ESD ASSIGNMENT SUBMISSION

Group Members: Date: 02/11/2022

Himanshu Kumar 2022H1400180P

Bharathi Shrinivasan T R 2022H1400182P

Modules prepared:

GPIO port bit set/reset register (GPIOx_BSRR) (x=A..I/J/K) –

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy:** Port x reset bit y (y = 0..15)

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Resets the corresponding ODRx bit

Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BSy:** Port x set bit y (y= 0..15)

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Sets the corresponding ODRx bit

- void GPIO_Init(void);
  This function is responsible for GPIO initialisation.

  __HAL_RCC_GPIOG_CLK_ENABLE() ;// Clock enable for GPIO port G for UserButton
  GPIO_InitTypeDef GPIO_UserPIN; // USER Pin @PORT-G
  GPIO_UserPIN.Pin = GPIO_PIN_15; //PIN-15
  GPIO_UserPIN.Mode = GPIO_MODE_INPUT;
  GPIO_UserPIN.Pull = GPIO_PULLUP; //upon press the value goes GND
  GPIO_UserPIN.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOG, &GPIO_UserPIN); // this function sets IDR,ODR registers accordingly
  GPIOG->BSRR=0xFFFF0000; // Reset

- Write and Read operation-
  - HAL_GPIO_WritePin(GPIOH, GPIO_PIN_2, GPIO_PIN_SET); // Warning LED lit
    This operation sets/resets <PORT>.PIN
  - HAL_GPIO_ReadPin(GPIOG, GPIO_PIN_15); // Read UserPUSH button
    This operation returns the current digital reading of <PORT>.PIN

TIMER – (Used TIM2 general purpose)

In this project, we have used TIM2 for clock sec updating.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UART8 EN	UART7 EN	DAC EN	PWR EN	Reser-ved	CAN2 EN	CAN1 EN	Reser-ved	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART 3 EN	USART 2 EN	Reser-ved
rw	rw	rw	rw		rw	rw		rw	rw	rw	rw	rw	rw	rw	
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
SPI3 EN	SPI2 EN	Reserved		WWDG EN	Reserved		TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw			rw	rw	rw	rw	rw	rw	rw	rw	rw

- First, enable the TIM2 clock by writing to '1' to bit 0 of RCC_APB1ENR register
- The SYSCLK is configured to 168Mhz, APB1 to 42 Mhz. In the SystemClock_Config() function in the code, the APB1 prescalar is set to 4. Hence the timer frequency will be twice that of APB1 frequency .

  Since the APB1 prescalar value is 4--> APB1 frequency =168/4=42Mhz and timer connected to APB1 receives a clock frequency is 84Mhz as input. The timer can be programmed to count a frequency decided by the prescalar for timer. [To summarize APB1 frequency is 42 Mhz. However, timers connected to APB1 peripheral receives a frequency of 84Mhz. However, the clock input to timer can be further modified using a clock divider. The clock division value is configured using prescalar). Now if PRESCALAR is 8400, the timer works at a frequency of 84000000/8400=10000Hz or .0001seconds.

  The clock divider for timer, divides the input clock by a factor of prescalar+1 hence the 8399 should be written to TIM2->PSC (prescalar register) to provide the clock to the timer as 10000Hz.

  If the ARR_ Value is set to 9999, then the counter will run for **1 second** and raise the interrupt (as the interrupt is raised only after the counter count backs to zero. Hence, if the count is set to 9999, the interrupt will be raised after 1 second. TIM2->ARR is used to set the ARR value

- To enable interrupts, TIM2->DIER=(1UL<<0); //DMA/interrupt enable register
- Configure the nested vectored interrupt controller to respond to TIM2 interrupts

  TIM2->CR1=(1UL<<0); enable/start Timer control register. This basically starts the timer.

18.4.1 TIMx control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- In the timer interrupt handler, the interrupt flag Bit 0 UIF is set by hardware upon interrupt. It has to cleared by software during ISR().

18.4.5 TIMx status register (TIMx_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Reserved		TIF	Res	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0			rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

- Initialisation function()

  void TIM2_Initialize (void){ // Used for 1 sec interrupt -> to updateClock()
  const uint16_t PSC_val = 8400; //prescalar value 84MHZ/8400

```
const uint16_t ARR_val= 10000;//Auto Reload Register value (5000) every 1 sec the event occurs
RCC->APB1ENR|=(1UL<<0);//enable timer clock
TIM2->PSC=PSC_val-1;//comparison of ARR and Timer cont will take 1 cycle hence count-1 has to be uploaded for both PSC & ARR
TIM2->ARR=ARR_val-1;//
TIM2->DIER=(1UL<<0); // enable interrupt for event
NVIC_EnableIRQ(TIM2_IRQn);
//TIM2->CR1=(1UL<<0);//set control register >> counter enable (enable/disable via software)
}
```

- ISR handler routine:

```
void TIM2_IRQHandler(void){ // TIM2 interrupt service
        TIM2->SR &=~(1<<0); // clear interrupt flag

        /*==== Update clock ==*/
        togg=~togg; //timer pulse monitoring
        sec=sec+1; // update sec
        if (sec>59)
        {
                sec=0;
                min=min+1; // update min
                if(min>59) {min=0;}
                /*==== Sampling section ==*/
                int remainderMin=min; // @min= 0,5,10.. initiate an ADC sample()
                for(;remainderMin>=5;)remainderMin=remainderMin-5;
                if (remainderMin==0)
                        {ADCSample();}//Sample }
        }
}
```

The 5mins ADC sample is done by checking if min is multiple of 5 at every update of minute.

ADC Initialise:

- Enable clock to ADC3 clock setting bit 10 of RCC_APB2ENR register (As ADC is connected to APB2). This should be performed before any of the ADC registers are accessed.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													TIM11 EN	TIM10 EN	TIM9 EN
													rw	rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reser-ved	SYSCF G EN	Reser-ved	SPI1 EN	SDIO EN	ADC3 EN	ADC2 EN	ADC1 EN	Reserved		USART 6 EN	USART 1 EN	Reserved		TIM8 EN	TIM1 EN
	rw		rw	rw	rw	rw	rw			rw	rw			rw	rw

- Enable clock to GPIOF using bit 5 of RCC_AHB1ENR register (As GPIOF is connected to AHB1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reser-ved	OTGH S ULPIE N	OTGH SEN	ETHM ACPTP EN	ETHM ACRXE N	ETHM ACTXE N	ETHMA CEN	Res.	DMA2D EN	DMA2E N	DMA1E N	CCMDAT ARAMEN	Res.	BKPSR AMEN	Reserved	
	rw	rw	rw	rw	rw	rw		rw	rw	rw			rw		
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved			CRCE N	Res.	GPIOK EN	GPIOJ EN	GPIOIE N	GPIOH EN	GPIOG EN	GPIOFE N	GPIOEEN	GPIOD EN	GPIOC EN	GPIO BEN	GPIO AEN
			rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Set the mode of GPIOF port pin7 as analog. (Using GPIOF_MODER register)- PF7 will be the analog input pin. Refer Table 1- ADC3 details. We will use channel 5 for providing input.So choose pin 14,15 should be 11This is done using statement GPIOF->MODER|=(3UL<<14);// GPIO mode PF7 in Analog mode,7th pin of PORT-F is in analog mode to configure the ADC to assign the channel to be first converted SQR1,SQR2,SQR3 registers may be used. To say channel 5 has to be converted first, write 5 into SQR3[4:0] bits.
- Sample and Hold time can be set for each channel by writing to the two ADC Sample Time registers (SMPR1 and SMPR2). In this case, as the input voltage is derived from light sensor whose output will not change rapidly. Hence the sample /hold time of 480 cycles can be set.
- Enable ADC3: ADC3->CR2|=(1UL<<0);// ADC Enable
- Enable ADC to start conversion by - ADC3->CR2 |= ADC_CR2_SWSTART; // start conversion

13.13.3 ADC control register 2 (ADC_CR2)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved	SWSTART	EXTEN		EXTSEL[3:0]				reserved	JSWSTART	JEXTEN		JEXTSEL[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved				ALIGN	EOCS	DDS	DMA	Reserved						CONT	ADON
				rw	rw	rw	rw							rw	rw

- EOC and data read: while(~((ADC3->SR) & 1UL)); // wait for EOC

13.13.1 ADC status register (ADC_SR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										OVR	STRT	JSTRT	JEOC	EOC	AWD
										rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

- Read data: ADCData=(int)ADC3->DR;

Level-isation of light intensity:

ADC is set in 8-bit mode so 0v (0x00) and 3.3v(0xFF). Level-0 is value>0xC8. Level-1 is 0xC8>value>0x96. Level-2 is 0x64. Level-3 is 0x32.

Storage of light intensity in array:

uint16_t StorageArray[30]={0}; // global array to store sampled data in array

- ADCSample(); is used to sample a value (ADC read) and threshold level-lisation, and save it in the global array. Increment a sampleCounter to keep record.
- Display the count of level-0. The below function is responsible to count the StorageArray for level-0. And that count is converted into String, and displayed in LCD.

```
void Display_Level0_Count (void)
{
        GLCD_DrawString    (0, 5*24, "              ");
        GLCD_DrawString    (0, 6*24, "              ");
        int Level0Count=0;
        for(int i=0; i<SampleCount;i++)
        {
                if(StorageArray[i]==0) Level0Count++; // if level-0 count
```

```
			}
			GLCD_DrawString	(0, 5*24, "Level-0 count:        ");
			//Display_ADCValue(Level0Count);

			char int_string[10];
			int i;

/*Convert the integer value into String so as to print in LCD*/
			for(i=0;~(Level0Count==0) && (i<10);i++)
			{
				int_string[9-i]=Level0Count%10 + 48; // 48->'0'
				Level0Count=Level0Count/10;
			}
			GLCD_DrawString	(0, 6*24, int_string);
}
```

Flow Chart: