실험 결과 보고서
11주차 Timer PWM

실험 일자 : 2024. 11. 12.

실험 참여 : 3조 강태진, 김기윤, 김도완, 임성표

차례

I. 실험 목적

i) 임베디드 시스템의 기본 원리 습득

ii) Timer, PWM 이해 및 실습

II. 세부 목표

i). 레지스터 및 주소에 대한 설정 이해

- Datasheet 및 Reference Manual 참고.

ii) 예제 코드 학습 및 LCD 설정

iii) 타이머 종류 학습 및 활용 방식 이해

- 분주 계산법 학습

iv) 서보모터 제어

- 정방향, 역방향 회전 및 PWM 신호의 펄스 폭 변경을 통한 각도 제어

v) 종합적으로, Timer, PWM 서보모터 이해 및 실습

III. 실험 과정

2. 배경 지식

- 타이머
  - 주기적 시간 처리에 사용하는 디지털 카운터 회로 모듈
  - 펄스폭 계측, 주기적인 interrupt 발생 등에 사용
  - 주파수가 높기 때문에 우선 prescaler를 사용하여 주파수를 낮춘 후 낮아진 주파수로 8, 16비트 등의 카운터 회로를 사용하여 주기를 얻음
- 타이머의 종류와 특징
  - Systick Timer : Real-time operating system 전용이지만 standard down counter로 사용할 수도 있음
    - 24bit down counter
    - Autoreload capability
    - Counter가 0에 도달하면 설정에 따라 인터럽트가 발생
    - Programmable clock source
  - IWDG/WWDG Timer-Watching timer
    - WATCHDOG(WDG)은 임베디드 시스템 등 특수 상황에서 CPU가 올바르게

작동하지 않을 시 강제로 리셋 시키는 기능을 의미
  - 소프트웨어 고장으로 인한 오작동을 감지하고 해결하는 역할
  - 카운터가 주어진 시간 초과 값에 도달했을 때 시스템 재설정 또는 인터럽트(only WWDG)를 트리거
  - IWDG
    - IWDG는 LSI 클럭 기반으로 메인 클럭 고장에도 활성 상태 유지 가능
    - 타이밍 정확도 제약이 낮은 애플리케이션에 적합
    - 카운터가 0이 되면 Reset
  - WWDG
    - 7-bit down counter
    - WWDG의 클럭은 APB1 클럭을 프리스케일해서 정의 가능
    - 비정상적 어플리케이션 동작 감지를 위해 설정 가능한 time-window가 있음
    - Time-window 내에서 반응하도록 요구하는 애플리케이션에 적합
    - 카운터가 0x40 보다 작을 경우 또는 카운터가 Time-window 밖에 Reload 됐을 경우 Reset 가능
    - Early wakeup interrupt (EWI): 카운터가 0x40과 같을 때, EWI 인터럽트 발생하게 설정 가능
- Advanced-control Timer (TIM1, TIM8)
  - prescaler를 이용해 설정 가능한 16-bit auto-reload counter를 포함
  - 입력 신호 펄스 길이 측정(input capture) 또는 출력 파형 생성(output compare, PWM, complementary PWM with dead-time insertion) 등에 사용 가능
  - advanced-control (TIM1&TIM8)와 general-purpose (TIMx)는 자원을 공유하지 않는 독립적인 구조, 동기화 시키는 것도 가능
- General-purpose Timer (TIM2 ~ TIM5)
  - prescaler를 이용해 설정 가능한 16-bit up, down, up/down auto-reload counter를 포함
  - 입력 신호의 펄스 길이 측정(input capture) 또는 출력 파형 발생(output compare and PWM) 등 다양한 용도로 사용 가능
  - 펄스 길이와 파형 주기는 timer prescaler와 the RCC clock controller prescaler를 사용하여 몇 μs에서 몇 ms까지 변조 가능, 타이머들은 완전히 독립적이며, 어떤 자원도 공유하지 않으나 동기화 가능
- Basic Timer (TIM6, TIM7)

- 16-bit auto-reload up counter
- 설정 가능한 16-bit prescaler를 이용해 the counter clock 주파수를 나눠서 설정 가능
- DAC 트리거에 사용
- 카운터 오버플로우 발생 시 인터럽트/DMA 생성

- 분주 계산

$$\frac{1}{f_{clk}} \times prescaler \times period$$

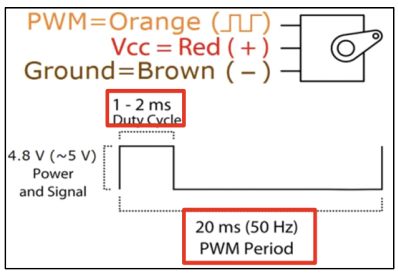
$$\frac{1}{72Mhz} \times 7200 \times 10000 = 1[s]$$

$$f_{clk} \times \frac{1}{prescaler} \times \frac{1}{period} = \text{주파수 } [Hz]$$

- 분주: MCU에서 제공하는 Frequency를 사용하기 쉬운 값으로 바꾸어 주는 것
- Counter clock frequency를 1~65536의 값으로 나누기 위해 16-bit programmable prescaler 사용
- period 로 몇 번 count하는지 설정

- PWM(Pulse Width Modulation)
  - 일정한 주기 내에서 Duty ratio를 변화시켜서 평균 전압을 제어하는 방법
  - Duty ratio
    - PWM 신호의 전체 주기 대비 HIGH신호가 유지되는 시간의 비율- Ex) 듀티 사이클이 50%라면 신로의 절반 시간동안 high(5V), 나머지 절반시간 동안 LOW(0V)로 유지
    - 이를 통해 0~5V의 전력 범위에서 평균 전압을 2.5V로 제어 가능
  - ex) 0~5V의 전력 범위에서 2.5V 전압을 가하고 싶다면, 50% Duty cycle 적용

- 서보모터
  - 대부분의 서보모터는 50Hz 주파수에서 작동하며 이는 20ms 주기를 의미.
  - SG90 서보모터는 PWM 신호의 듀티 사이클에 따라 위치를 제어
    - 1ms 펄스 폭: 서보모터가 왼쪽 끝으로 이동
    - 1.5ms 펄스 폭: 서보모터가 중간 위치로 이동
    - 2ms 펄스 폭: 서보모터가 오른쪽 끝 위치로 이동



IV. 실험 시행

- 실험 과정

(1) TFT LCD 에 team 이름, LED 토글 ON/OFF 상태, LED ON/OFF 버튼 생성

(2) LED ON 버튼 터치 시 TIM2 interrupt, TIM3 PWM 을 활용하여 LED 2 개와 서보모터 제어

a. LED: 1 초 마다 LED1 TOGGLE, 5 초 마다 LED2 TOGGLE

b. 서보모터: 1 초 마다 한쪽 방향으로 조금씩(100) 이동

(3) LED OFF 버튼 터치 시 LED Toggle 동작 해제 및 서보모터 동작 반전

a. 서보모터: 1 초 마다 반대쪽 방향으로 조금씩(100) 이동

- 코드

1. RccInit()

```
void RccInit(void) {
  RCC_APB1PeriphClockCmd(RCC_APB1ENR_TIM2EN, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
}
```

- APB1 버스: TIM2, TIM3 타이머 활성화
- APB2 버스: GPIOB, GPIOC, GPIOD 포트, AFIO (Alternate Function I/O) 활성화

2. GpioInit()

```
// TODO
void GpioInit(void) {
  GPIO_InitTypeDef GPIO_InitStructure;

  // LED setup on GPIOD Pin 2 and Pin 3
  // LED 1, 2번 설정
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOD, &GPIO_InitStructure);

  // PWM motor setup on GPIOB Pin 0
  // PWM으로 서브 모터 제어 신호 출력 설정
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOB, &GPIO_InitStructure);
}
```

- GPIO 핀 초기화 및 설정.
- GPIO_InitTypeDef 구조체와 GPIO_Init 함수를 사용하여 LED1, LED2를 General purpose output push-pull, max speed 50Mhz로 설정.

- TIM3의 채널 3을 Alternate function output push-pull, max speed 50Mhz로 설정

3. TIM_Configure() : TIM2와 TIM3 타이머를 초기화하고 설정하는 함수
   - 시스템의 기본 클락 72Mhz로 설정.

```
126     // TODO
127     void TIM_Configure(void) {
128       // TIM2 Configuration
129       TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
130       // 0부터 10000까지 count
131       // 0부터 세기 때문에, -1을 해줘야 함.
132       TIM_TimeBaseStructure.TIM_Period = 10000-1;
133       // prescaler 7200 -> 1초마다 이벤트 발생
134       // 시스템의 기본 클락 72Mhz -> prescaler = 7200, period = 10000
135       // 7200 = 72Mhz / 10000
136       // 그냥 7200 - 1을 사용해도 무방
137       TIM_TimeBaseStructure.TIM_Prescaler = (uint16_t) (SystemCoreClock / 10000)-1;
138       // clock 분할 x
139       TIM_TimeBaseStructure.TIM_ClockDivision = 0;
140       TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
141       // 카운트 업 모드, 0부터 period값 까지 증가
142       TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
143       // TIM2 위 설정으로 초기화
144       TIM_ARRPreloadConfig(TIM2, ENABLE);
145       // 타이머가 Period에 도달할 때 마다 초기화
146       TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
147       // 활성화
148       TIM_Cmd(TIM2, ENABLE);
150       // TIM3 Configuration
151       TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
152       TIM_OCInitTypeDef TIM_OCInitStructure;
153
154       // Enable the peripheral clock for TIM3
155       RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
156
157       // Configure the time base
158       // 20,000개 카운터
159       // 서브 모터는 50Hz(20ms)를 필요로 함.
160       TIM_TimeBaseStructure.TIM_Period = 20000 - 1;
161       // 시스템의 기본 클락을 72MHz라고 설정.
162       // 20ms가 되어야 하므로, prescaler이 72이면 1MHz/20,000 -> 50Hz가 성립
163       TIM_TimeBaseStructure.TIM_Prescaler = 72 - 1;
164       TIM_TimeBaseStructure.TIM_ClockDivision = 0;
165       // 카운트 다운 모드
```

```
165     // 카운트 다운 모드
166     TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Down;
167     //타이머 초기화
168     TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
169
170     // Configure TIM3 Channel 3 as PWM output
171     // 타이머의 카운트 값이 TIM_Pulse보다 작으면 신호 활성, 크면 비활성
172     TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
173     TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
174     // Example: 1ms pulse width, adjustable as needed
175     TIM_OCInitStructure.TIM_Pulse = 0;
176     // PMW신호 활성시 전압 HIGH로 출력
177     TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
178     TIM_OC3Init(TIM3, &TIM_OCInitStructure);
179
180     TIM_Cmd(TIM3, ENABLE);
```

- TIM2 : 1초 타이머 설정 (1Hz).
- TIM3 : PWM 출력 설정 (50Hz, 서보 모터 제어용)

4. NvicInit()

```
183   void NvicInit(void) {
184     NVIC_InitTypeDef NVIC_InitStructure;
185
186     NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
187
188     NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
189     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
190     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
191     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
192     NVIC_Init(&NVIC_InitStructure);
193
194     NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
195     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
196     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
197     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
198     NVIC_Init(&NVIC_InitStructure);
199   }
```

- NVIC_PriorityGroupConfig 함수를 사용하여 우선순위 그룹을 NVIC_PriorityGroup_1로 설정
- NVIC_InitTypeDef 구조체와 NVIC_Init 함수를 사용하여 각 Interrupt의 우선순위를 설정.

5. ledToggle()

```
201  //ledtoggle(n) -> n번 LED 점등
202  void ledToggle(int num) {
203    uint16_t pin;
204
205    switch (num) {
206      case 1:
207        pin = GPIO_Pin_2;
208        break;
209      case 2:
210        pin = GPIO_Pin_3;
211        break;
212      default:
213        return;
214    }
215
216    if (GPIO_ReadOutputDataBit(GPIOD, pin) == Bit_RESET) {
217      GPIO_SetBits(GPIOD, pin);
218    }
219    else {
220      GPIO_ResetBits(GPIOD, pin);
221    }
222  }
```

- GPIO_ReadOutputDataBit(GPIOD, pin)로 현재 LED 상태 읽기.
- GPIO_SetBits(): LED켜기 (HIGH 출력).
- GPIO_ResetBits(): LED 끄기 (LOW 출력).

6. moveMotor()

```
224  //motorDir == 0(LED On) -> 정방향 회전
225  //motorDir == 1(LED Off) -> 역방향 회전
226  void moveMotor() {
227    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
228    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
229    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
230    TIM_OCInitStructure.TIM_Pulse = motorAngle + 700;
231    if (motorDir == 0) {
232      motorAngle = motorAngle + 100;
233    if (motorAngle == 1500)
234      motorAngle = 0;
235    }
236    else {
237      motorAngle = motorAngle - 100;
238    if (motorAngle == 0)
239      motorAngle = 1500;
240    }
241
242    TIM_OC3Init(TIM3, &TIM_OCInitStructure);
243  }
```

- 서보 모터 회전 방향(motorDir), 각도(motorAngle)를 조절하여 PWM 신호 생성 및 모터 제어. 0일 때 정방향 회전, 1일 때 역방향 회전.

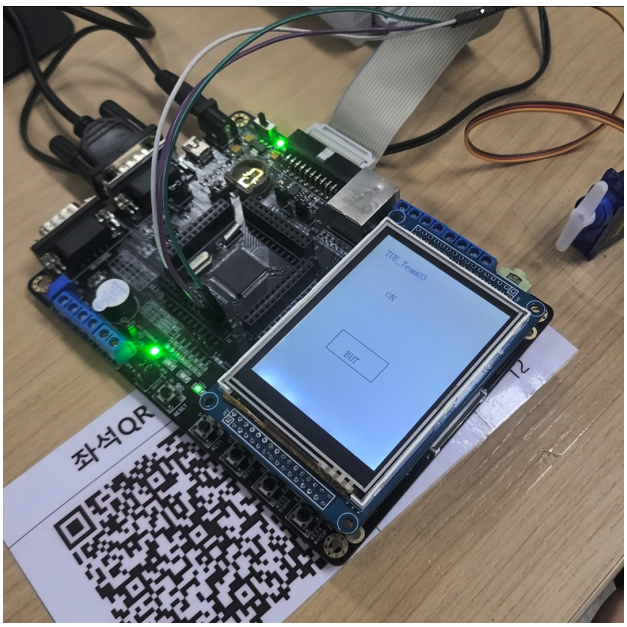
7. main()

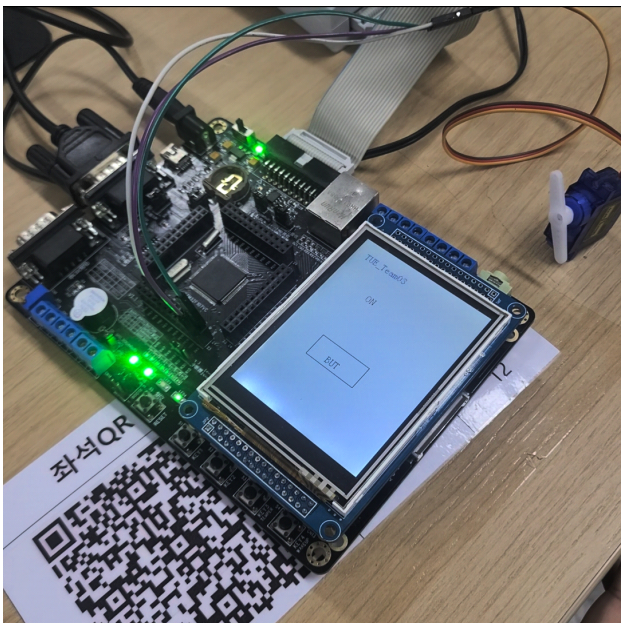
```
44  int main() {
45    Init();
46
47    LCD_Clear(WHITE);
48
49    // TODO: Display team name and button on LCD
50    LCD_ShowString(20, 20, "TUE_Team03", BLUE, WHITE);
51    LCD_ShowString(90, 200, "BUT", BLACK, WHITE);
52    LCD_DrawRectangle(60, 180, 150, 220);
53
54  // TODO: Motor setup
55    while (1) {
56    //LED OFF 상태
57      if (ledOn == 0) {
58        motorDir = 1;
59      }
60      //LED ON 상태
61      else {
62        motorDir = 0;
63      }
64
65    // Get touch position
66      Touch_GetXY(&x, &y, 1);
67      Convert_Pos(x, y, &x, &y);
68
69    // Detect button touch
70      if (60 <= x && x <= 180 && 150 <= y && y <= 220) {
71        if (ledOn == 0) {
72            LCD_ShowString(70, 80, " ", BLACK, WHITE);
73            ledOn = 1;
74            LCD_ShowString(70, 80, "ON", RED, WHITE);
75          } else {
76            LCD_ShowString(70, 80, " ", BLACK, WHITE);
77            ledOn = 0;
78            LCD_ShowString(70, 80, "OFF", RED, WHITE);
79        }
```

- LCD, 시스템 초기화 후 LCD에 팀명과 버튼 표시.
- LED 상태에 따라 모터 회전 방향 결정.
- 터치 입력을 읽어 버튼 입력 확인.
    - 버튼 입력 시 LED 반전 및 LCD에 ON, OFF 표시.

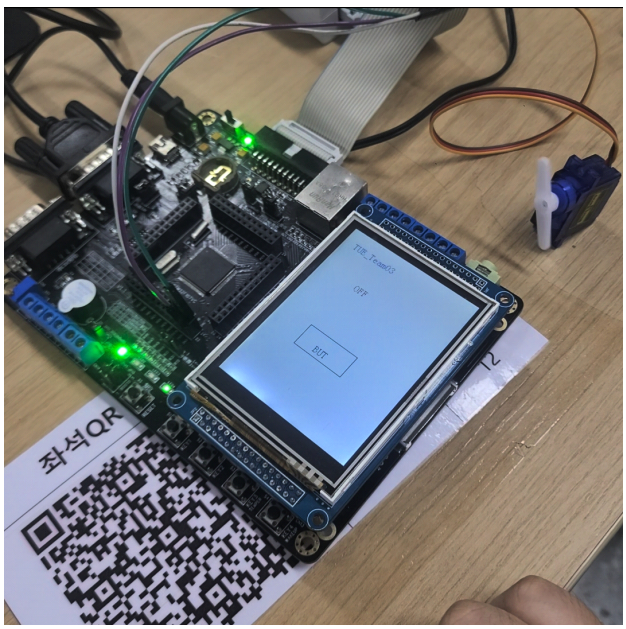
V. 실험 결과

- LCD ON -> LED 1초/5초 주기 토글 및 모터 정방향 회전





- LCD OFF -> LED 토글 정지 및 모터 역방향 회전



VI. 참고문헌

- 11주차 Timer PWM.pdf