Embedded Thursday

Inputs + Switch Bouncing + if & while statement

Today

- The project Goal and System
- Recap
- While loop
- Hardware Inputs
- Update Blinky

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 get an input from the board
  - Learn how to set up the serial port
  - Learn how to set up a timer

Recap on 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

Note: Register functionality definition is in the datasheet

Operators

- XOR ^= Use: switch bit to opposite value
- OR |= Use: Impact a bit, don’t disturb others

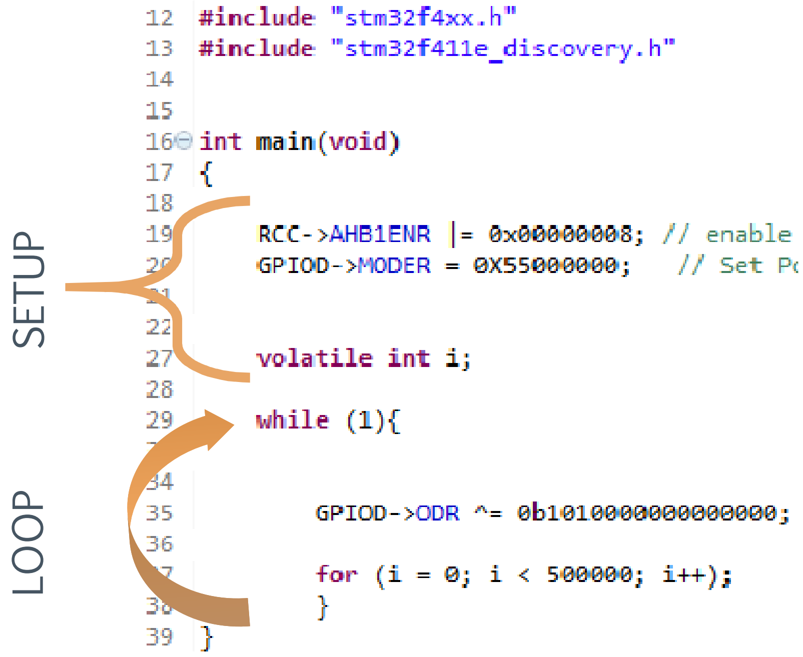
Examples

- RCC_AHB1ENR = 0x00000008 Enable GPIO D
- RCC_AHB1ENR |= 0x00000008 Add GPIO D enable
- GPIOD_MODER = 0X55000000 Set PortD-Pin12 to 15 to OUTPUTs
- GPIOD_ODR ^= 0b1010000000000000 swap value of bit at location of 1, write 1

The while loop

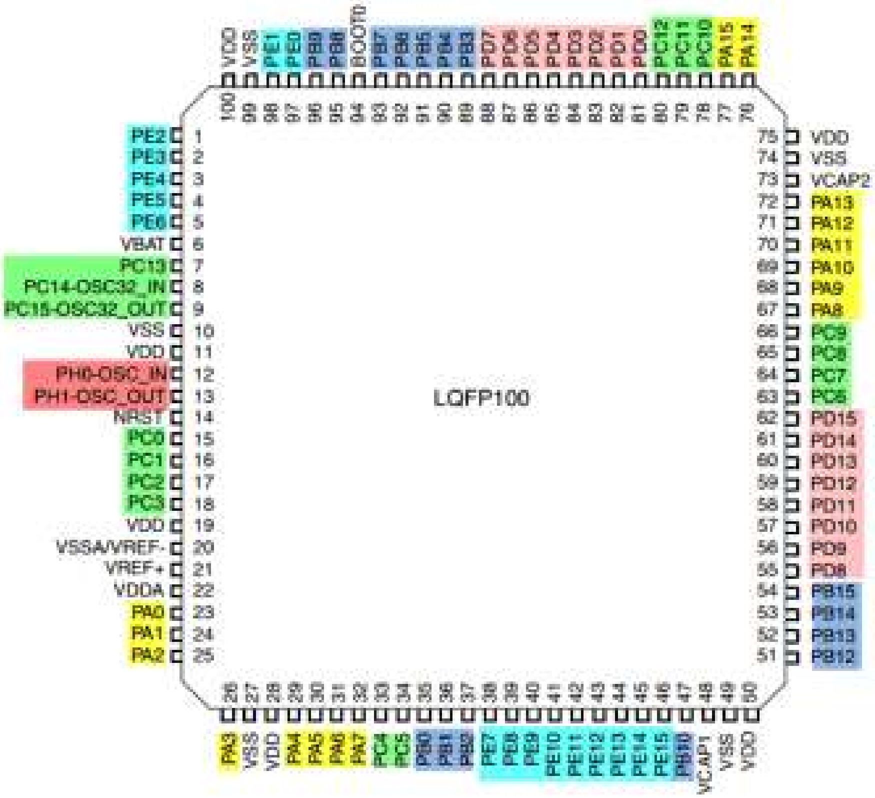
- A controller at power up always start at the same start location
- Program will enter the *main(void)* function
  - It will execute each line of code once and move on
  - Once it finds the last code it will stop for ever
    - An infinite loop allows a microcontroller to keep doing things over and over
- While will perform statements within bracket if condition is TRUE
  - Zero (0): Condition = FALSE
  - Non-Zero (i.e.1): Condition = TRUE

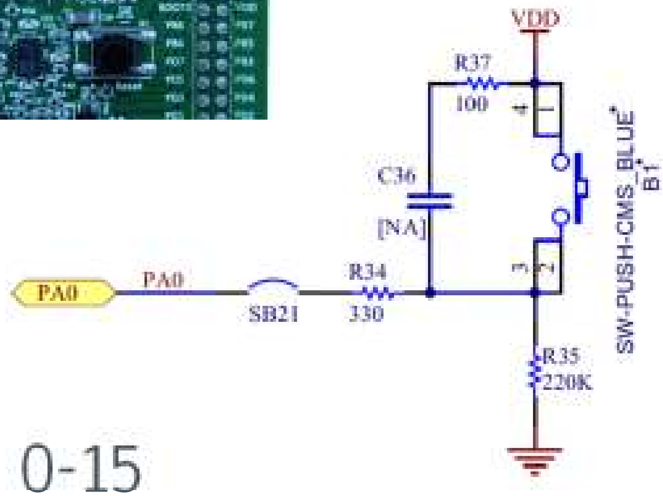
```
while(condition)  {
   statement1;
   statement2;
}
```



```
#include "stm32f4xx.h"
#include "stm32f411e_discovery.h"


int main(void)
{

	RCC->AHB1ENR |= 0x00000008; // enable
	GPIOD->MODER = 0X55000000;    // Set P


	volatile int i;

	while (1){


		GPIOD->ODR ^= 0b1010000000000000;

		for (i = 0; i < 500000; i++);
	}
}
```

STM32F411E





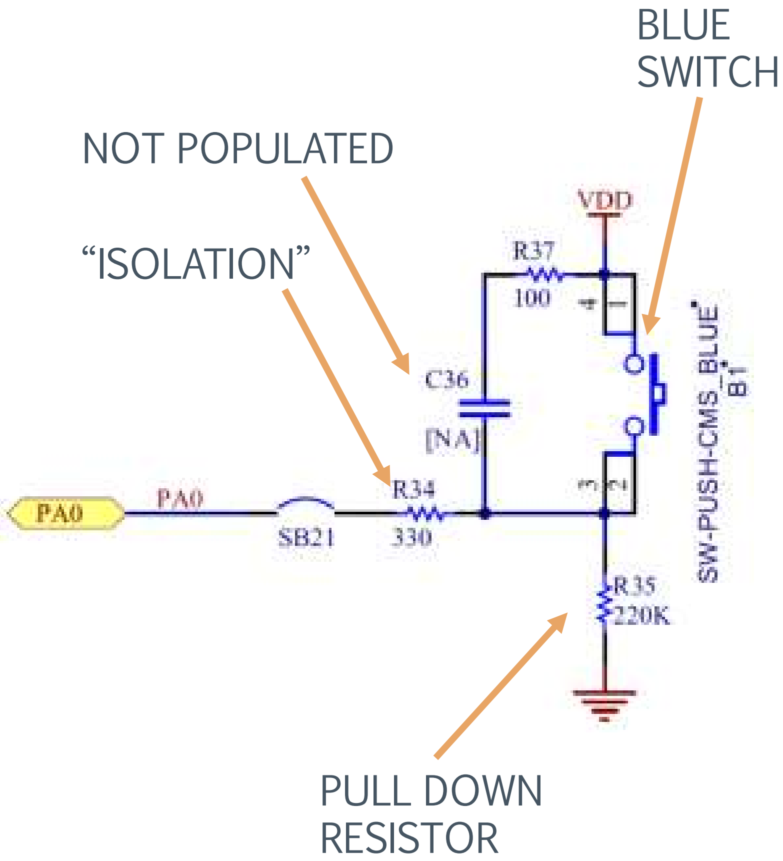
- Port-A: 0-15
- Port-B: 0-10, 12-15
- Port-C: 0-15
- Port-D: 0-15
- Port-H: 0-1

Hardware of Inputs

- Input should always be defined as one of two states
  - Active LOW
  - Active HIGH

Board has Active HIGH switch

- When the switch is pressed voltage VDD will be connected to PA0
- When switch is not pressed the resistor will pull down the voltage to ground



BLUE SWITCH

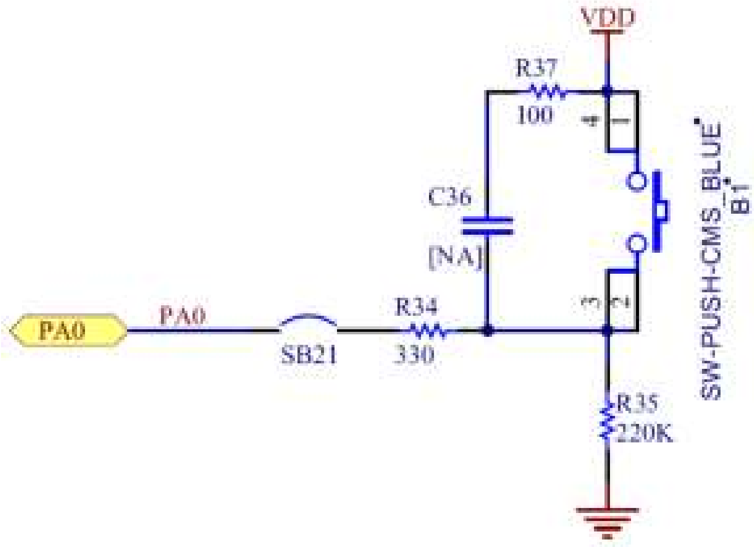
NOT POPULATED

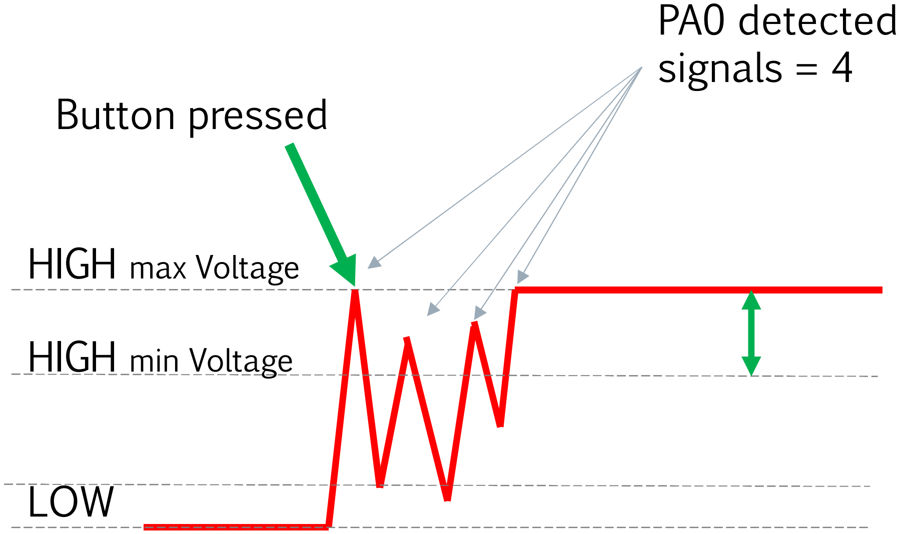
“ISOLATION”

PULL DOWN RESISTOR

SWITCH BOUNCING

- Physical reaction of contacts inside switch
- When switch is pressed the contacts bounce up and down microscopically
- Bounce is seen by microprocessors as multiple button presses




PA0 detected signals = 4

Button pressed

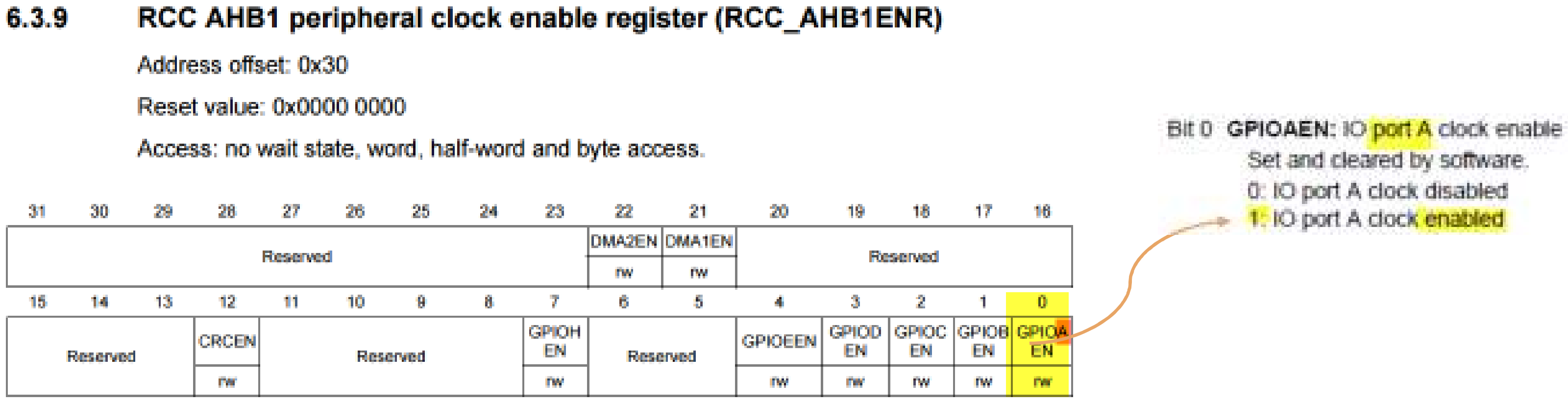
HIGH max Voltage

HIGH min Voltage

LOW

Enable Ports to use

- › Switch is physically located on PA0
- › First thing to do, enable Port-A



6.3.9 RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

Address offset: 0x30

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									DMA2EN	DMA1EN	Reserved				
									rw	rw					

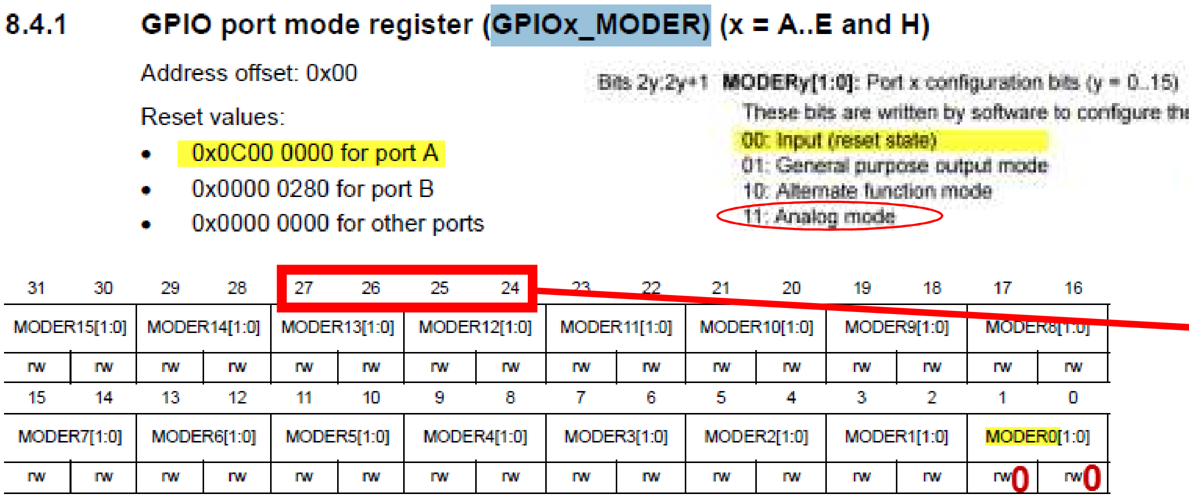
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CRCEN	Reserved				GPIOH EN	Reserved		GPIOEEN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
			rw					rw			rw	rw	rw	rw	rw

Bit 0 **GPIOAEN:** IO port A clock enable
Set and cleared by software.
0: IO port A clock disabled
1: IO port A clock enabled

- RCC_AHB1ENR = 0x00000001 Enable GPIO A (…or)
- RCC_AHB1ENR |= 0x00000001 Add GPIO A enable

Define Port direction

- We need Port-A Pin0 to be an INPUT
- Register GPIOx_MODER (General Purpose Input Output Mode Register) controls direction



8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..E and H)

Address offset: 0x00

Reset values:

- 0x0C00 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

Bits 2y:2y+1 **MODERy[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

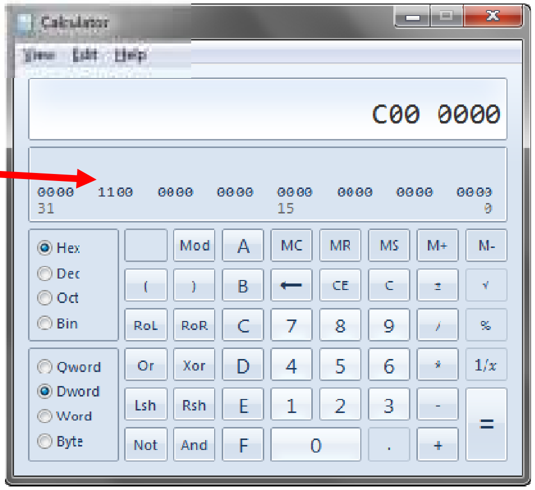
00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

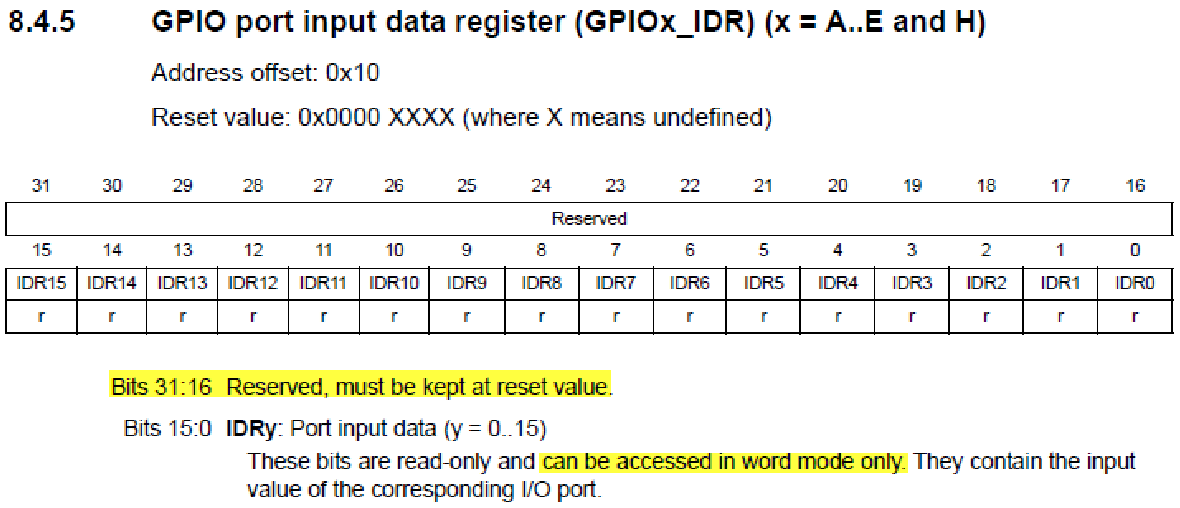
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw 0	rw 0



GPIOA_MODER = 0X00000000; Set all PORT-A to Inputs

Read Pin Status

- The input data register (GPIOx_IDR) captures the data present on the I/O pin at every AHB clock cycle.
  - AHB clock cycle max 100Mhz



8.4.5 GPIO port input data register (GPIOx_IDR) (x = A..E and H)

Address offset: 0x10

Reset value: 0x0000 XXXX (where X means undefined)

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**
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy**: Port input data (y = 0..15)

These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

- PortA_status = GPIOA_IDR

Reading Button Pressed

- Step 1: Start button not pressed
  - PA0= LOW
  - If GPIOA_IDR=0b0000 0000
    - Then GPIOA->IDR & (0x1) = TRUE
- Step 2: Button pressed
  - PA0= HIGH
  - If GPIOA_IDR=0b0000 0001
    - Then GPIOA->IDR & (0x1) = TRUE
- Step 3: Check if pressed

AND Table (Symbol &)

A	B	A&B
0	0	0 = FALSE
0	1	0 = FALSE
1	0	0 = FALSE
1	1	1 = TRUE

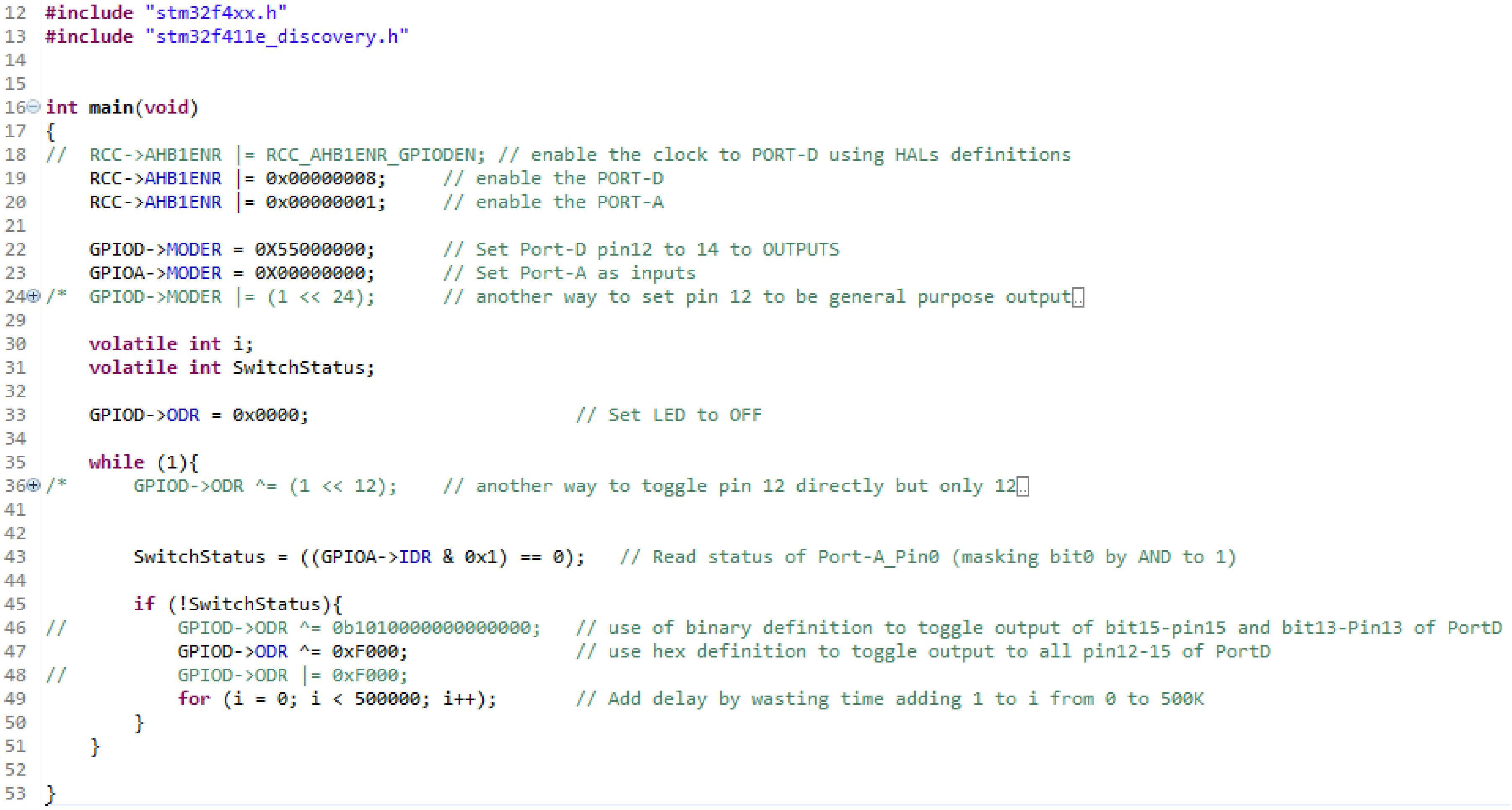
```
  0000 0001
& 0000 0001
-----------
  0000 0001
```

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

SwitchStatus = FALSE (not pressed)

SwitchStatus = TRUE (pressed)

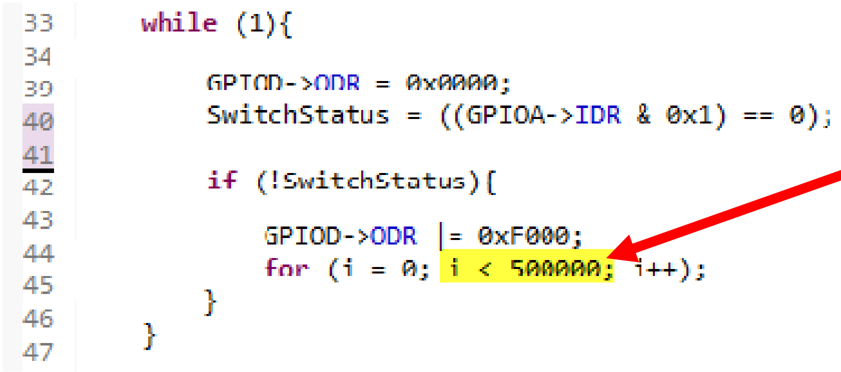
Blinky Update



```c
#include "stm32f4xx.h"
#include "stm32f411e_discovery.h"


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

    GPIOD->MODER = 0X55000000;      // Set Port-D pin12 to 14 to OUTPUTS
    GPIOA->MODER = 0X00000000;      // Set Port-A as inputs
/*  GPIOD->MODER |= (1 << 24);      // another way to set pin 12 to be general purpose output

    volatile int i;
    volatile int SwitchStatus;

    GPIOD->ODR = 0x0000;                        // Set LED to OFF

    while (1){
/*      GPIOD->ODR ^= (1 << 12);    // another way to toggle pin 12 directly but only 12


        SwitchStatus = ((GPIOA->IDR & 0x1) == 0);   // Read status of Port-A_Pin0 (masking bit0 by AND to 1)

        if (!SwitchStatus){
//          GPIOD->ODR ^= 0b1010000000000000;   // use of binary definition to toggle output of bit15-pin15 and bit13-Pin13 of PortD
            GPIOD->ODR ^= 0xF000;               // use hex definition to toggle output to all pin12-15 of PortD
//          GPIOD->ODR |= 0xF000;
            for (i = 0; i < 500000; i++);       // Add delay by wasting time adding 1 to i from 0 to 500K
        }
    }

}
```

De-bounce

- Software add delay to wait for bouncing to dampen

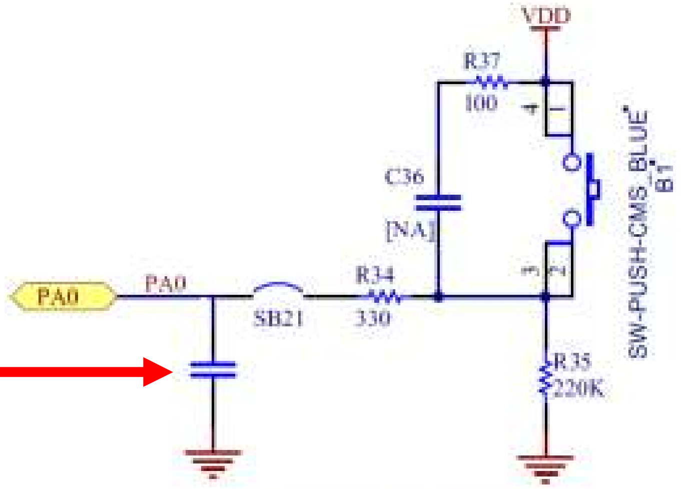


```
while (1){

    GPIOD->ODR = 0x0000;
    SwitchStatus = ((GPIOA->IDR & 0x1) == 0);

    if (!SwitchStatus){

        GPIOD->ODR |= 0xF000;
        for (i = 0; i < 500000; i++);
    }
}
```

Reduce delay value until unexpected behavior

- Hardware fix is to add filter

Capacitor will filter noise along with R34



USER & WAKE-UP Button

Internal Pull Up or Pull down

- All GPIO pins have weak internal pull-up and pull-down resistors
  - GPIOx_PUPDR register can activated or not depending on its value.

Homework?

Extra Activities

Homework

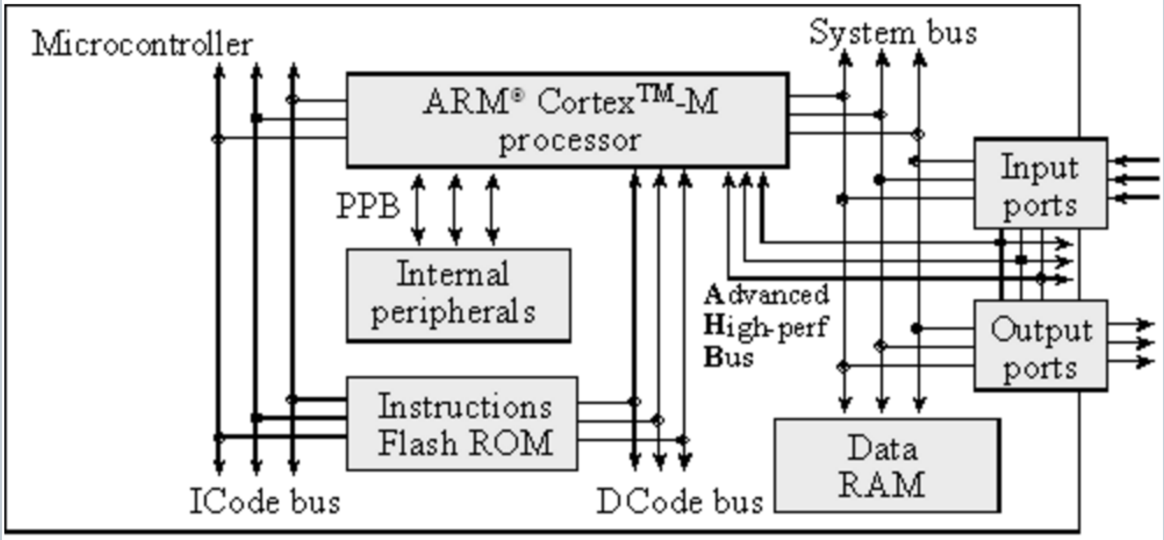
- Create block diagram of design
- Get switch to blink lights at different rates

Back Up Slides

Hardware Reference Material

SIMPLIFIED STM34F411 ARCHITECTURE

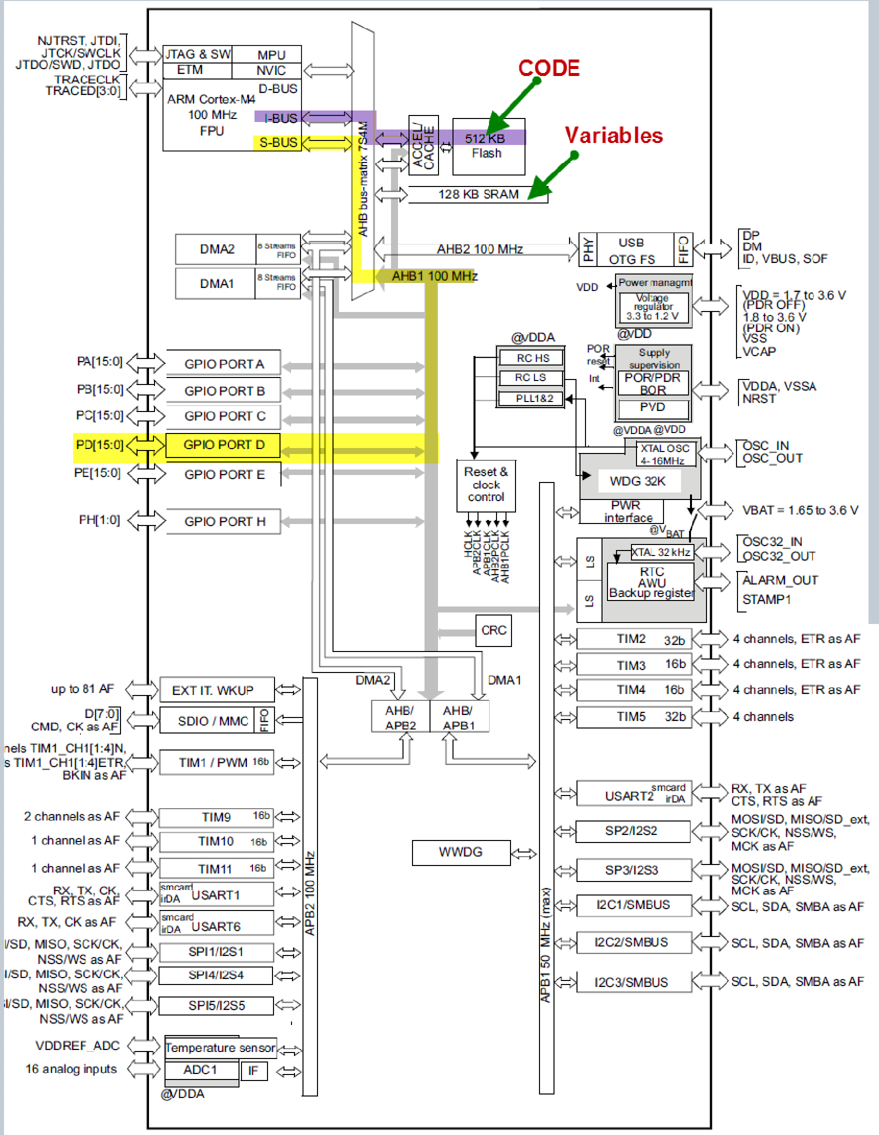
- I-Code Bus use to fetch instructions from Flash ROM
- System Bus: use to work with variables and IO Ports
- D-Code Bus: debug bus
- Adv Hi Bus: Connection to IO ports and dedicated USB ports



STM32F411 BLOCKDIAGRAM

Note the following buses:

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


CODE
Variables
NJTRST, JTDI, JTCK/SWCLK JTDO/SWD, JTDO
TRACECLK TRACED[3:0]
JTAG & SW
MPU
ETM
NVIC
D-BUS
ARM Cortex-M4 100 MHz FPU
I-BUS
S-BUS
AHB bus-matrix 7S4M
ACCEL/ CACHE
512 KB Flash
128 KB SRAM
DMA2
DMA1
8 Streams FIFO
8 Streams FIFO
AHB2 100 MHz
AHB1 100 MHz
PHY
USB OTG FS
FIFO
DP DM ID, VBUS, SOF
VDD
Power managmt
Voltage regulator 3.3 to 1.2 V
@VDD
VDD = 1.7 to 3.6 V (PDR OFF) 1.8 to 3.6 V (PDR ON) VSS VCAP
@VDDA
RC HS
RC LS
PLL1&2
POR reset
Int
Supply supervision
POR/PDR BOR
PVD
VDDA, VSSA NRST
@VDDA @VDD
XTAL OSC 4-16MHz
OSC_IN OSC_OUT
PA[15:0]
GPIO PORT A
PB[15:0]
GPIO PORT B
PC[15:0]
GPIO PORT C
PD[15:0]
GPIO PORT D
PE[15:0]
GPIO PORT E
PH[1:0]
GPIO PORT H
Reset & clock control
WDG 32K
PWR interface
VBAT = 1.65 to 3.6 V
@VBAT
HCLK APB2PCLK APB1PCLK AHB2PCLK AHB1PCLK
LS
LS
XTAL 32 kHz
RTC AWU Backup register
OSC32_IN OSC32_OUT
ALARM_OUT STAMP1
CRC
TIM2 32b
TIM3 16b
TIM4 16b
TIM5 32b
4 channels, ETR as AF
4 channels, ETR as AF
4 channels, ETR as AF
4 channels
up to 81 AF
EXT IT. WKUP
D[7:0] CMD, CK as AF
SDIO / MMC
FIFO
DMA2
DMA1
AHB/ APB2
AHB/ APB1
nels TIM1_CH1[1:4]N, s TIM1_CH1[1:4]ETR, BKIN as AF
TIM1 / PWM 16b
2 channels as AF
TIM9 16b
1 channel as AF
TIM10 16b
1 channel as AF
TIM11 16b
RX, TX, CK, CTS, RTS as AF
smcard irDA USART1
RX, TX, CK as AF
smcard irDA USART6
/SD, MISO, SCK/CK, NSS/WS as AF
SPI1/I2S1
I/SD, MISO, SCK/CK, NSS/WS as AF
SPI4/I2S4
I/SD, MISO, SCK/CK, NSS/WS as AF
SPI5/I2S5
VDDREF_ADC
Temperature sensor
16 analog inputs
ADC1
IF
@VDDA
APB2 100 MHz
WWDG
APB1 50 MHz (max)
USART2 smcard irDA
RX, TX as AF CTS, RTS as AF
SP2/I2S2
MOSI/SD, MISO/SD_ext, SCK/CK, NSS/WS, MCK as AF
SP3/I2S3
MOSI/SD, MISO/SD_ext, SCK/CK, NSS/WS, MCK as AF
I2C1/SMBUS
SCL, SDA, SMBA as AF
I2C2/SMBUS
SCL, SDA, SMBA as AF
I2C3/SMBUS
SCL, SDA, SMBA as AF