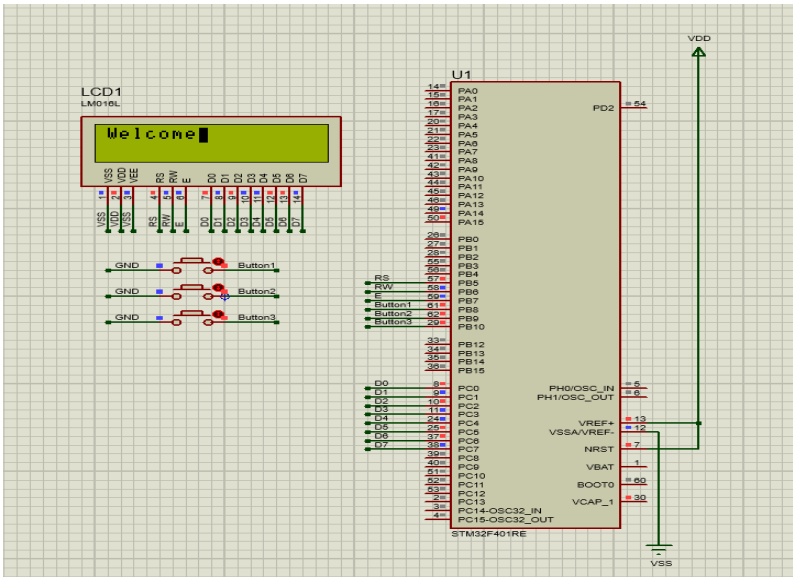
تابع EXTI9_5_IRQHandler نیز برای این است که تشخیص دهیم که دکم اول فشار داده شده است یا دکمه دوم که متناسب با هر کدام عملیات مربوطه آنها را انجام دهیم اگر دکمه اول فشرده شد با TIM2->CR1 |= TIM_CR1_CEN کانتر ما شمارشش فعال شده و اگر دکمه دوم فشرده شود با TIM2->CR1 &= ~TIM_CR1_CEN کانتر ما شمارشش متوقف میشود.

```
void EXTI9_5_IRQHandler(void)
{
  if (EXTI->PR & 0x0100)
  {
    chronometer_state = counting;
    TIM2->CR1 |= TIM_CR1_CEN;
    EXTI->PR |= 0x0100;
  }
  else if (EXTI->PR & 0x0200)
  {
    chronometer_state = paused;
    TIM2->CR1 &= ~TIM_CR1_CEN;
    show_time();
    EXTI->PR |= 0x0200;
  }
  NVIC_ClearPendingIRQ(EXTI9_5_IRQn);
}
```

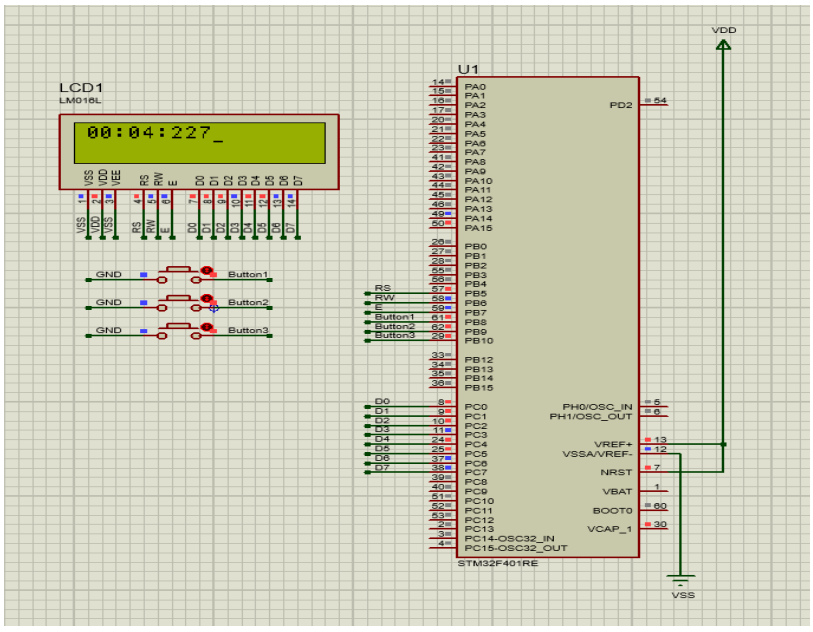
تابع TIM2_IRQHandler بررسی میکند که اگر در استیت counting بودیم MMM را یکی زیاد میکند و چک میکند که اگر بیشتر از 999 شد یکی به SS زیاد کند و باز چک میکند که اگر SS بیشتر از 59 شد یکی به MM زیاد کند و SS را برابر با 0 کند و دوباره چک میکند که اگر MM بیشتر از 99 شد مقدارش را 0 کند و این روند تا وقتی که در استیت counting هستیم ادامه دارد.

در ادامه تعدادی عکس از نتیجه نهایی شبیه سازی در proteus را آورده ایم:

1- شروع اولیه :



2- بعد از فشرده شدن دکمه اول :



3- بعد از فشردن دکمه سوم به مدت 3 ثانیه :