Embedded Thursday

Variable + Timers + PWM + Intro to Interrupts

Ivan Quiroz

Today

- The project Goal and System
- Recap
- Variables
- Hardware PWM
- Timers
- Intro to Interrupts

Goal Description

› Learn C as embedded language

› Use C to understand underlying processor

› Have a project so learning stays
  – We are making a self balancing robot

Todays Goal

– Learn to set PWM
– Learn to set Timers
– Learn how interrupts work

Recap on Inputs/Outputs and registers

- Step 1: enable ports use register RCC_AHB1ENR
- Step 2: set Ports as IN or OUT writing to register GPIOx_MODER
- Step 3: set Pin HIGH or LOW writing to register GPIOx_ODR
- Step 4: read Input Pin by reading register GPIOx_IDR
- Debounce if you are reading a switch

Operators

- XOR ^= Use: switch bit to opposite value
- OR |= Use: Impact a bit, don't disturb others by OR'ing desired bit with 0x001
- AND &= Use: Impact a bit, don't disturb others by OR'ing desired bit with 0x110
- AND & Use: Mask a bit with using 0x001

Variables brief Introduction

Variable allocate a location in memory during compiling

The datatype defines the expected data we will use in a variable

```
#include <stdint.h>
```

- `int8_t`
- `int16_t`
- `int32_t`

- `uint8_t`
- `uint16_t`
- `uint32_t`

Data type	Precision	Range
unsigned char	8-bit unsigned	0 to +255
signed char	8-bit signed	-128 to +127
unsigned int	compiler-dependent	
int	compiler-dependent	
unsigned short	16-bit unsigned	0 to +65535
short	16-bit signed	-32768 to +32767
unsigned long	unsigned 32-bit	0 to 4294967295L
long	signed 32-bit	-2147483648L to 2147483647L
float	32-bit float	±10-38 to ±10+38
double	64-bit float	±10-308 to ±10+308

`Volatile:` A variable that may change at any time without any action being taken by the code

```
volatile int8_t Switchstatus
```

In embedded volatile is used to

- Define I/O ports (value of ports can change outside of software action. i.e. switch pressed
- Share a global variable between the main program and an interrupt service routine.
- Global variables accessed by multiple tasks within a multi-threaded application

Variables

```
int main(void)
{
//  RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN; // enable
    RCC->AHB1ENR |= 0x00000008;      // enable the
    RCC->AHB1ENR |= 0x00000001;      // enable the

    GPIOD->MODER |=  0X55000000;     // Set Port-D
    GPIOA->MODER &=  0XFFFFFFFE;        // Set Por
/*  GPIOD->MODER |= (1 << 24);       // another way

    int8_t i;
    volatile int SwitchStatus;

    GPIOD->ODR = 0x0000;                              //

    while (1){
/*      GPIOD->ODR ^= (1 << 12);     // another way


        SwitchStatus = ((GPIOA->IDR & 0x1) == 0);

        if (!SwitchStatus){
//          GPIOD->ODR ^= 0b1010000000000000;    //
            GPIOD->ODR ^= 0xD000;                //
//          GPIOD->ODR |= 0xF000;
            for (i = 0; i < 500000; i++);        //
        }
    }

}
```

Table 1. STM32F411xC/E register boundary addresses

Boundary address	Peripheral	Bus	Register map
0x5000 0000 - 0x5003 FFFF	USB OTG FS	AHB2	*Section 22.16.6: OTG_FS register map on page 744*
0x4002 6400 - 0x4002 67FF	DMA2		*Section 9.5.11: DMA register map on page 194*
0x4002 6000 - 0x4002 63FF	DMA1		
0x4002 3C00 - 0x4002 3FFF	Flash interface register		*Section 3.8: Flash interface registers on page 58*
0x4002 3800 - 0x4002 3BFF	RCC		*Section 6.3.22: RCC register map on page 133*
0x4002 3000 - 0x4002 33FF	CRC	AHB1	*Section 4.4.4: CRC register map on page 68*

#define RCC ((RCC_TypeDef *) RCC_BASE)

Pointer Definition

0x40023800

#define PERIPH_BASE ((uint32_t) 0x40000000U)

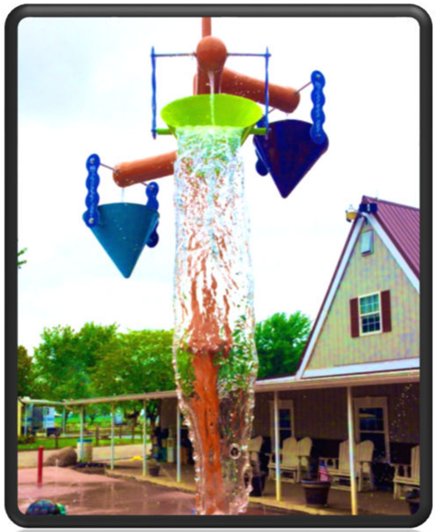
#define AHB1PERIPH_BASE (PERIPH_BASE + 0x00020000U)

#define RCC_BASE (AHB1PERIPH_BASE + 0x3800U)

Table 21. RCC register map and reset values for STM32F411xC/E

Addr. offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x30	RCC_ AHB1ENR	Reserved									DMA2E	DMA1E	Reserved								CRCEN	Reserved				GPIOHE	Reserve		GPIOEE	GPIODE	GPIOCE	GPIOBE	GPIOAE

Timers – TIM4



- A timer is a special register that once enabled it counts
  - The bucket to count is only so big
  - Once the bucket is full, it overflows
  - You can prefill the bucket
  - You can set the speed to fill the bucket
  - Interrupts can inform you if bucket has overflown
- We will use the Advance Control Timer TIM4
  - 16 Bit bucket 2^{16} :: 0 to 65,536 (count up/down)
  - Once it reaches value on Auto-Reload Register it restarts
  - We will use it for PWM generation (square wave form)
  - Use pre-scalars to set speed of count

STM32F411E-DISCO

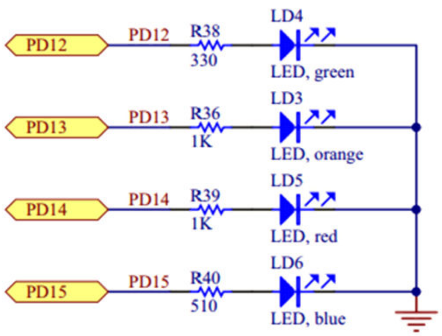


Table 9. Alternate function ma

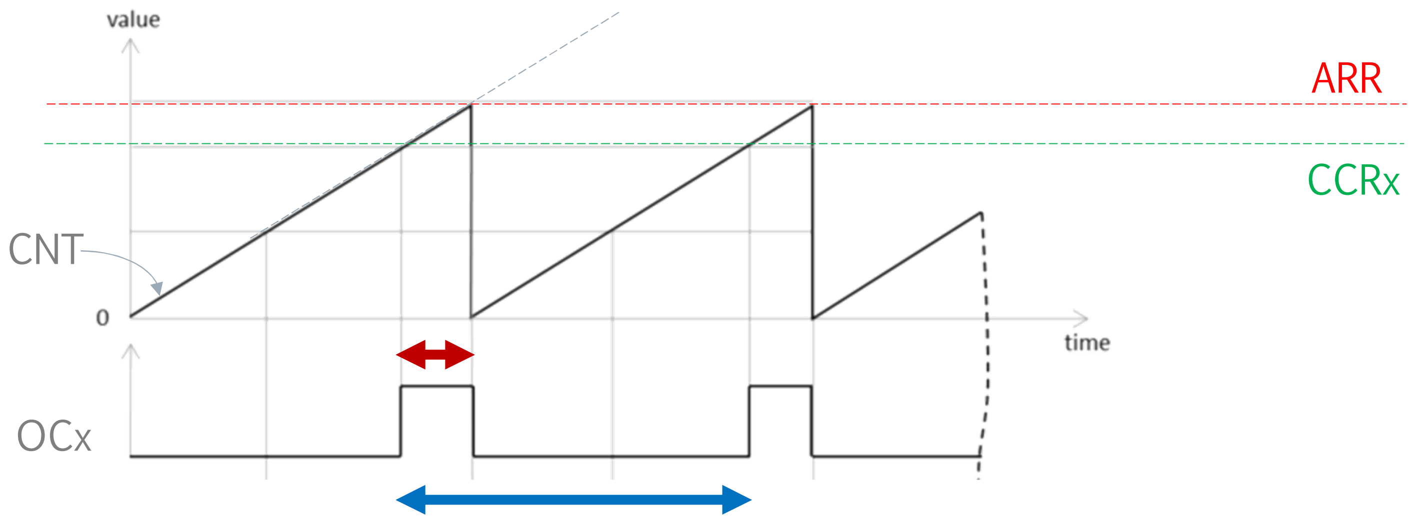
Port		AF00	AF01	AF02	AF03	AF04	AF05	AF06
		SYS_AF	TIM1/TIM2	TIM3/ TIM4/ TIM5	TIM9/ TIM10/ TIM11	I2C1/I2C2/ I2C3	SPI1/I2S1S PI2/ I2S2/SPI3/ I2S3	SPI2/I2S2/ SPI3/ I2S3/SPI4/ I2S4/SPI5/ I2S5
Port D	PD12	-	-	TIM4_CH1	-	-	-	-
	PD13	-	-	TIM4_CH2	-	-	-	-
	PD14	-	-	TIM4_CH3	-	-	-	-
	PD15	-	-	TIM4_CH4	-	-	-	-

Given location of LED we will use Timer 4 (TIM4) to generate PWM

- Port-D.Pin12: AF02 - CH1
- Port-D.Pin13: AF02 - CH2
- Port-D.Pin14: AF02 – CH3
- Port-D.Pin15: AF02 – CH4
- *Source: STM32F411 datasheet table-9*

PWM Mode (Reference Manual 13.3.9)

› Generate a square signal of determined frequency
  - Frequency determined by TIM4_ARR register
  - Duty Cycle determined by TIM4_CCR1 register



FREQUENCY = ON and OFF per second

DUTY CYCLE = Ratio of time ON to OFF

PWM Setup (follow section 13.3.9)

a. Configure Port-D as outputs
   a. Enable clock to Port D
   b. Set PD12-15 as outputs

b. Set up timer to start counting
   a. Count upwards
   b. Set the period
   c. Set clock divider
   d. Set prescalar

c. Set Port-D PD12-15 to alternate Function

d. Configure timer for duty cycle
   a. Set CH1-4 to PWM mode
   b. Select AF2 for PD12-14

TIM4 registers to use:

- ❑_ARR
- ❑_CCRx
- ❑_CCMR1
- ❑_CR1
- ❑_EGR
- ❑_CCER
- ❑_SR
- ❑_OSPEER
- ❑_CNT
- ❑_PSC

Autoreload reg determines PWM frequency

$$\text{PWM frequency} = \frac{\text{Counter clock (21MHz)}}{\text{Autoreload value} + 1}$$

Compare reg determines PWM duty cycle

$$\text{Duty Cycle} = \frac{\text{Compare reg value} * 100}{\text{Autoreload value} + 1}$$

Code – still needs work

```
int main(void)
{
//  RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN; // enable the clock to PORT-D using HALs definitions
    RCC->AHB1ENR |= 0x00000008;     // enable the PORT-D
    RCC->AHB1ENR |= 0x00000001;     // enable the PORT-A
    RCC->APB1ENR |= 0x00000004;     //  Enable TIM4 timer


    GPIOA->MODER &=  0xFFFFFFFE;        // Set Port-A as inputs
/*  GPIOD->MODER |= (1 << 24);      // another way to set pin 12 to be general purpose output
    GPIOD->MODER |= (1 << 26);      // another way to set pin 13 to be general purpose output
    GPIOD->MODER |= (1 << 28);      // another way to set pin 14 to be general purpose output
    GPIOD->MODER |= (1 << 30);      // another way to set pin 15 to be general purpose output
*/
    GPIOD->AFR[2]   |= 0x22220000;      //  Enable alternate functions using AFRH
    GPIOD->MODER    |=  0xAA000000;     //  Set Port-D pin12 to 14 to alternate function OUTPUTS
    GPIOD->OSPEEDR  |= 0xAA000000;      // set port speed to fast for ports D12-14 (50Mhz)

    TIM4->EGR    |= 0x0001;             // set udpate generation
    TIM4->CCMR1  |= 0x006C;             // Set PWM Mode 1 and enable ARR register
    TIM4->CCER   |= 0x1111;             // Set all channels (and pins outputs) to active HIGH
    TIM4->SMCR   |= 0x0030;             // Trigger selection to internal trigerr based on TIM4
    TIM4->CNT    |= 0x0000;             // Counter Register at zero
    TIM4->PSC    |= 0x0001;             // set prescaler to APB1/2  (21Mhz)
    TIM4->ARR    |= 0x1067;             // Computed by 21Mhz/(4199+1). We want 5Khz = 21Mhz / (ARR + 1) solve for ARR
    TIM4->CCR1   |= 0x0000;             // Duty Cycle Using 5Khz as reference then (4199+1) is to 100% PWM as x is to 50%. Solv
    TIM4->CCR2   |= 0x0834;             // Duty Cycle Using 5Khz as reference then (4199+1) is to 100% PWM as x is to 50%. Solv
    TIM4->CCR3   |= 0x0834;             // Duty Cycle Using 5Khz as reference then (4199+1) is to 100% PWM as x is to 50%. Solv
    TIM4->CCR4   |= 0x0834;             // Duty Cycle Using 5Khz as reference then (4199+1) is to 100% PWM as x is to 50%. Solv
    TIM4->CR1    |= 0x0085;             // Set ARR to buffered, PWM edge align, upcount timer, no counter stop. ENABLE COUNTER
```

Application

Extra Activities

Homework

› Create block diagram of design

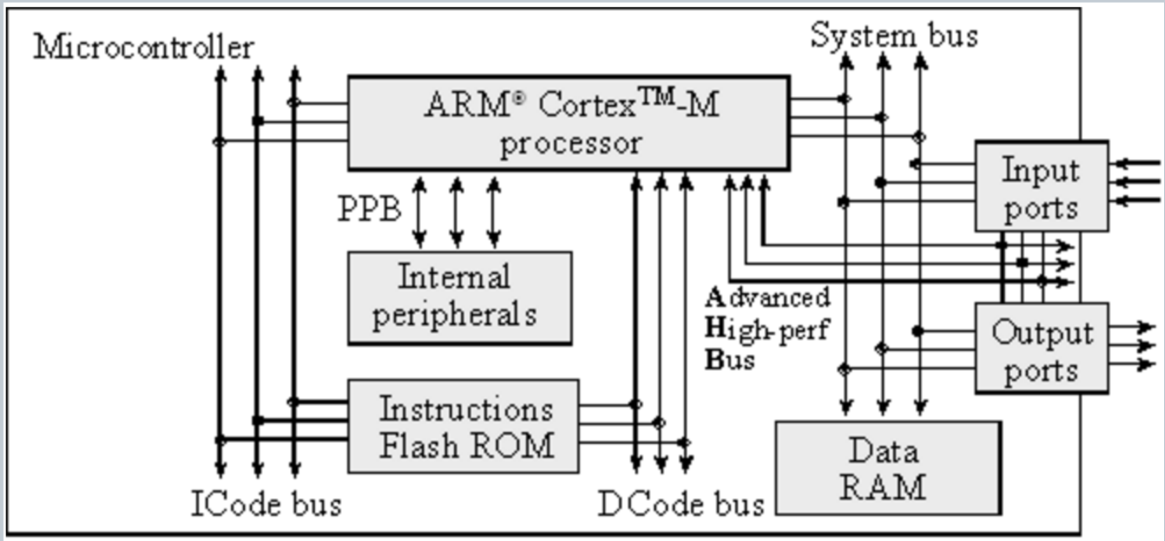
› Get switch to blink lights at different rates

Back Up Slides

Hardware Reference Material

SIMPLIFIED STM34F411 ARCHITECTURE

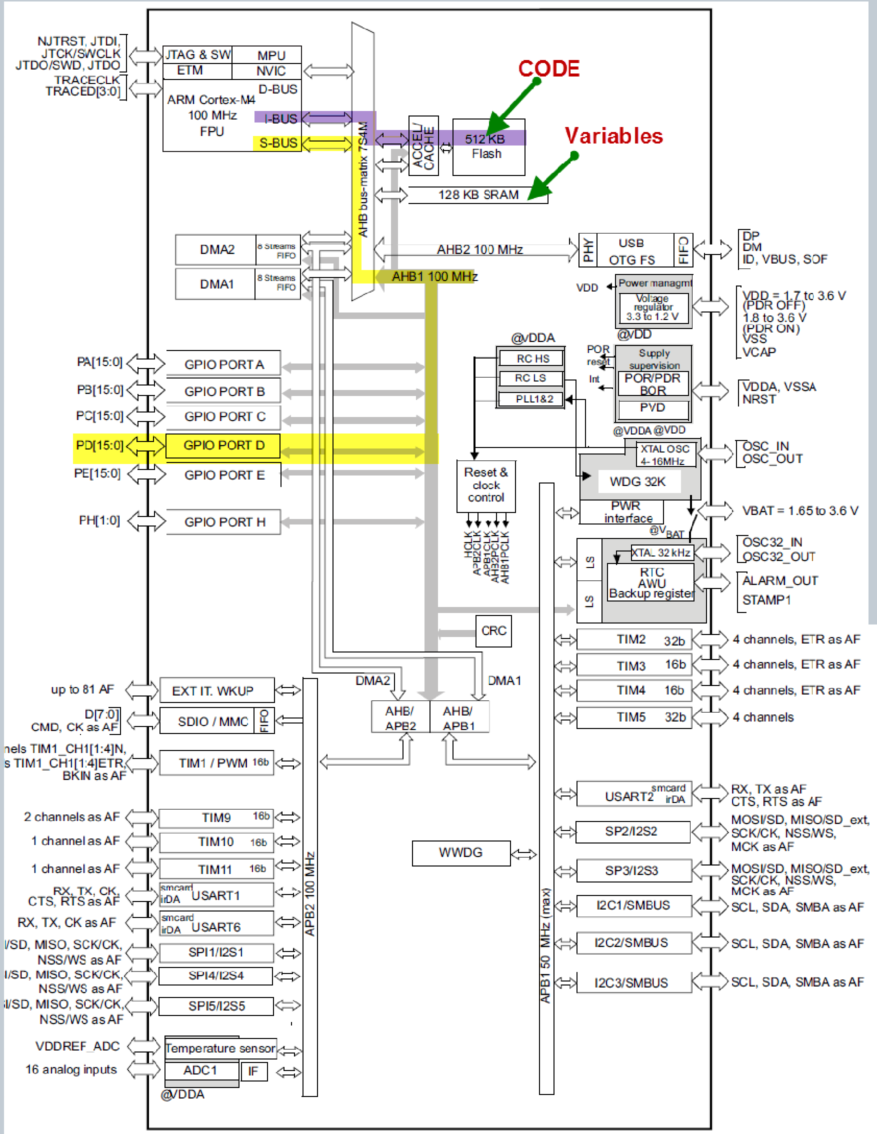
- I-Code Bus use to fetch instructions from Flash ROM
- System Bus: use to work with variables and IO Ports
- D-Code Bus: debug bus
- Adv Hi Bus: Connection to IO ports and dedicated USB ports



STM32F411 BLOCKDIAGRAM

Note the following buses:

- RCC->AHB1ENR needed as Port D uses AHB1 (yellow)



PWM DIAGRAM

CLOCK GENERATION

- APB1 clock is used at 42Mhz
- Prescaler set to divide by 2
- For a 10Khz PWM a ARR of 2099 would be used
  - Consider the 21Mhz clock used