گزارش آزمایش 6

نگار هنرور صدیقیان – 99243076

امین احسانی‌مهر- 99243009

سوالات تحلیلی- بخشA

1- Input capture: در زمان وقوع یک رویداد مانند rising or falling edge تایمر با دریافت مقدار شمارنده زمان بین دو رویداد را محاسبه میکند.

Output compare: تایمر یک مقدار از پیش تعیین شده دارد که به طور مرتب مقدار شمارنده خود را با آن مقدار مقایسه میکند ، در صورت تساوی این دو مقدار یک رویداد خارجی مانند ایجاد پالس یا شروع وقفه میتواند ایجاد شود.

PWM generation: در این حالت تایمر یک موج مربعی با فرکانسی ثابت و دوره متغییر تولید میکند که با مقداردهی های متفاوت به این دوره میتوان توان مصرفی یک قطعه را کنترل کرد. از آن برای تولید سیگنال با عرض پالس های مختلف استفاده میشود.

One pulse mode output: با فراخوانی این حالت ویژه تایمر یک پالس با عرض مشخص تولید کرده و سپس متوقف میشود، به این ترتیب در این حالت امکان تولید رویداد با زمان دقیق فراهم بوده و برای تولید پالس با عرض یا دوره زمانی مشخص کاربرد دارد.

2- Auto reload register: به کمک این تایمر و با ست کردن یک مقدار که به آن شمارنده ترمینال تایمر می‌گویند میتوان مدت زمان موردنظر را برای تایمر ست کرد. هنگامی که شمارنده به این مقدار رسید یک وقفه یا رویداد مورد نظر میتواند راه اندازی شود و در ادامه مقدار رجیستر به صفر یا تعداد ترمینال تایمر ست میشود.

Pre-Scalar register: با تنظیم یک مقدار یا فاکتور خاص این رجیستر کلاک ورودی را پیش از اعمال آن به تایمر بر این مقدار تقسیم میکند و به ما این امکان را میدهد که فرکانس ساعت را کاهش دهیم و وضوح تایمر را تغییر دهیم.

Counter register: این شمارنده مقدار فعلی تایمر را بررسی میکند و با هرچرخه کلاک افزایش میابد. از آن برای اندازه‌گیری زمان سپری شده استفاده می‌شود .

گزارش کد-بخش A

سوال اول

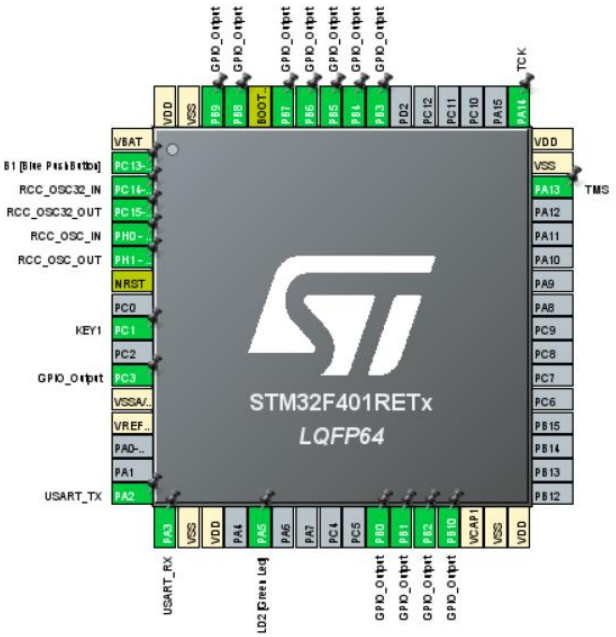
در این سوال به دنبال پیاده سازی یک تایمر هستیم که پس از فشردن کلید شروع به شمردن کند و پس از آن در صورت فشردن و رها کردن سریع کلید به طوری که کمتر از 0.5 ثانیه باشد تایمر توقف کرده و با فشردن مجدد دوباره ادامه دهد و اگر کلید را بیش از 0.5 ثانیه نگه داشتیم تایمر ری ست شود.

ابتدا یک پروژه جدید در STMCUBE ایجاد کرده و تنظیمات زیر را اعمال می کنیم:

1- پین های خروجی lcd را بصورت output و پین کلید را EXTI تعریف می کنیم. پین آن (PC1) را pull down می کنیم و وقفه را در هر دو لبه falling و rising ست می کنیم.

2- تایمر های 2 و 3 را فعال کرده و تنظیم می کنیم و وقفه آنها را روشن میکنیم،تایمر ۳ هر ۰.۱ ثانیه و تایمر ۲ هر ۰.۵ ثانیه وقفه می دهد.

در نهایت کد را generate میکنیم.



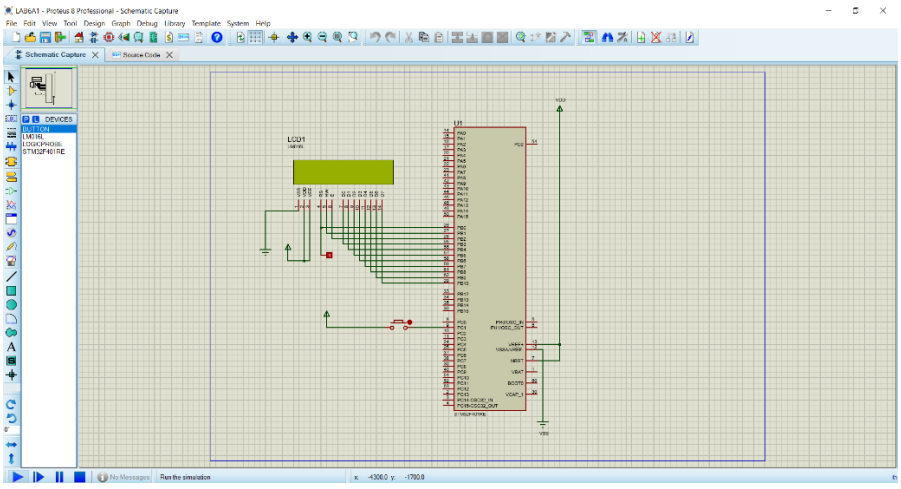
در مرحله بعد کد تولیدی را در کیل بررسی کرده و تغییر میدهیم. این کد شامل دو بخش است. بخشی از آن مشابه با آنچه درلب 5 دیدیم برای پیاده سازی و راه اندازی ال سی دی بوده و بخش دیگر مربوط به تایمرها و بررسی این است که کلید چه میزان زمانی فشرده شده و در صورتی تطابق با 3 توقف و تطابق با تایمر 2 ری ست انجام میشود.

```
77  }
78  void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
79      if(htim->Instance == TIM3){
80          counter++;
81          display();
82      }else if(htim->Instance == TIM2){
83          reset = 1;
84      }
85  }
86  void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
87      if(GPIO_Pin == KEY1_Pin){
88          if(isPressed == 0){
89                  HAL_TIM_Base_Start_IT(&htim2);
90                  isPressed = 1;
91          }else{
92                  isPressed = 0;
93                  if(reset == 1){
94                      counter = 0;
95                      reset = 0;
96                      counterIsOn = 0;
97                      HAL_TIM_Base_Stop(&htim3);
98                      HAL_TIM_Base_Stop(&htim2);
99                      display();
100
101                 }else{
102                     HAL_TIM_Base_Stop(&htim2);
103                     reset = 0;
104
105                     if(counterIsOn == 1){
106                         counterIsOn = 0;
107                         HAL_TIM_Base_Stop(&htim3);
108                     }else{
109                         counterIsOn = 1;
110                         HAL_TIM_Base_Start_IT(&htim3);
111                     }
112                 }
113         }
114     }
```

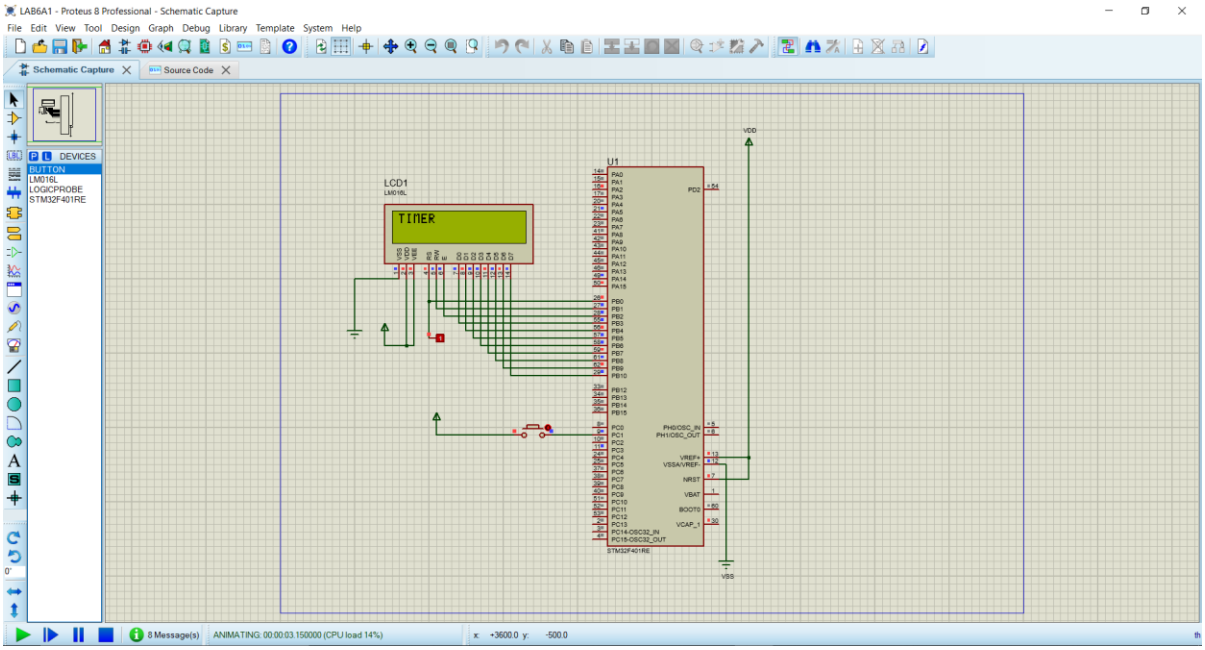
در ادامه از کد بیلد گرفته و فایل hex تولیدی را در پروتئوس باز کرده و شماتیک آن را تولید میکنیم:

(در هر مرحله نتیجه قابل مشاهده است)

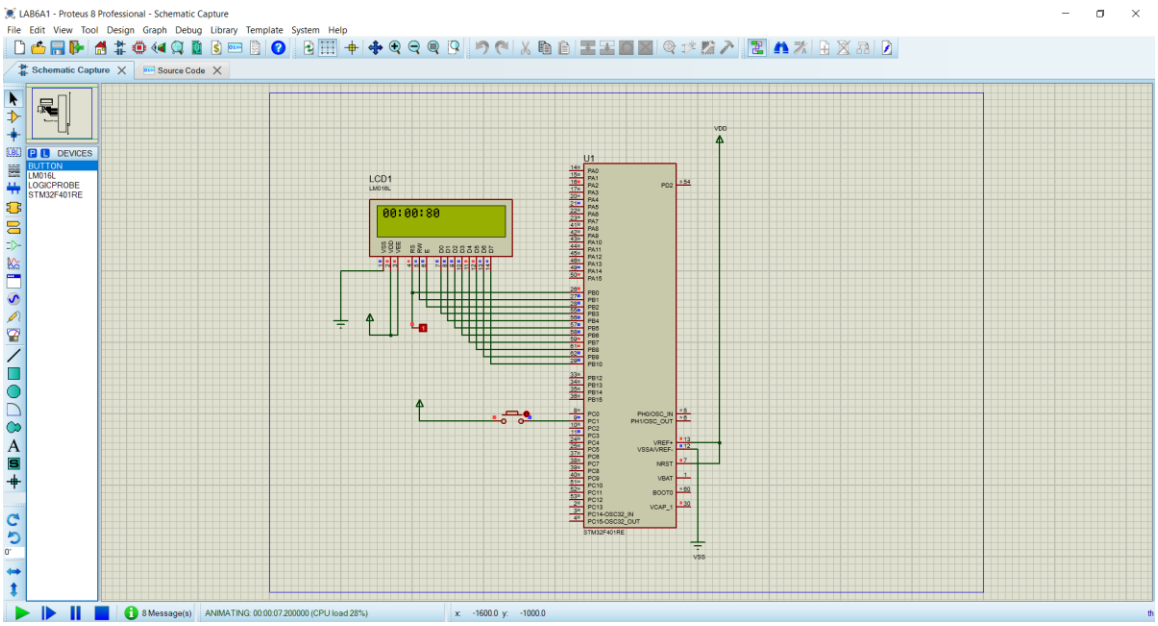
1-پیش از ران کردن



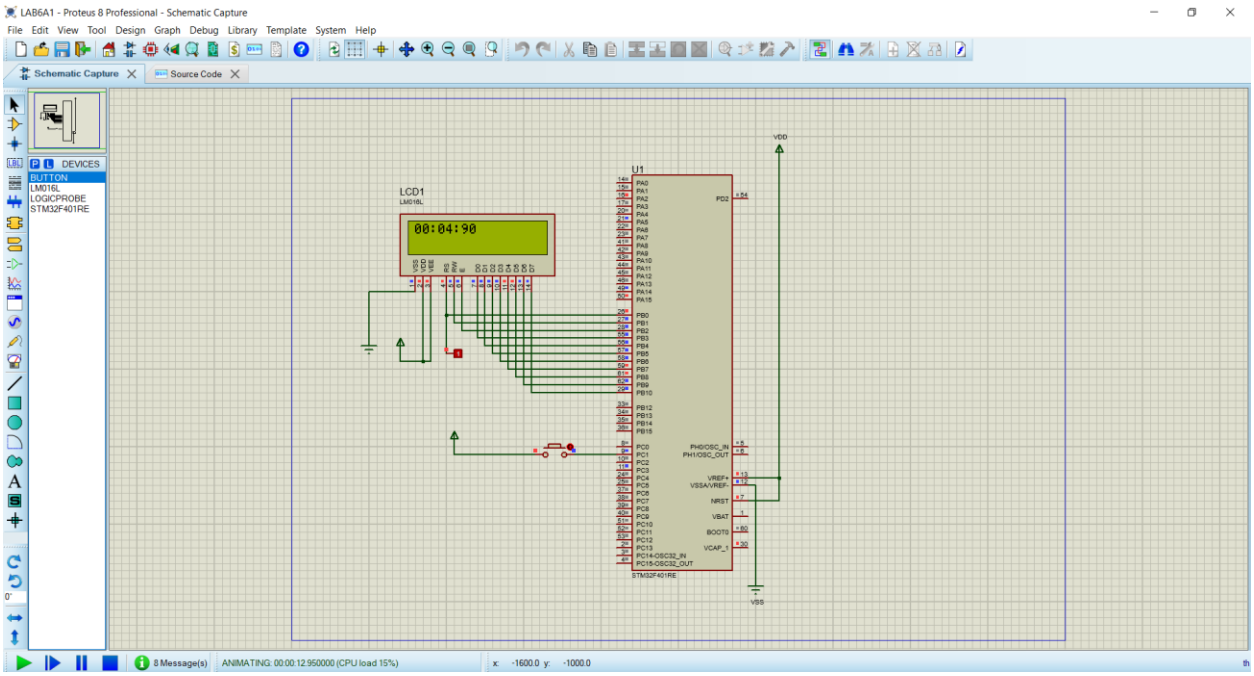
2- پس از ران کردن



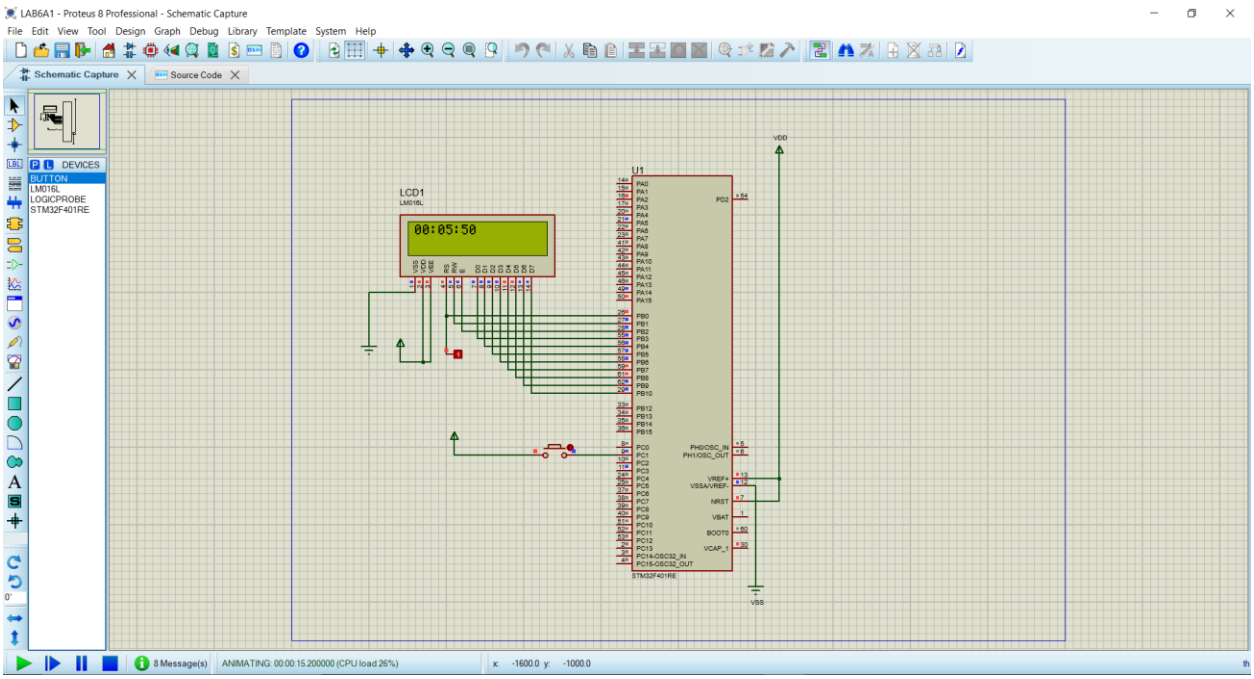
3- پس از اولین فشردن کلید



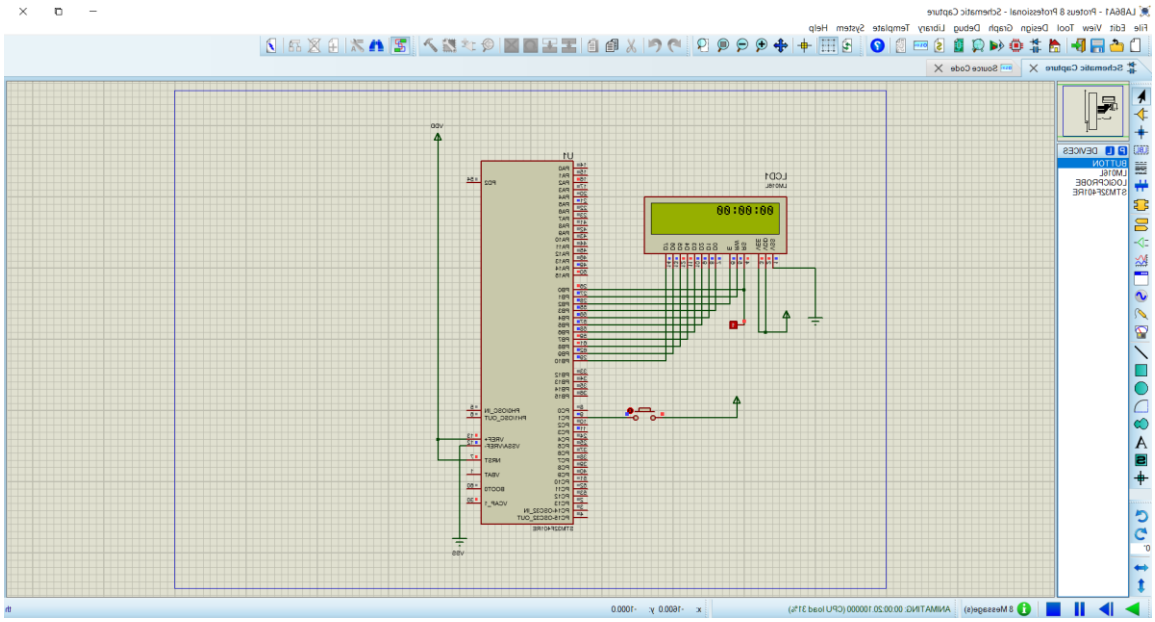
4- پس از فشردن کوتاه کلید برای ایجاد توقف



5- پس از فشردن مجدد کلید و ادامه



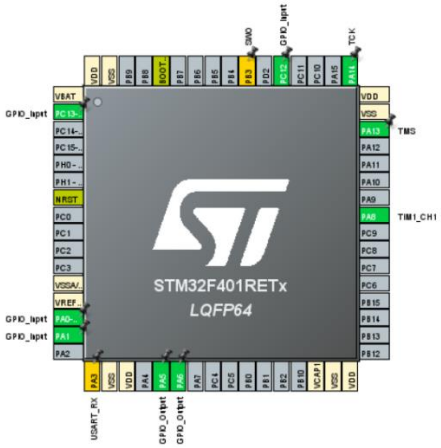
6- پس از فشردن طولانی کلید برای ریست کردن



سوال اول

در این سوال به دنبال پیاده سازی سیستمی با LED که توسط سیگنال PWM هستیم که با کنترل duty cycle و با توجه به تغییری که در چرخه duty cycle ایجاد میکنیم میزان روشنایی LED نیز تغییر میکند. به کمک یک دکمه به نام inc_brightness و با هربار فشردن آن چرخه ده درصد افزایش یافته و روشنایی LED ده درصد افزایش میابد، به طور مشابه با هربار فشردن دکمه dec_brightness روشنایی LED و چرخه ده درصد کاهش میابد.

در مرحله اول پین های مورد نظر را در STM cubemx کانفیگ میکنیم و کد آن را generate کرده و در keil پروژه ای جدید میسازیم.



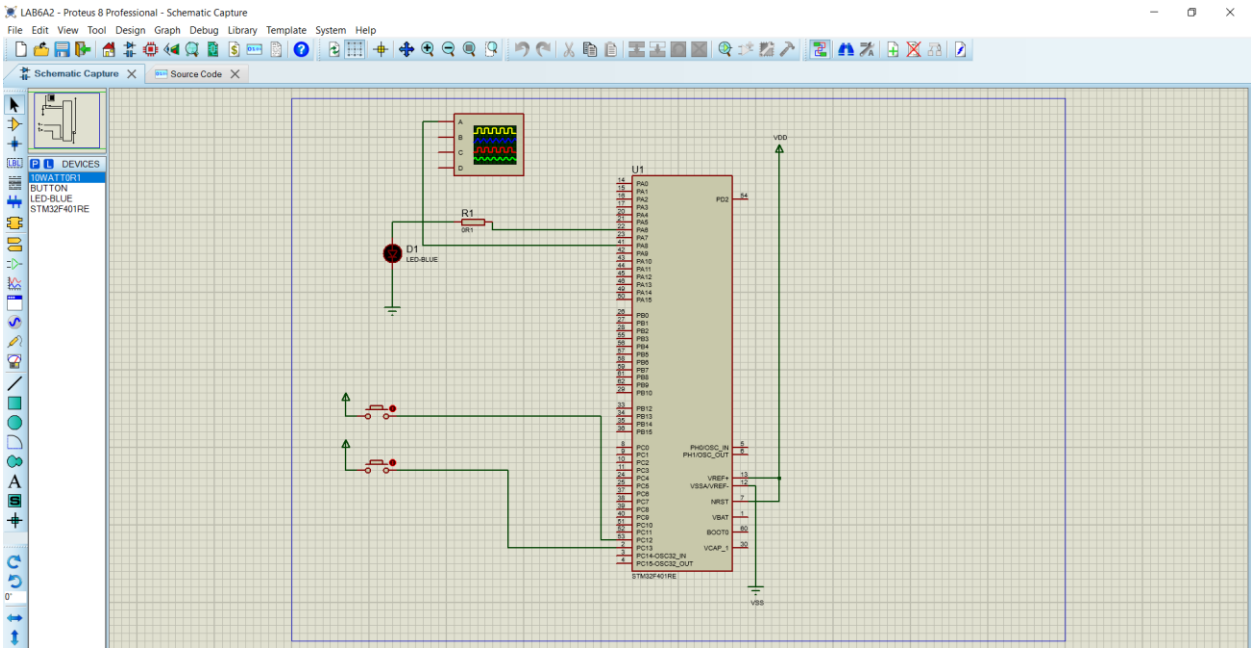
جزئیات کانفیگ شدن و پین هایی که کانفیگ شده اند مطابق بخش زیر از کد است:

```
128  static void MX_GPIO_Init(void)
129  {
130    GPIO_InitTypeDef GPIO_InitStruct = {0};
131
132    /* GPIO Ports Clock Enable */
133    __HAL_RCC_GPIOC_CLK_ENABLE();
134    __HAL_RCC_GPIOA_CLK_ENABLE();
135    __HAL_RCC_GPIOB_CLK_ENABLE();
136
137    /*Configure GPIO pin Output Level */
138    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5|GPIO_PIN_6, GPIO_PIN_RESET);
139
140    /*Configure GPIO pins : PC13 PC12 */
141    GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_12;
142    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
143    GPIO_InitStruct.Pull = GPIO_PULLDOWN;
144    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
145
146    /*Configure GPIO pins : PA0 PA1 */
147    GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1;
148    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
149    GPIO_InitStruct.Pull = GPIO_NOPULL;
150    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
151
152    /*Configure GPIO pin : USART_RX_Pin */
153    GPIO_InitStruct.Pin = USART_RX_Pin;
154    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
155    GPIO_InitStruct.Pull = GPIO_NOPULL;
156    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
157    GPIO_InitStruct.Alternate = GPIO_AF7_USART2;
158    HAL_GPIO_Init(USART_RX_GPIO_Port, &GPIO_InitStruct);
159
160    /*Configure GPIO pins : PA5 PA6 */
161    GPIO_InitStruct.Pin = GPIO_PIN_5|GPIO_PIN_6;
162    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
163    GPIO_InitStruct.Pull = GPIO_NOPULL;
164    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
165    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

در ادامه و در بخش آغازین کد یک متغیر به نام dc_value تعریف میکنیم که دوره چرخه duty cycle برای سیگنال pwm را مشخص میکند و بسته به اینکه کدام کلید فشرده شود مقدار 500 واحد به آن افزوده یا از آن کم میشود.از آن جایی که لازم است این تغییر در شماتیک قابل مشاهده باشد یک اسیلوسکوپ کانفیگ میکنیم که تغییر در میزان روشنایی LED را با توجه به تغییر دوره سیگنال میتوان متوجه شد. پیاده سازی های مربوط به اسیلوسکوپ مطابق قطعه کد صفحه بعد است:

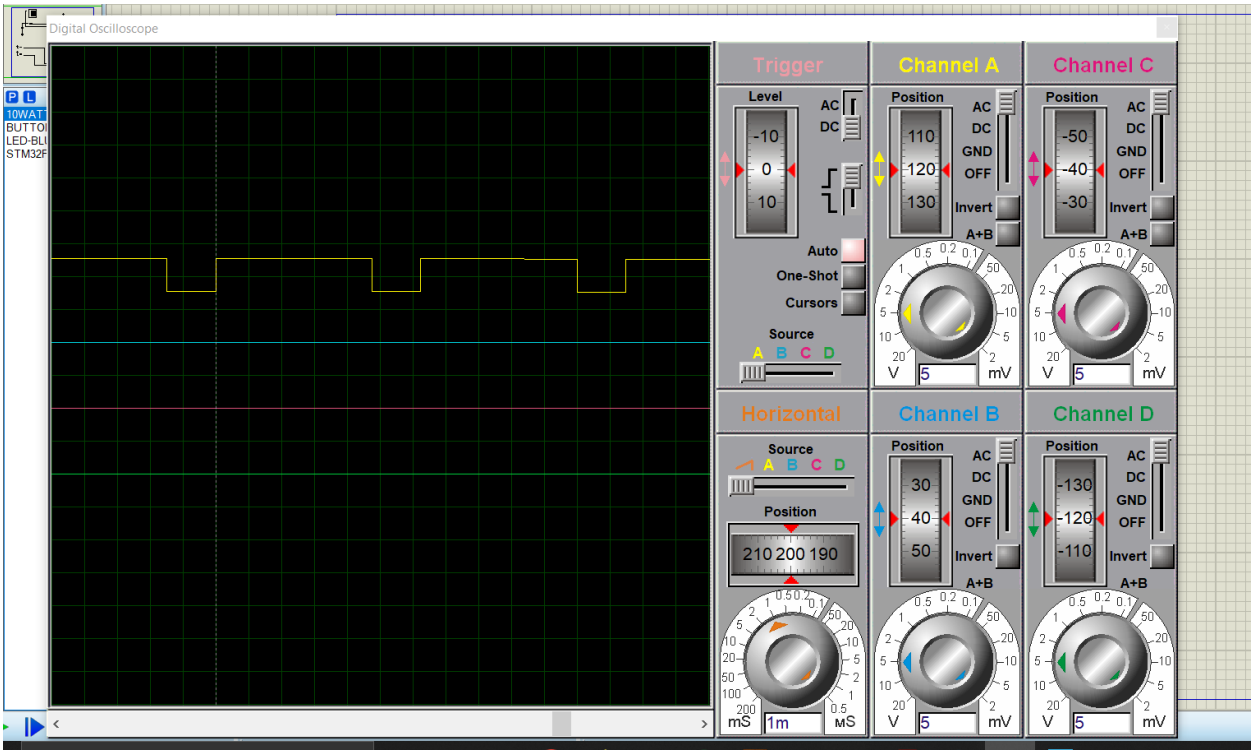
```c
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}
```

در ادامه از کد فوق build گرفته و فایل hex تولیدی را به پروتئوس منتقل کرده و شماتیک مربوطه را میسازیم که مطابق شکل زیر است:

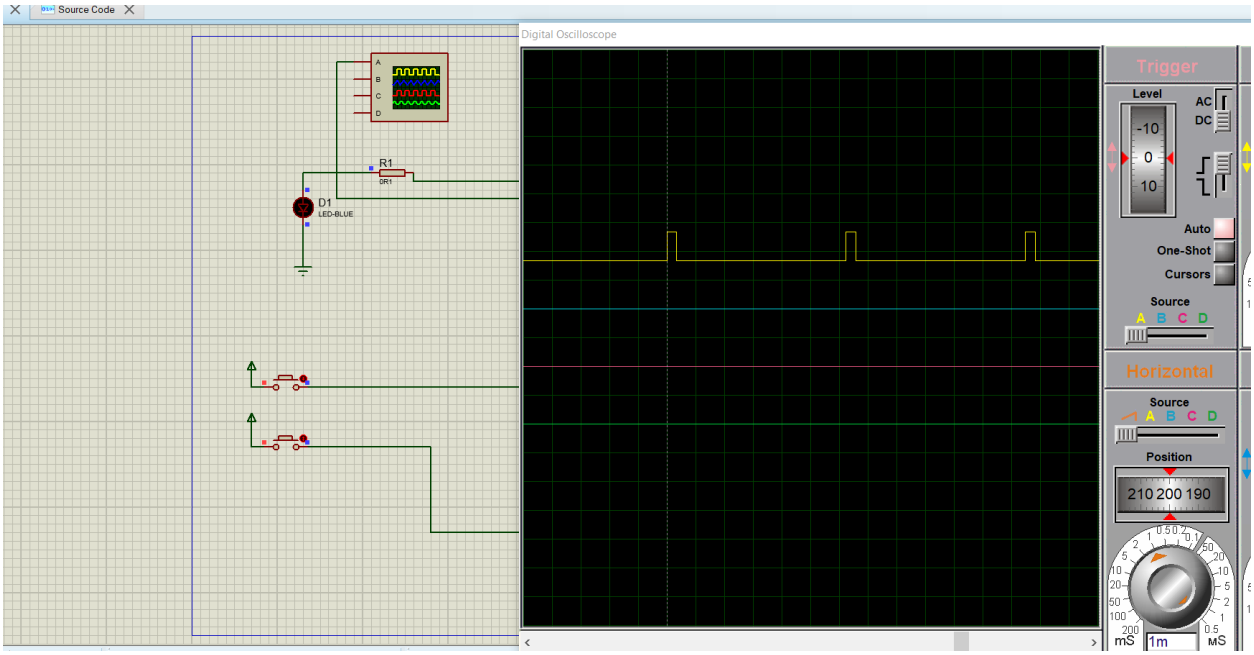


سپس این شماتیک را اجرا کرده و با فشردن کلید تغییر در میزان روشنایی را با اسیلوسکوپ بررسی میکنیم.

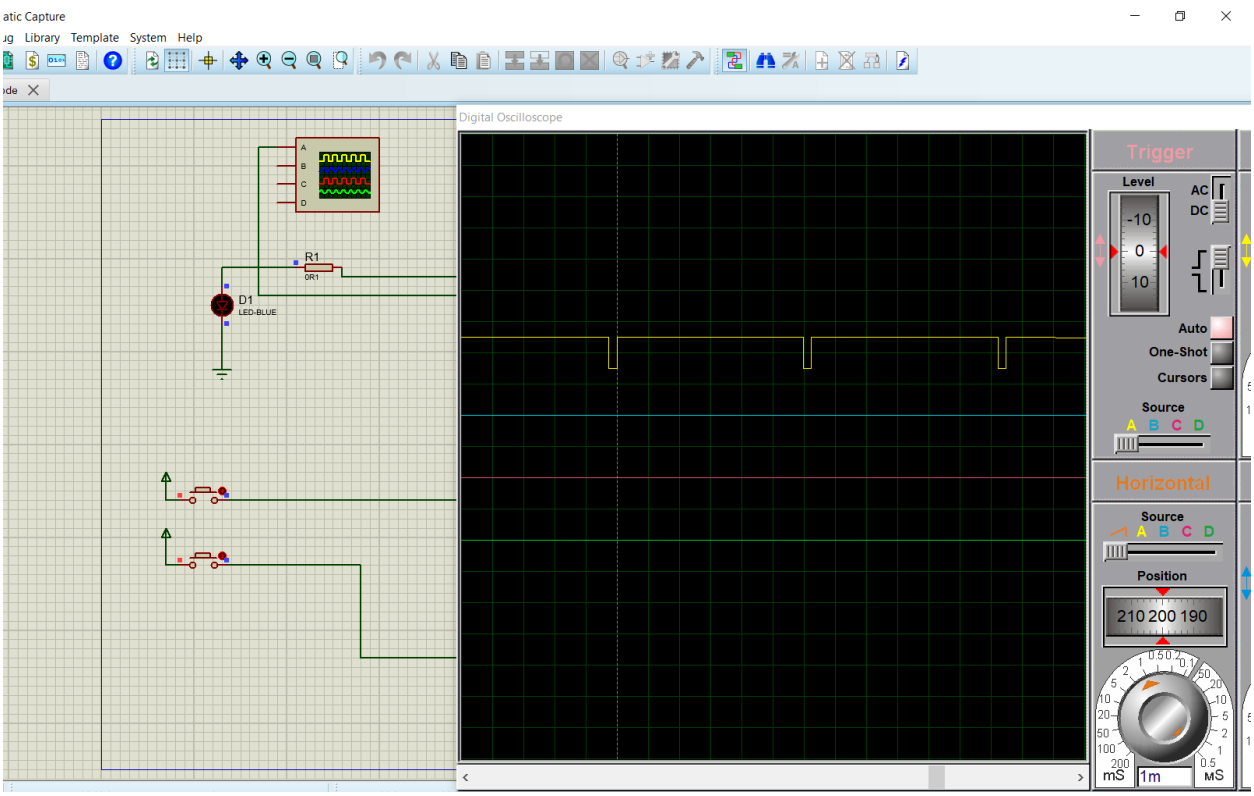
تصویر صفحه بعد وضعیت اسیلوسکوپ پیش از فشردن هر یک از دکمه هاست:



در ادامه و ابتدا با فشردن دکمه بالایی که به پین 12 وصل بوده و دوره و در نتیجه روشنایی را افزایش میدهد دوره سیگنال دچار تغییر میشود که این تغییرات در اسیلوسکوپ قابل مشاهده است:(چندبار فشرده شده)



و در نهایت دکمه پایینی که به پین 13 وصل بوده و روشنایی و چرخه سیگنال را تغییر میدهد را چندبار میفشاریم. نتیجه مشهود است:



سوالات تحلیلی- بخش B

1- Single conversion mode: در زمانهایی که قصد داریم تا تنها یکبار از ورودی آنالوگ نمونه برداری کنیم این حالت را به کار میگیریم به این صورت که یک تبدیل واحد از یک کانال ورودی آنالوگ انتخابی انجام شده و پس از آن ADC متوقف میشود و نمونه به میکروکنترلر برای پردازش بیشتر ارسال میشود.

Continuous conversion mode: در این حالت ADC از چندین کانال آنالوگ ورودی نمونه برداری کرده و هیچ مداخله نرم افزاری ای رخ نمیدهد. نتایج در یک بافر ذخیره میشوند.

Scan conversion mode: در زمانی که ترتیب نمونه برداری اهمیت دارد و نیاز داریم تا از چندین کانال به ترتیبی ثابت نمونه برداری کنیم از این حالت استفاده میکنیم. مشابه حالت قبلی است با این تفاوت که کانال های آنالوگ یک گروه از پیش تعریف شده هستند که پیش از نمونه برداری اسکن میشوند.

Injected conversion mode: در این حالت این امکان به ما داده میشود تا در زمانی که یکی از تبدیل های فوق انجام میشود تبدیل های اضافی را در کانال های ورودی آنالوگ انتخابی اعمال کنیم و از کانال های خاصی نمونه برداری کنیم.

Discontinuous conversion mode: مشابه حالت continuous بوده اما این امکان را به ما میدهد تا پس از انجام تبدیل تعداد مشخصی از کانال ها توالی تبدیل را قطع کنیم و ADC را برای انجام تبدیل زیرمجموعه ای از کانال ها به کار بگیریم و توان مصرفی را کاهش دهیم.

2- نرخ نایکوئیست مفهومی در پردازش سیگنال است به این صورت که حداقل نرخ نمونه برداری مورد نیاز برای نمایش دقیق سیگنال زمان پیوسته در حوزه دیجیتال را تعریف میکند.با توجه به این قضیه برای بازسازی صادقانه یک سیگنال زمان پیوسته از نمونه های گسسته آن بدون از دست دادن اطلاعات، نرخ نمونه برداری باید حداقل دو برابر بالاترین فرکانس موجود در سیگنال باشد که به آن پهنای باند سیگنال میگوییم.این نرخ دقیقا نیمی از نرخ نمونه برداری مورد نیاز برای گرفتن تمام اطلاعات ضروری در سیگنال زمان پیوسته اصلی است. این نرخ حداکثر فرکانس سیگنال دیجیتال را تعیین میکند.برای گرفتن جزئیات دقیق یک سیگنال پیوسته باید با نرخی بیشتر از نرخ نایکوئیست نمونه برداری کرد.

3- پارامترهای قابل تنظیم بسته به میکروکنترلر یا پلتفرم میتواند متفاوت باشد .

-baud rate: سرعت انتقال/ دریافت داده از طریق خط ارتباطی سریال که سرعت تغییر سیگنال را نیز تعیین میکند.

- بیتهای داده: امکان انتخاب بیت های داده در هر فریم.

- بیت های توقف: نشان دهنده تعداد بیت های پایان یک قاب داده

- برابری: یک مکانیسم برای تشخیص خطا در انتقال داده ها

- انتخاب حالت: امکان انتخاب بین حالت های همزمان (که در آن یک سیگنال بین فرستنده و گیرنده به اشتراک گذاشته میشوند) و ناهمزمان (سیگنال یا کلاک مشترک ندارد)

4- در مرحله اول enable uart peripheral که یعنی مطمئن شویم که uart روشن است که اینکار با تنظیم بیت فعال در رجیستر کنترل انجام میشود.مرحله دوم پیکربندی تنظیمات uart است که پارامترهای گفته شده در سوال سوم را در اینجا برای uart مقداردهی میکنیم و رجیسترهای مربوطه را تنظیم میکنیم.مرحله سوم enable receive interrupt که در اینجا رجیستر مسئول فعال سازی وقفه در uart که به آن رجیستر ثبت کننده وقفه (IER) یا ثبت نقاب وقفه (IMR) میگویند را پیدا کرده و بیت مربوط به وقفه دریافت را تنظیم میکنیم.در مرحله چهارم در صورتیکه میکروکنترلر از اولویت های وقفه پشتیبانی کند آن را از طریق ثبت اولویت کنترل تنظیم میکنیم.در مرحله پنجم یک روال سرویس وقفه(ISR) برای وقفه دریافت مینویسیم که در زمان رخ دادن وقفه دریافتی اجرا میشود .ISR باید داده‌های دریافتی را از رجیستر داده UART بخواند و مطابق آن عمل کند.مرحله آخر فعال کردن global interrupt enable است که در آن باید مطمئن شویم تمام وقفه ها با تنظیم بیت فعال وقفه global به درستی تنظیم شده است.