



NPTEL ONLINE CERTIFICATION COURSES

Course Name: Embedded System Design with ARM

Faculty Name: Prof. Indranil Sen Gupta

Department : Computer Science and Engineering

Topic

Lecture 7: ARM Instruction Set (Part 1)

CONCEPTS COVERED

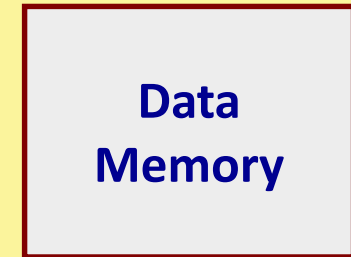
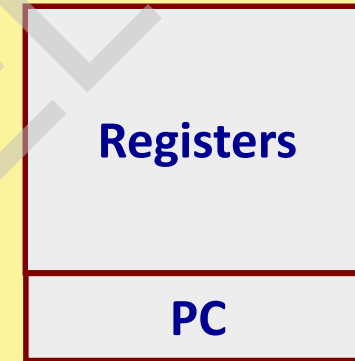
- ❑ Categories of ARM instructions
- ❑ Data processing instructions



The ARM Instruction Set

- ARM instruction can be categorized into three groups:

- a) Data processing instructions
 - Operate on values in registers
- b) Data transfer instructions
 - Move values between registers and memory
- c) Control flow instructions
 - Change the value of the program counter (PC)



(a) Data Processing Instructions

- All operands are 32-bits in size:
 - Either registers
 - Or literals (immediate values) specified as part of the instruction
- The result, if any, is also 32-bit in size and goes into a specified register.
 - One exception: long multiply, that generates 64-bit results.
- All operand and result registers are independently specified as part of the instruction.

- **Arithmetic instructions:**

ADD	r0, r1, r2	; r0 = r1 + r2
ADC	r0, r1, r2	; r0 = r1 + r2 + C (C is carry bit)
SUB	r0, r1, r2	; r0 = r1 - r2
SBC	r0, r1, r2	; r0 = r1 - r2 + C - 1
RSB	r0, r1, r2	; r0 = r2 - r1
RSC	r0, r1, r2	; r0 = r2 - r1 + C - 1

- All operations can be viewed as either unsigned or 2's complement signed.
 - Means the same thing.

- **Bit-wise logical instructions:**

AND r0, r1, r2 ; r0 = r1 and r2

ORR r0, r1, r2 ; r0 = r1 or r2

EOR r0, r1, r2 ; r0 = r1 xor r2

BIC r0, r1, r2 ; r0 = r1 and not r2

- BIC is the acronym for “*bit clear*”
 - Each 1-bit in r2 clears the corresponding bit in r1.

- **Register-register move instructions:**

`MOV     r0, r2                ; r0 = r2`

`MVN     r0, r2                ; r0 = not r2`

- MVN is the acronym for “*move negated*”
 - Each 1-bit in r2 clears the corresponding bit in r0.
- In the instruction encoding, the first operand r1 is not specified, as these are unary operations.

- **Comparison instructions:**

CMP	r1,r2	; set cc on (r1 - r2)
CMN	r1,r2	; set cc on (r1 + r2)
TST	r1,r2	; set cc on (r1 and r2)
TEQ	r1,r2	; set cc on (r1 xor r2)

- All these instructions affect the condition codes (N, Z, C, V) in the current program status register (CPSR).
 - These instructions do not produce result in any register (r0).

- **Specifying immediate operands:**

ADD	r1, r2, #2	; r1 = r2 + 2
SUB	r3, r3, #1	; r3 = r3 - 1
AND	r6, r4, #&0f	; r6 = r4[3:0]

- **Notations:**

- # indicates immediate value
- & indicates hexadecimal notation

- **Allowed immediate values:**

- 0 to 255 (8 bits), rotated by any number of bit positions that is multiple of 2.

- **Shifted register operands:**

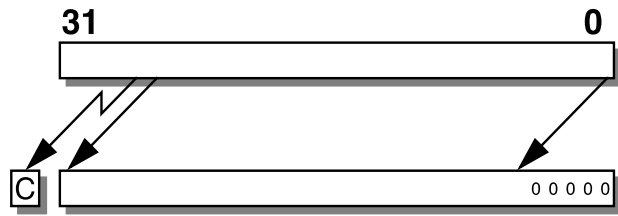
- The second source operand may be shifted either by a constant number of bit positions, or by a register-specified number of bit positions.

`ADD r1, r2, r3, LSL #3 ; r1 = r2 + (r3 << 3)`

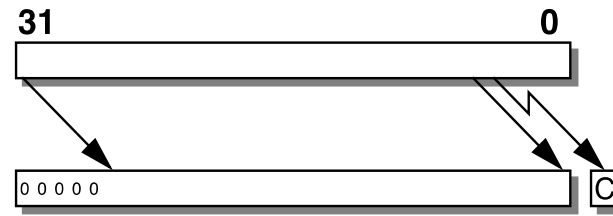
`ADD r1, r2, r3, LSL r5 ; r1 = r2 + (r3 << r5)`

- Various shift and rotate options:

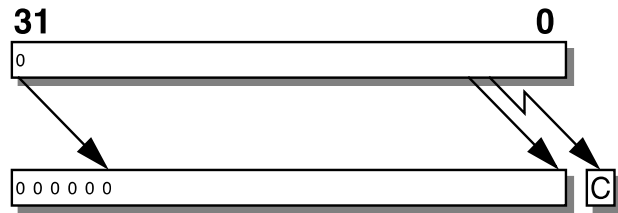
- **LSL**: logical shift left
- **LSR**: logical shift right
- **ROR**: rotate right
- **RRX**: rotate right extended by 1 bit
- **ASL**: arithmetic shift left
- **ASR**: arithmetic shift right



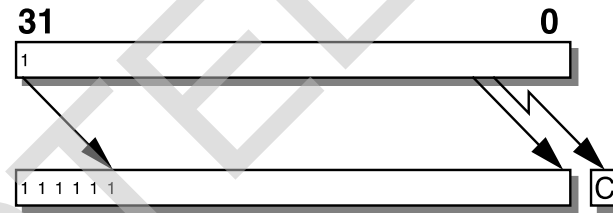
LSL # 5



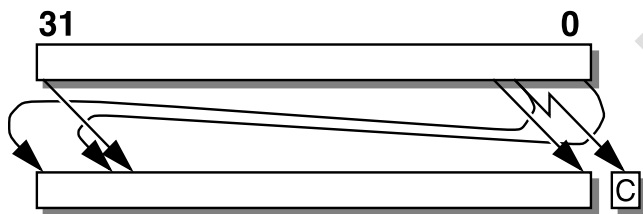
LSR # 5



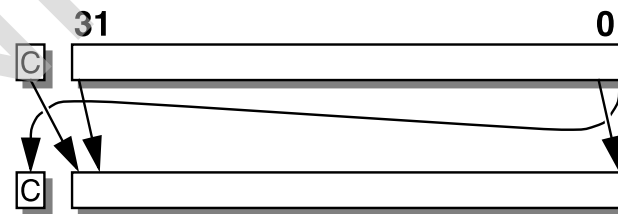
ASR # 5 – positive operand



ASR # 5 – negative operand



ROR # 5



RRX

- **Multiplication instruction**

MUL **r1, r2, r3** ; **r1 = (r2 x r3) [31:0]**

- Only the least significant 32-bits are returned.
- Immediate operands are not supported.

- **Multiply-accumulate instruction:**

MLA **r1, r2, r3, r4** ; **r1 = (r2 x r3 + r4) [31:0]**

- Required in digital signal processing (DSP) applications.
- Multiplication with 64-bit results is also supported.



NPTEL ONLINE CERTIFICATION COURSES

**Thank
you!**



NPTEL ONLINE CERTIFICATION COURSES

Course Name: Embedded System Design with ARM

Faculty Name: Prof. Indranil Sen Gupta

Department : Computer Science and Engineering

Topic

Lecture 8: ARM Instruction Set (Part 2)

CONCEPTS COVERED

- ☐ Data transfer instructions
- ☐ Single register transfer instructions
- ☐ Multiple register transfer instructions
- ☐ Memory-mapped I/O in ARM



(b) Data Transfer Instructions

- ARM instruction set supports three types of data transfers:
 - a) Single register loads and stores
 - Flexible, supports byte, half-word and word transfers
 - b) Multiple register loads and stores
 - Less flexible, multiple words, higher transfer rate
 - c) Single register-memory swap
 - Mainly for system use (for implementing locks)

- All ARM data transfer instructions use *register indirect addressing*.
 - Before any data transfer, some register must be initialized with a memory address.

```
ADRL    r1,Table    ; r1 = memory address of Table
```

- Example:

```
LDR     r0,[r1]      ; r0 = mem[r1]
```

```
STR     r0,[r1]      ; mem[r1] = r0
```

- **Single register loads and stores**

- The simplest form uses register indirect without any offset:

```
LDR    r0, [r1]      ; r0 = mem[r1]
```

```
STR    r0, [r1]      ; mem[r1] = r0
```

- An alternate form uses register indirect with offset (limited to 4 Kbytes):

```
LDR    r0, [r1, #4]   ; r0 = mem[r1+4]
```

```
STR    r0, [r1, #12]  ; mem[r1+12] = r0
```

- We can also use auto-indexing in addition:

```
LDR    r0, [r1, #4]!   ; r0 = mem[r1+4], r1 = r1 + 4
```

```
STR    r0, [r1, #12]!  ; mem[r1+12] = r0, r1 = r1 + 4
```

- We can use post indexing:

```
LDR    r0,[r1],#4    ; r0 = mem[r1],    r1 = r1 + 4
STR    r0,[r1],#12   ; mem[r1] = r0,    r1 = r1 + 12
```

- We can specify a byte or half-word to be transferred:

```
LDRB   r0,[r1]       ; r0 = mem8[r1]
STRB   r0,[r1]       ; mem8[r1] = r0
LDRSH  r0,[r1]       ; r0 = mem16[r1]
STRSH  r0,[r1]       ; mem16[r1] = r0
```

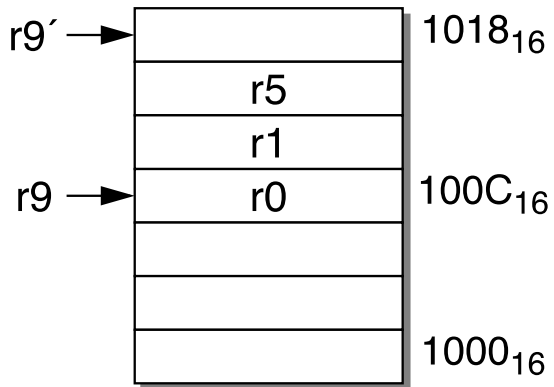
- **Multiple register loads and stores**

- ARM supports instructions that transfer between several registers and memory.
- Example:

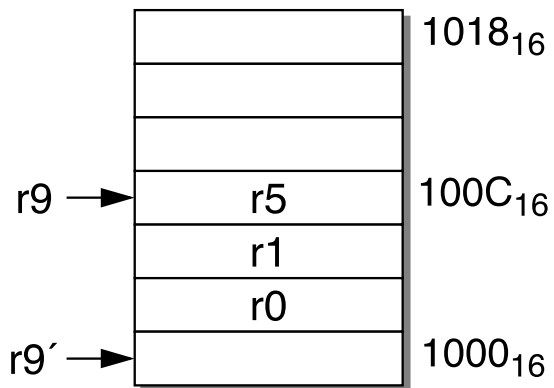
```
LDMIA r1, {r3, r5, r6}      ; r3 = mem[r1]
                             ; r5 = mem[r1+4]
                             ; r6 = mem[r1+8]
```

- For **LDMIB**, the addresses will be **r1+4, r1+8, and r1+12**.
- The list of destination registers may contain any or all of r0 to r15.
- Block copy addressing
 - Supported with addresses that can *increment (I)* or *decrement (D)*, *before (B)* or *after (A)* each transfer.

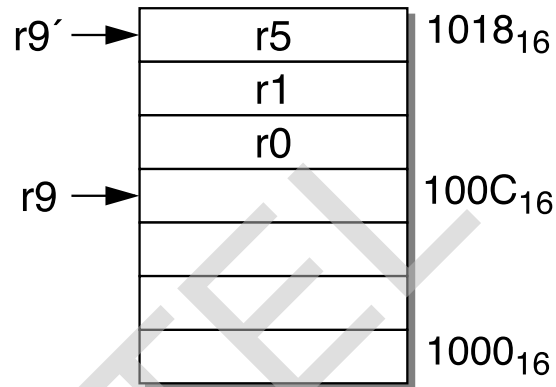
- Examples of addressing modes in multiple-register transfer:



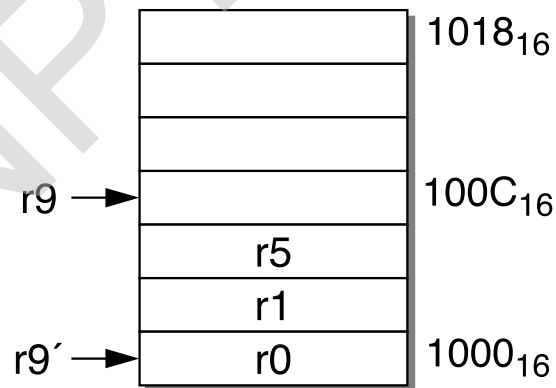
STMIA r9!, {r0,r1,r5}



STMDA r9!, {r0,r1,r5}



STMIB r9!, {r0,r1,r5}



STMDB r9!, {r0,r1,r5}

LDMIA, STMIA

- Increment after

LDMIB and STMIB

- Increment before

LMDA, STMDA

- Decrement after

LMDDB, STMDDB

- Decrement before

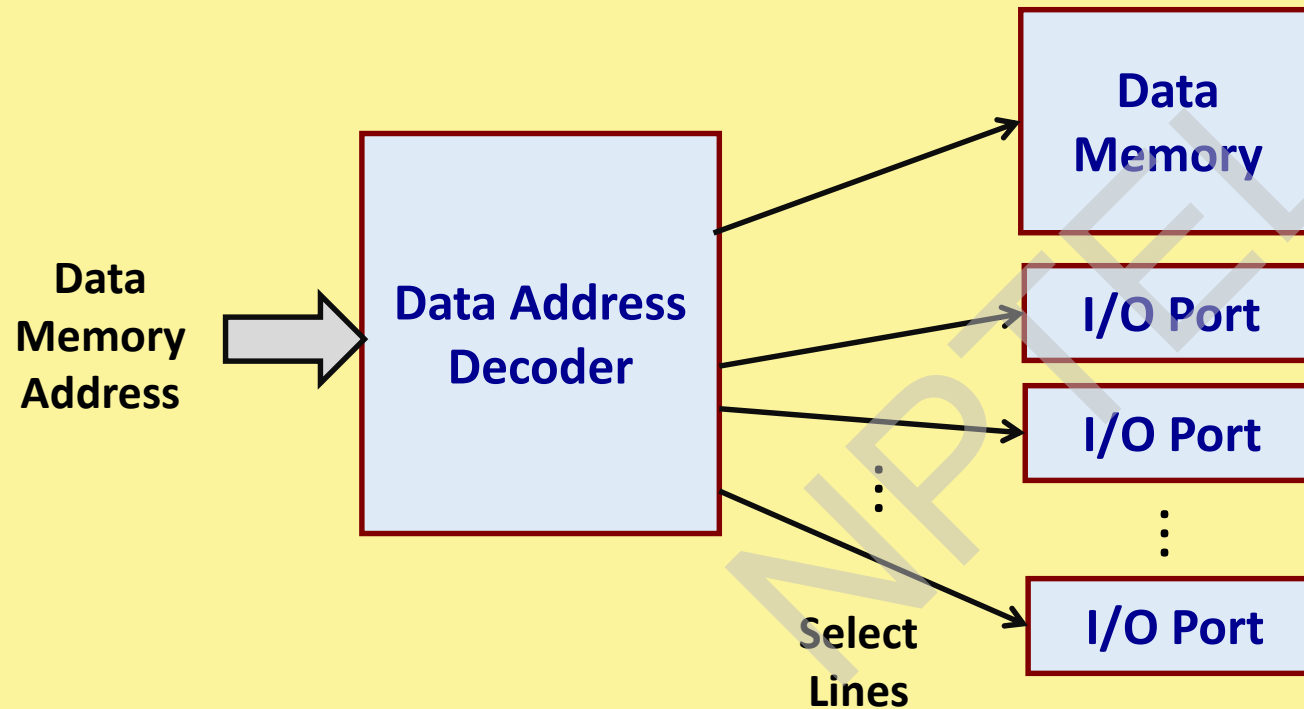
- Point to note:
 - ARM does not support any hardware stack.
 - Software stack can be implemented using the **LDM** and **STM** family of instructions.

An Example

- Copy a block of memory (128 bytes aligned).
 - r9: address of the source
 - r10: address of the destination
 - r11: end address of the source

```
Loop:  LDMIA  r9!, {r0-r7}  
       STMIA  r10!, {r0-r7}  
       CMP   r9, r11  
       BNE   Loop
```

Memory Mapped I/O in ARM



- No separate instructions for input/output.
- The I/O ports are treated as data memory locations.
 - Each with a unique (memory) address.
- Data input is done using the LDR instruction.
- Data output is done with the STR instruction.



NPTEL ONLINE CERTIFICATION COURSES

**Thank
you!**



NPTEL ONLINE CERTIFICATION COURSES

Course Name: Embedded System Design with ARM

Faculty Name: Prof. Indranil Sen Gupta

Department : Computer Science and Engineering

Topic

Lecture 9: ARM Instruction Set (Part 3)

CONCEPTS COVERED

- ☐ Control flow instructions
- ☐ Branch and link instruction
- ☐ Conditional execution instructions



(c) Control Flow Instructions

- These instructions change the order of instruction execution.
 - Normal flow is sequential execution, where PC is incremented by 4 after executing every instruction.
- Types of conditional flow instructions:
 - Unconditional branch
 - Conditional branch
 - Branch and Link
 - Conditional execution

- **Unconditional branch instruction:**

	B	Target
	...	
	...	
Target	...	

- **Conditional branch instruction:**

	MOV	r2 , #0
LOOP	...	
	...	
	ADD	r2 , r2 , #1
	CMP	r0 , #20
	BNE	LOOP
	...	

- Branch conditions that are supported:

- | | |
|------------|--------------------------------|
| • B, BAL | Unconditional branch |
| • BEQ, BNE | Equal or not equal to zero |
| • BPL, PMI | Result positive or negative |
| • BCC, BCS | Carry set or clear |
| • BVC, BVS | Overflow set or clear |
| • BGT, BGE | Greater than, greater or equal |
| • BLT, BLE | Less than, less or equal |

• Branch and link instruction

- Used for calling subroutines in ARM.
- The return address is saved in register r14 (called *link register*).
- To return from the subroutine, we have to jump back to the address stored in r14.

```
BL      MYSUB      ; Branch to subroutine
...
...              ; Return here
MYSUB   ...         ; Subroutine starts here
...
MOV     pc,r14     ; Return
```

Nested subroutine calls cannot be used in this way.

- We can use software stack to save/restore the return address and registers.
- An example showing nested subroutine calls and return.

```

        BL      MYSUB1
        ...

MYSUB1  STMFD   r13!, {r0-r2,r14}
        BL      MYSUB2
        ...
        LDMFD   r13!, {r0-r2,pc}

MYSUB2  ...
        ...
        MOV     pc,r14

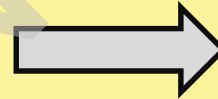
```

Nested subroutine calls can be used.

• Conditional execution

- A unique feature of the ARM instruction set.
- All instructions can be made conditional, i.e. will get executed only when a specified condition is true.
- Helps in removing many short branch instructions (improves performance and code density)
- An example: **if (r2 != 10) r5 = r5 +10 - r3**

```
CMP    r2, #10
BEQ    SKIP
ADD    r5, r5, r2
SUB    r5, r5, r3
SKIP   ...
```



```
CMP    r2, #10
ADDNE  r5, r5, r2
SUBNE  r5, r5, r3
```

- Various instruction postfix supported for conditional execution:

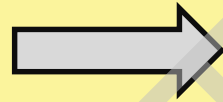
Postfix	Condition
CS	Carry set
EQ	Equal (zero set)
VS	Overflow set
GT	Greater than
GE	Greater than or equal
PL	Plus (positive)
HI	Higher than
HS	Higher or same (i.e. CS)

Postfix	Condition
CC	Carry clear
NE	Not equal (zero clear)
VC	Overflow clear
LT	Less than
LE	Less than or equal
MI	Minus (negative)
LO	Lower than (i.e. CC)
LS	Lower or same

- Another example:

`if ((r1 == r3) && (r5 == r6)) r7 = r7 + 10`

```
CMP    r1,r3
BNE    SKIP
CMP    r5,r6
BNE    SKIP
ADD    r7,r7,#10
SKIP   ...
```



```
CMP    r1,r3
CMPEQ  r5,r6
ADDEQ  r7,r7,#10
```

• Supervisor calls

- The *Software Interrupt Instruction (SWI)* is used to enter Supervisor Mode, usually to request a particular supervisor function (e.g. input or output).
- The CPSR is saved into the Supervisory Mode SPSR, and execution branches to the SWI vector.
- The SWI handler reads the opcode to extract the SWI function number.



NPTEL ONLINE CERTIFICATION COURSES

Thank
you!



NPTEL ONLINE CERTIFICATION COURSES

Course Name: Embedded System Design with ARM

Faculty Name: Prof. Indranil Sen Gupta

Department : Computer Science and Engineering

Topic

Lecture 10: About the STM32F401 Nucleo Board

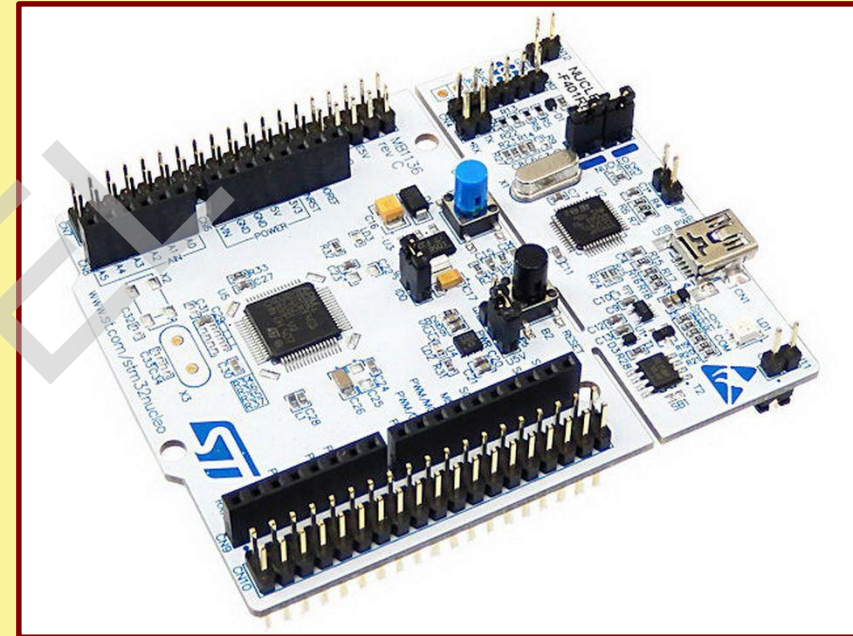
CONCEPTS COVERED

- ☐ About the STM32F01 Nucleo board
- ☐ Board features
- ☐ Connector details



About the STM32F401 Nucleo Board

- Developed by ST Microelectronics.
- CPU: ARM Cortex[®] M4
- 512 KB Programmable Flash Memory
- 96 KB SRAM
- USB 2.0: Type A to Mini B
- Mbed Enabled (mbed.org)
- Supports a wide choice of Integrated Development Environments (IDEs) including IAR[™], ARM[®] Keil[®], GCC-based IDEs.



Some Specific Details

- Contains STM32F401RET6 microcontroller with ARM Cortex-M4 CPU with FPU at 84 MHz clock, 512 KB of flash and 96 KB of SRAM with a variety of peripherals.
- Two types of extension connectors are available:
 - Arduino UNO Revision 3 connectivity.
 - STM Morpho extension pins for full access to all STM32F401RET6 I/Os.
- On-board ST-LINK/V2.1 interfacing with the PC, which provides:
 - Debug and programming port, to use the board with standard programmers.
 - Virtual Com port to send back traces to the PC.
 - Mass storage (USB Disk Drive) for drag-n-drop programming.

- Input/Output
 - 50 general-purpose I/O pins with external interrupt capability.
 - 12-bit ADC with 16 channels.
 - 7 general-purpose timers.
 - 2 watchdog timers.
 - USB 2.0 interface.
 - And several others ...
- User LED (LD2)
- Two push buttons: USER and RESET.

User Button

Green LED
(test)

Red LED
(power)

Arduino
Connector

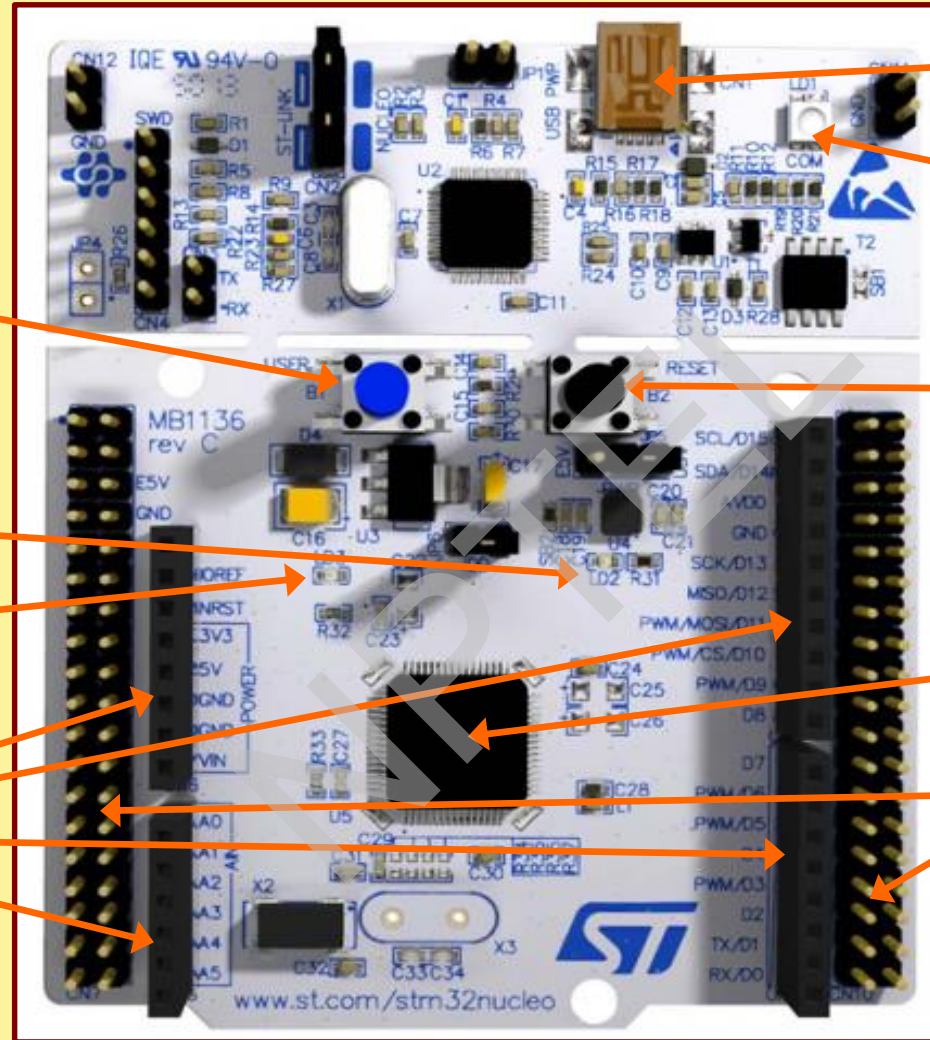
USB: Type-A
to Mini-B

Red/Green
LED (com)

Reset Button

STM32
Microcontroller

Port/Signal
Connector



Some Color Conventions Followed

Labels usable in code

PX_Y

MCU pin without conflict

PX_Y

MCU pin connected to other components

See **PeripheralPins.c** (link below) for more information

XXX

Arduino connector names (A0, D1, ...)

XXX

LEDs and Buttons (LED_1, USER_BUTTON, ...)

Labels not usable in code (for information only)

XXX

Serial pins (USART/UART)

XXX

SPI pins

XXX

I2C pins

XXX

PWMOut pins (TIMER n/c[N])

n = Timer number c = Channel

N = Inverted channel

XXX

AnalogIn (ADC) and AnalogOut pins (DAC)

XXX

CAN pins

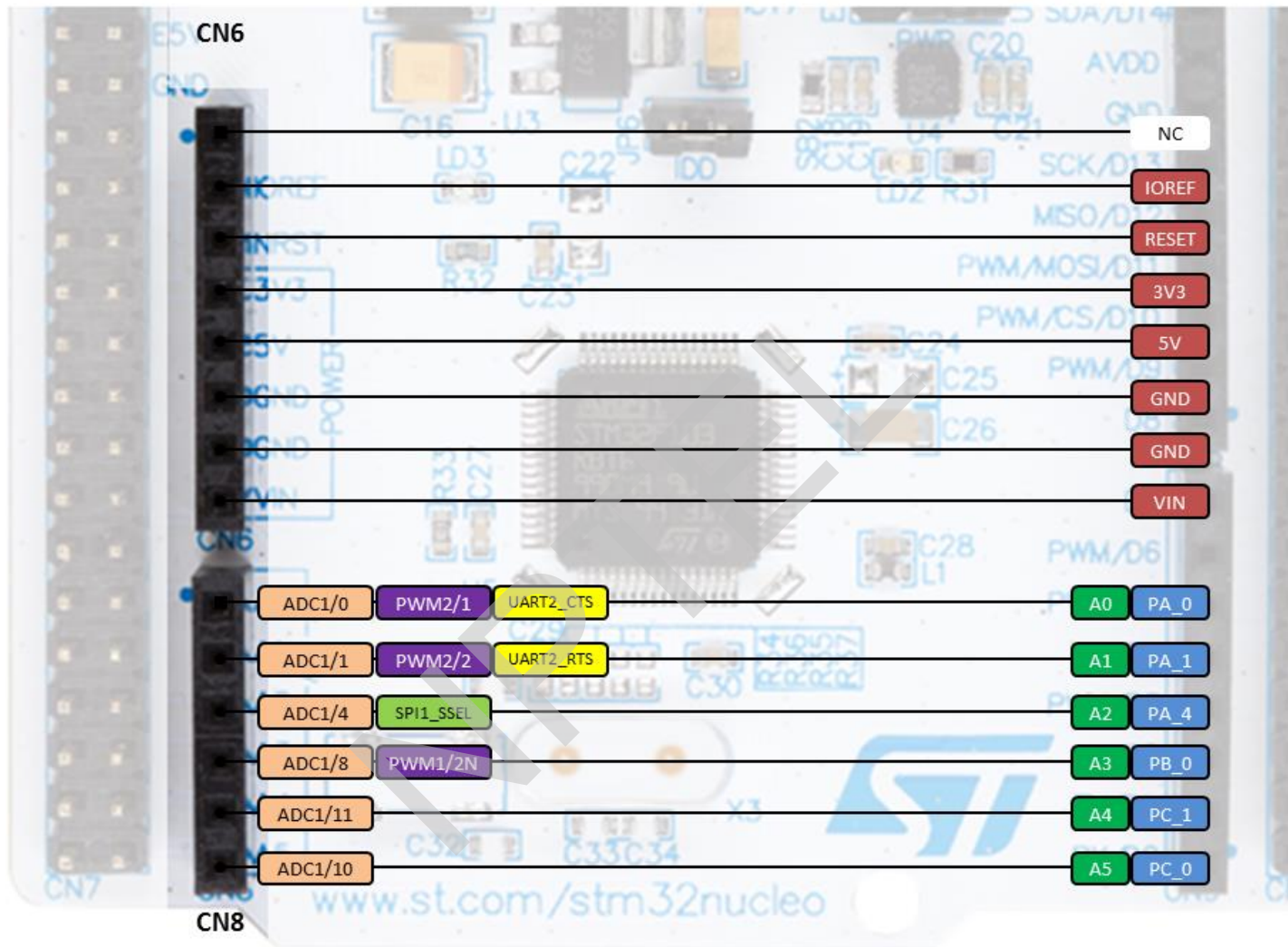
XXX

Power and control pins (3V3, GND, RESET, ...)

The Nomenclatures

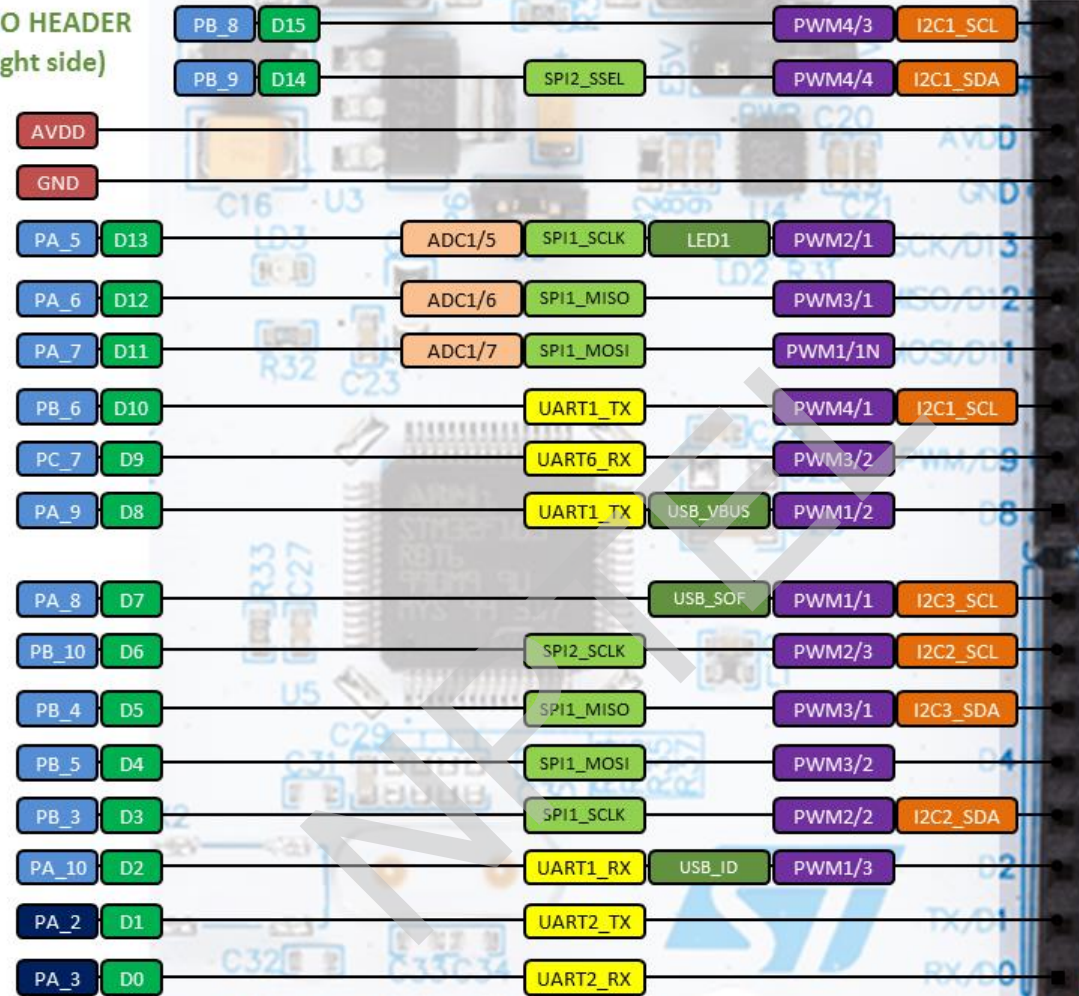
- SPI: Serial Peripheral Interface
- I2C: Inter-Integrated Circuit
 - Serial communication bus, widely used for attaching lower-speed peripheral ICs to microcontrollers over short distance.
- PWM: Pulse Width Modulation
- CAN: Controller Area Network
 - A robust vehicle bus standard, designed to allow microcontrollers and devices to communicate with each other without a host computer.

Arduino Header CN6, CN8



Arduino Header CN9

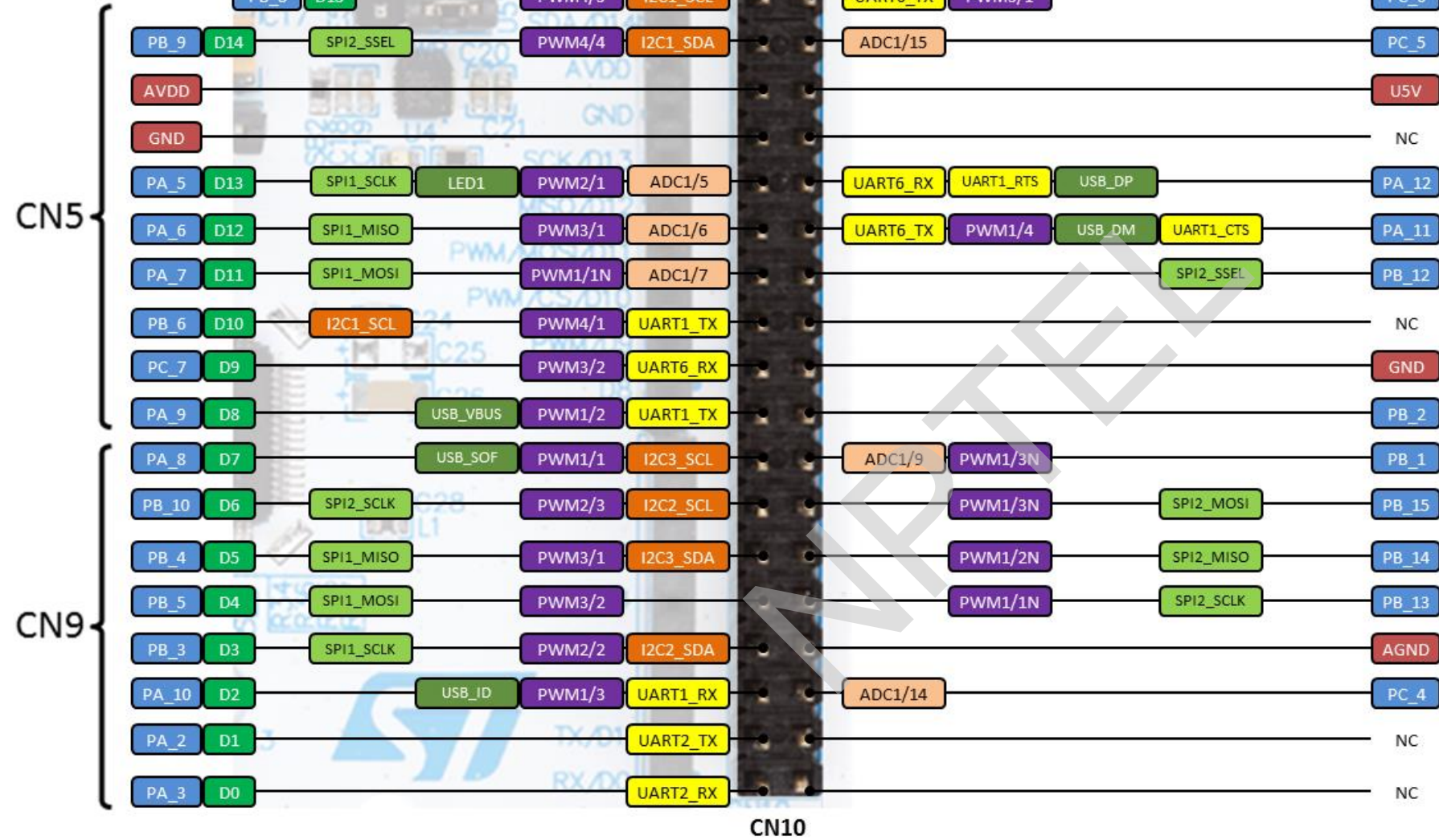
NUCLEO-F401RE
ARDUINO HEADER
(top right side)



CN9

CN7 HEADER
(top left side)

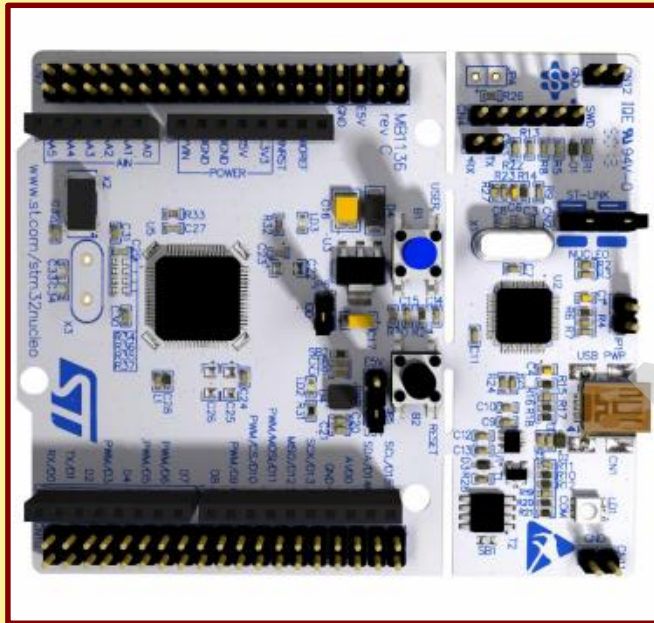




CN10 Header

Connecting the Board to the PC/Laptop

- Using the USB cable.





NPTEL ONLINE CERTIFICATION COURSES

**Thank
you!**



NPTEL ONLINE CERTIFICATION COURSES

Course Name: Embedded System Design with ARM

Faculty Name: Prof. Indranil Sen Gupta

Department : Computer Science and Engineering

Topic

Lecture 11: PWM AND INTERRUPT ON STM32F401

CONCEPTS COVERED

- ☐ Pulse width modulation (PWM)
- ☐ Using PWM on mbed platform
- ☐ Using interrupts on STM32 board



Pulse Width Modulation



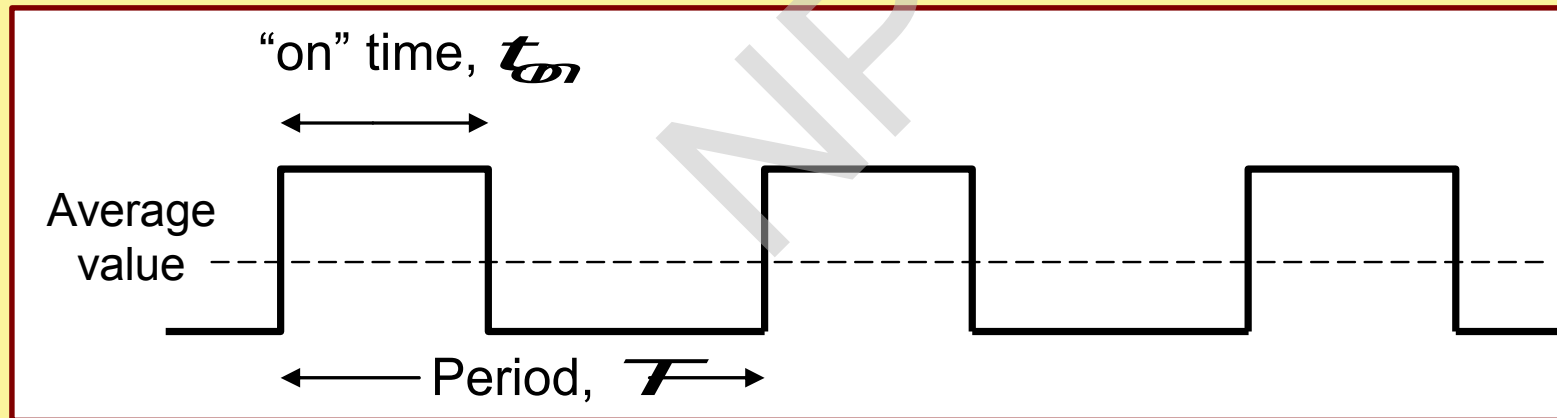
What is Pulse Width Modulation (PWM)?

- PWM is a simple method of using a rectangular digital waveform to control an analog variable.
 - The on-off behavior changes the average power of the signal.
 - Output signal alternates between ON and OFF with a specific time period.
- PWM control is used in a variety of applications, ranging from communications to automatic control.
- It can also be used to encode information for data transmission.

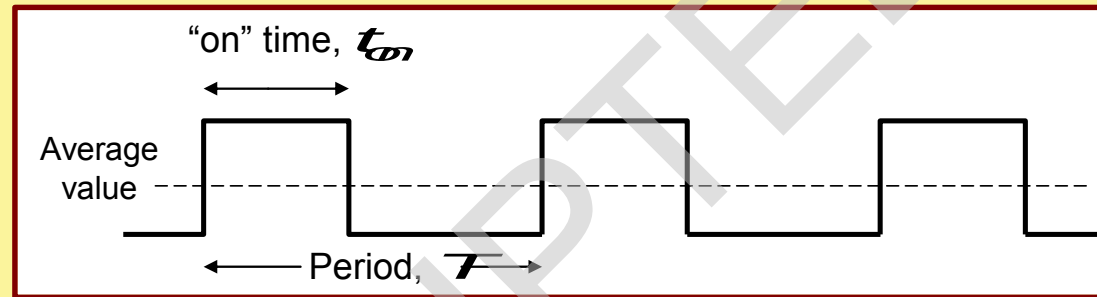
How it works?

- The period is normally kept constant, and the pulse width (or ON time) is varied.
- **Duty Cycle:** It is defined as the proportion of time the pulse is ON, expressed as a percentage.

$$\text{Duty Cycle} = (\text{pulse ON time}) / (\text{pulse period}) * 100\% = t_{on} / T * 100\%$$



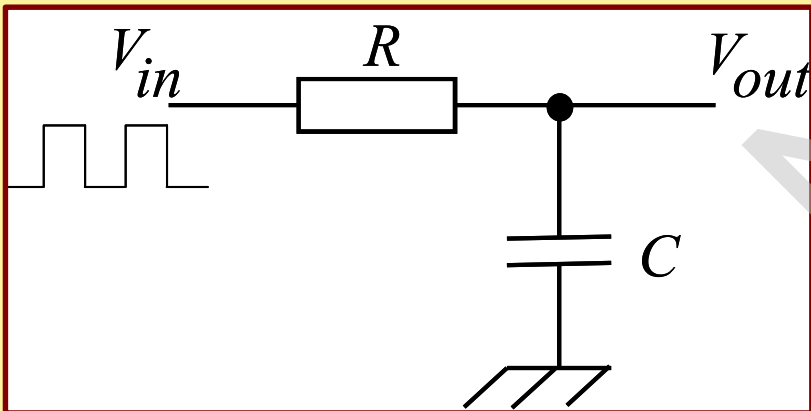
- Whatever duty cycle a PWM has, there is an *average value*, as indicated by the dotted line.
 - If the ON time is small, the average value is low; if it is large, the average value is high.
 - By controlling the duty cycle, we can control the average value.



- Average value of the signal $= \frac{1}{T} \int_0^T f(t) dt = t_{on} \cdot V_H + (1 - t_{on}) \cdot V_L$
- In general, V_L is taken as 0V for ease of calculation.
 - Average value becomes $t_{on} \cdot V_H$

How to Extract the Average Value?

- The average value can be extracted from the PWM stream using a low-pass filter.
- If the PWM frequency and the values of R and C are appropriately chosen, V_{out} becomes an analog output.
 - Can be used in place of a digital-to-analog converter.



In practice, the filter is not always required.

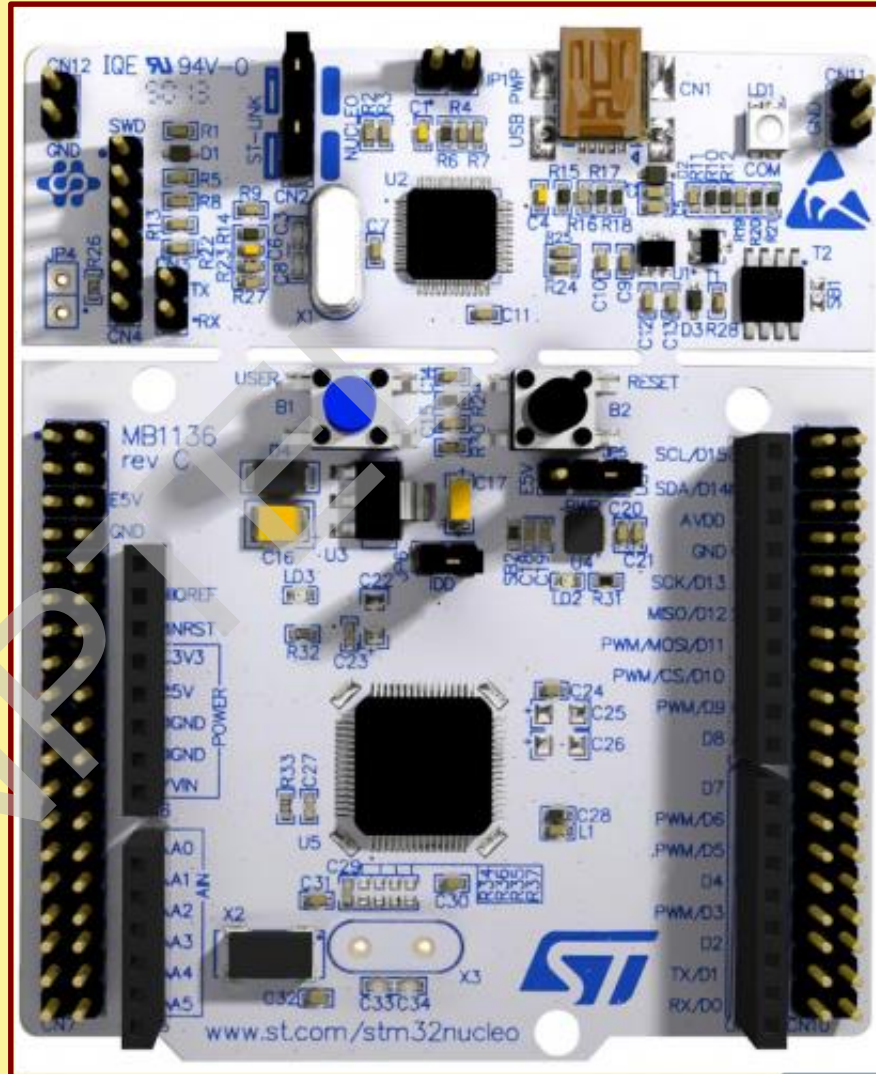
Many physical systems have response characteristics that act like low-pass filters.

Some Typical Applications

1. Control of DC motor.
 - The voltage supplied to the motor is proportional to the duty cycle.
2. Controlling the brightness of LED.
 - The duty cycle of the voltage source determines the brightness.
3. Control the temperature (heater).
 - Switch ON and OFF the heater with an appropriate duty cycle.
4. Many more ...

PWM on the STM32F401

- The *PwmOut interface* is used to control the frequency and duty cycle of a digital pulse train.
- The *Arduino Interface* supports up to 6 PWM outputs, although these *PwmOut ports* share the same period timer.



PWM Library Functions Available in mbed.h

Function	
PwmOut	Create a PwmOut connected to the specified pin
write	Set the output duty cycle, specified as normalized float (0.0 to 1.0)
read	Return the current output duty cycle setting (0.0 to 1.0)
period period_ms period_us	Set the PWM period, specified in seconds (float), milli-seconds (int) or micro-seconds (int), keeping the duty cycle the same.
pulsewidth pulsewidth_ms pulsewidth_us	Set the PWM pulsewidth, specified in seconds (float), milli-seconds (int) or micro-seconds (int), keeping the period the same.
operator =	Shorthand for write()
operator float()	Shorthand for read()

Example 1: 100 Hz pulse with 50% duty cycle

```
#include "mbed.h"
PwmOut PWM1(p21); // Pin 21
                    // (PWM/D9)

int main() {
    PWM1.period (0.01);
    PWM1.write (0.5);
}
```

```
#include "mbed.h"
PwmOut PWM1(p21); // Pin 21
                    //
                    (PWM/D9)

int main() {
    PWM1.period (0.01);
    PWM1 = 0.5;
}
```

Example 2: 1 KHz pulse with 80% duty cycle

```
#include "mbed.h"
PwmOut PWM1(p21);    // Pin 21
                      // (PWM/D9)

int main() {
    PWM1.period_ms(1);
    PWM1 = 0.8;
}
```

Interrupt on the STM32 Board



Using Interrupts on STM32 Board

- STM32F4 External Interrupts
 - Each STM32F4 device has 23 external interrupts or event sources, which are split into two sections.
- In the first-section, we can use the general-purpose I/O pins as interrupt sources.
 - We can use these interrupts in our experiments.
- The second section is for specific device interrupts, like *USB interrupt*, *real-time clock (RTC) interrupt*, *timer interrupt*, *Ethernet interrupt*, etc.

General-Purpose I/O (GPIO) Pin Interrupt Lines

- The 16 interrupt lines are referred to as Line0 to Line15.
 - Three 16-bit GPIO ports: PA, PB and PC.
 - Line0 is connected to PA0, PB0, and PC0.
 - Line1 is connected to PA1, PB1, and PC1.
 - Line1 is connected to PA1, PB1, and PC1, and so on.
- We cannot use PAx, PBx, PCx as interrupt sources together for the same value of x.

InterruptIn Library Functions Available in mbed.h

Function	
InterruptIn	Create an InterruptIn connected to the specified pin
int read()	Read the input, represented as 0 or 1 (int)
operator int()	Shorthand for read()
void rise (* func)	Attach a function to call when a rising edge occurs on the input
void enable_irq()	Enable IRQ
void disable_irq()	Disable IRQ

Example 3: Toggle a LED with every key press

```
#include "mbed.h"
InterruptIn switch (D3);
DigitalOut led (D2);

void toggle() {                      // Interrupt handler
    led = !led;
}

int main() {
    switch.rise (&toggle); // Install interrupt handler
    while (1) {
    }
}
```

- Switch connected to the interrupt input D3.
- A LED connected to output pin D2.



NPTEL ONLINE CERTIFICATION COURSES

Thank
you!