



NPTEL ONLINE CERTIFICATION COURSES

Course Name: Embedded System Design with ARM

Faculty Name: Prof. Indranil Sen Gupta

Department : Computer Science and Engineering

Topic

Lecture 30: Design of Control Systems

CONCEPTS COVERED

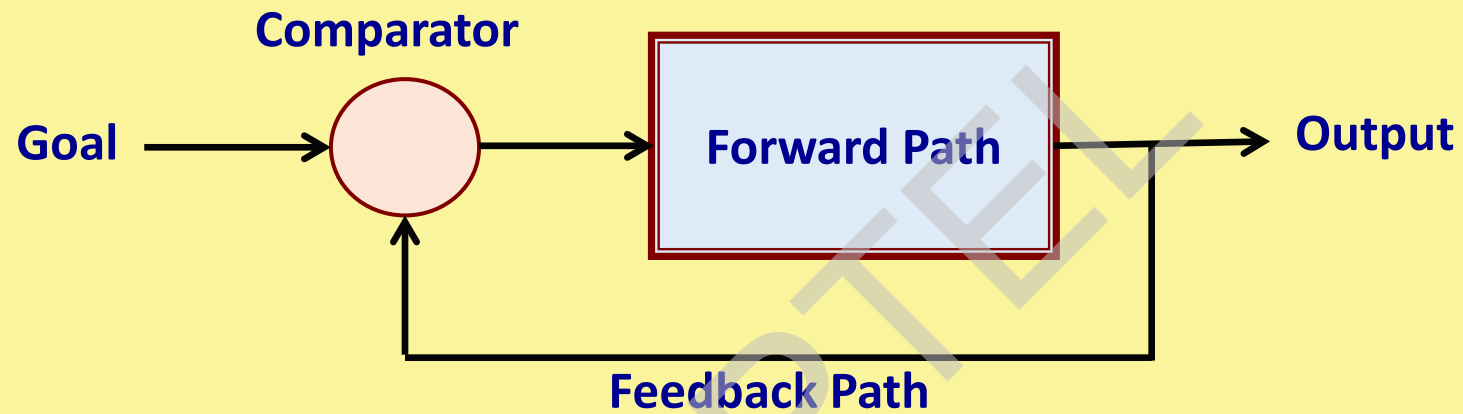
- ☐ Closed loop control system
- ☐ ON-OFF controller
- ☐ PID controller



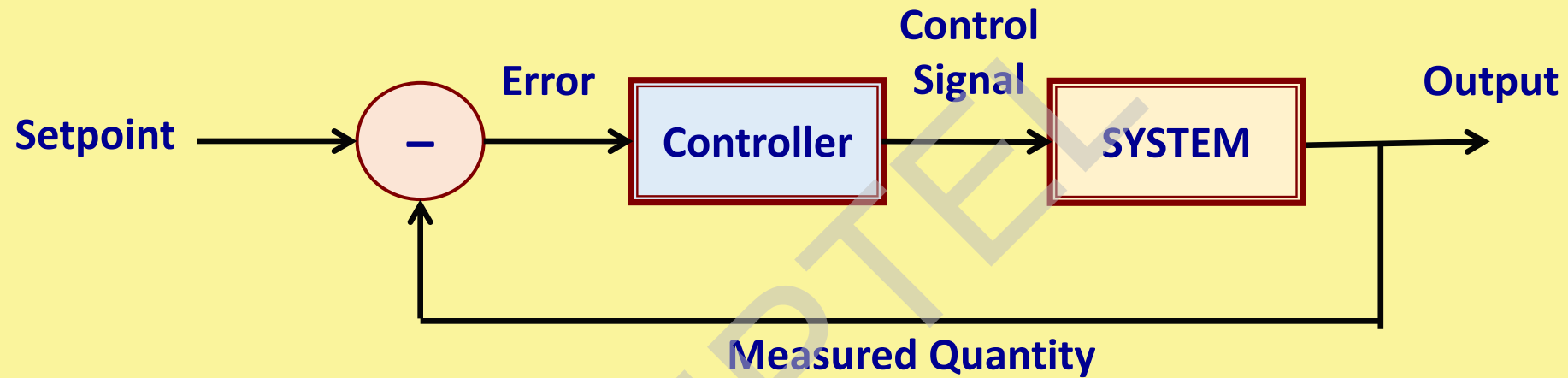
Introduction

- In many embedded system applications, we have to sense the value of some external parameter and take corrective actions to maintain it within acceptable limits.
 - Temperature of an oven, speed of a motor, etc.
 - Essentially a control system.
- Two types of control systems:
 - a) **Open loop**: where there is no feedback with respect to the measured value.
 - b) **Closed loop**: more sophisticated, corrective actions applied with feedback.

Closed Loop Control System



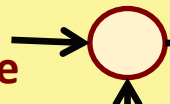
Role of the Controller



Examples



Required temperature

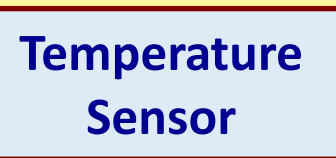


Forward Path



Heater

Actual temperature

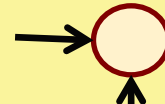


Temperature Sensor

Feedback Path



Required speed



Forward Path



Motor

Actual speed



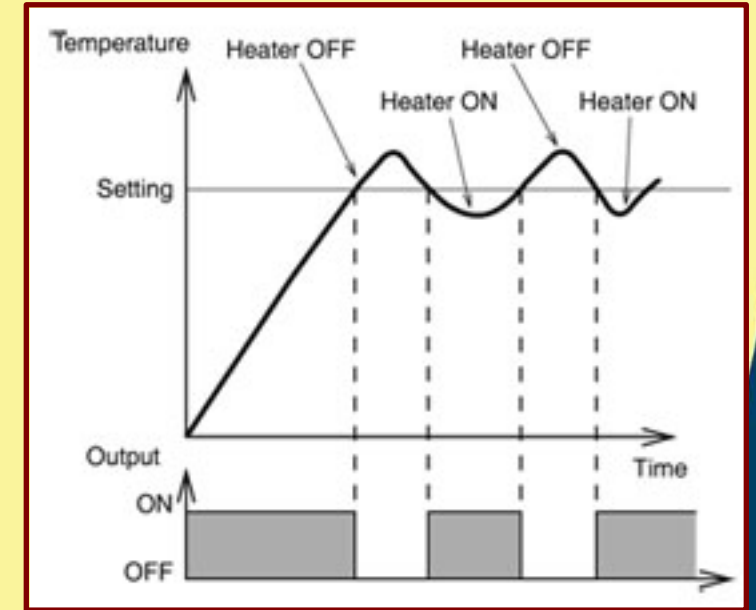
Speed Sensor

Feedback Path

Choice of Controller Type

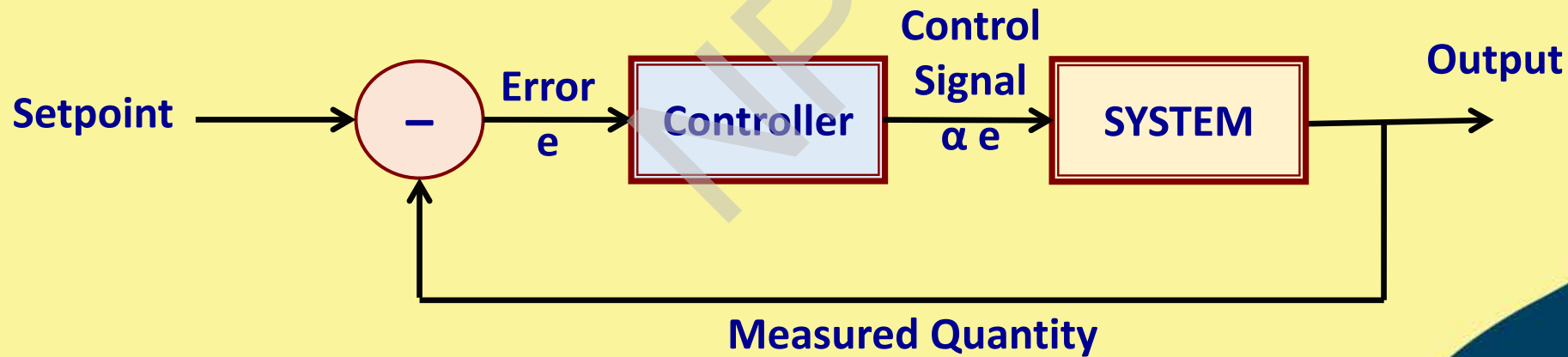
a) ON-OFF Controller:

- An ON-OFF controller is the simplest type of controller, where the control signal has only two levels.
 - For example, in a heater, if the actual temperature is less than the set temperature, the heater is turned ON; otherwise, it is turned OFF.
- This type of controller is inexpensive, but often causes oscillation in the output variable.
- It is often used in simple appliances such as oven, iron, refrigerators, etc. where oscillations can be tolerated.



b) Proportional (P) Controller:

- The control signal is set to be *proportional to the difference* between the actual output and the setpoint (i.e. the error).
- Need to find out the value of constant of proportionality.
 - Tuning the controller is a hard job.
- Typically, a P controller decreases response time, but increases overshoot.



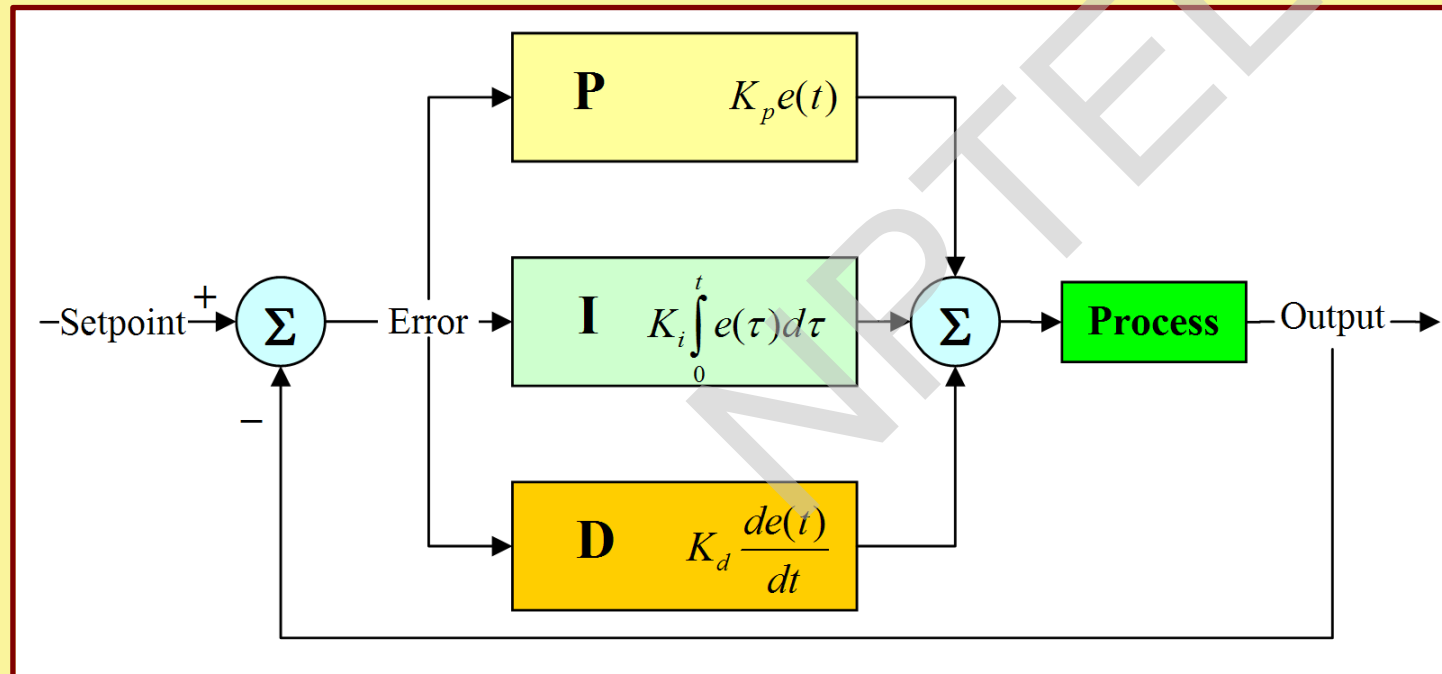
c) Proportional-Derivative (PD) Controller:

- To reduce the overshoot, we can take into account how fast we are approaching the setpoint.
 - We add D control in addition to P control.
- D is estimated as the *difference* between the current measure and the previous measure.
- PD controllers are slower than P controllers, but generates less oscillation, and smaller overshoot/ripple.
- **Drawback:**
 - Output is close to the setpoint, and so the error is very small.
 - Errors add up over time; we can define the integral (I) of the error:

$$\Sigma_{\text{time}} (\text{setpoint} - \text{output})$$

d) Proportional-Integral-Derivative (PID) Controller:

- This is most powerful, but also the most difficult to tune.
- Once we can set the parameters properly, it gives very good performance.



To Summarize

Controller	Response Time	Overshoot	Error
ON-OFF	Smallest	Highest	Large
Proportional	Small	Large	Small
Integral	Decreases	Increases	Zero
Derivative	Increases	Decreases	Small change



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 31: Experiments with Relay

CONCEPTS COVERED

- ☐ Relay interfacing
- ☐ Experiments with relay interface
- ☐ Demonstration



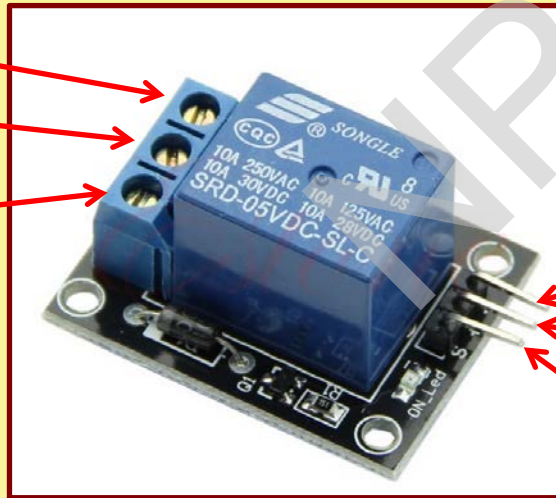
About the SRD-05DC-SL-C Relay

- Can be controlled using a 0V – 5V digital signal.
- The output load can be 10A, 250VAC.
- It is an electro-mechanical relay, with two states.

Normally Closed (NC)

Common (C)

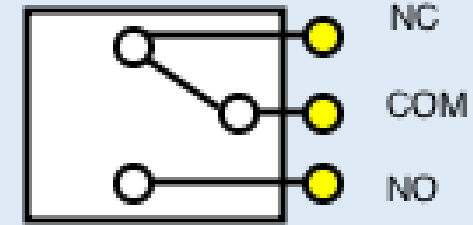
Normally Open (NO)



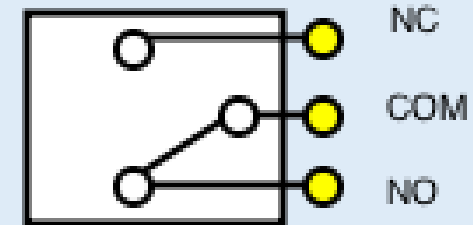
GND

+5V

Signal

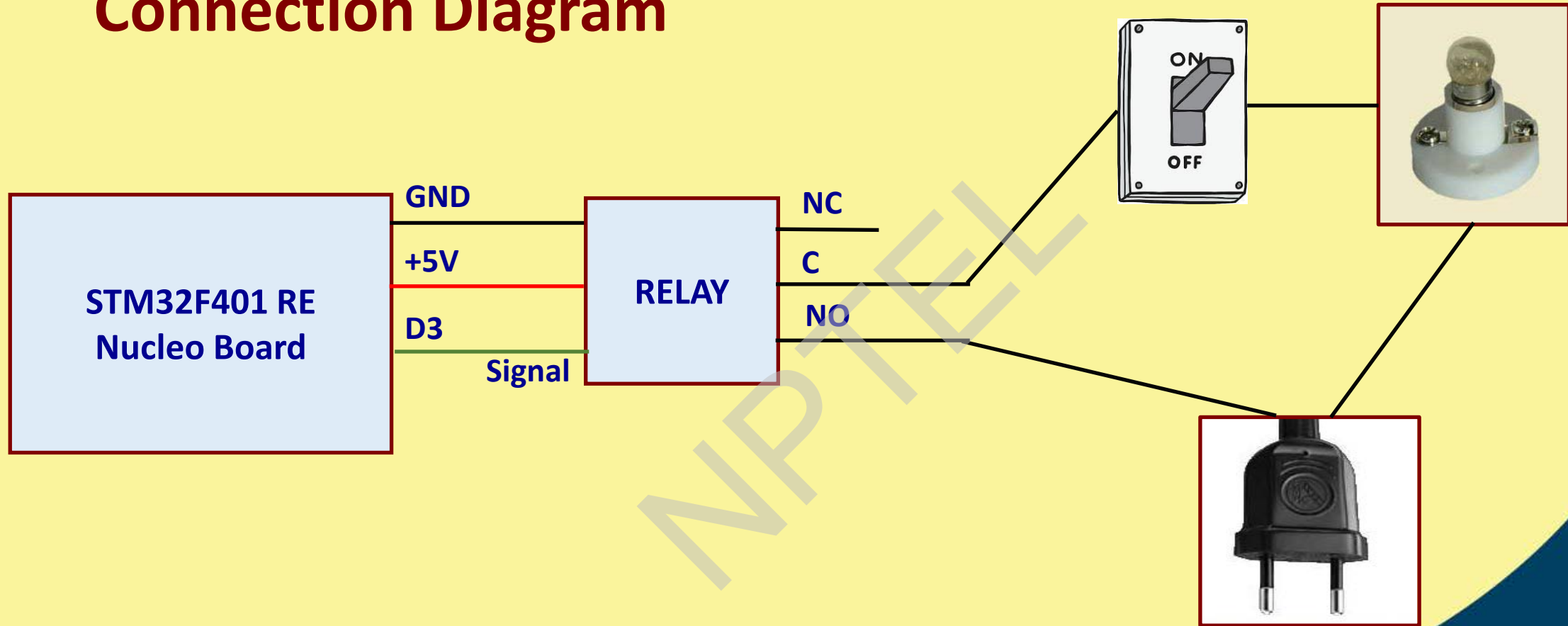


Relay: Is normal



Relay: Is triggered

Connection Diagram



Experiment 1

- Interface an electric bulb through relay, and switch it ON and OFF under program control through digital output port.
 - The relay is controlled by the digital output line D3 of the STM32 board.
 - The program turns ON the bulb for 2 seconds, turns it OFF for 5 seconds, and repeats it in a loop.

Mbed C Code

```
// Turn on a bulb for 2sec and off for 5sec repeatedly through a  
// relay, which is controlled by digital port line D3.
```

```
#include "mbed.h"  
DigitalOut relay (D3);  
int main() {  
    while(1) {  
        relay = 1;        // Turn ON bulb, and wait for 2 sec  
        wait(2.0);  
        relay = 0;        // Turn OFF bulb, and wait for 5 sec  
        wait(5.0);  
    }  
}
```

Experiment 2

- Interface an electric bulb through relay, and vary its brightness under program control through PWM port.
 - The circuit connection remains the same, but now the digital port D3 is used with PWM control.
 - By changing the duty cycle, the brightness of the bulb can be controlled.

Mbed C Code for STM32

```
#include "mbed.h"

PwmOut bulb (D3);    // D3 is set as a PWM controlled digital output
int main() {
    bulb.period (0.02);    // PWM period 20 msec

    while(1) {
        bulb.write (1.0);    // duty cycle = 1.0;
        wait (3.0);    // wait for 3 second

        bulb.write (0.8);    // duty cycle = 0.8;
        wait (3.0);    // wait for 3 second
    }
}
```

Mbed C Code

```
bulb.write (0.6);    // duty cycle = 0.6;
wait (3.0);          // wait for 3 second

bulb.write (0.4);    // duty cycle = 0.4;
wait (3.0);          // wait for 3 second

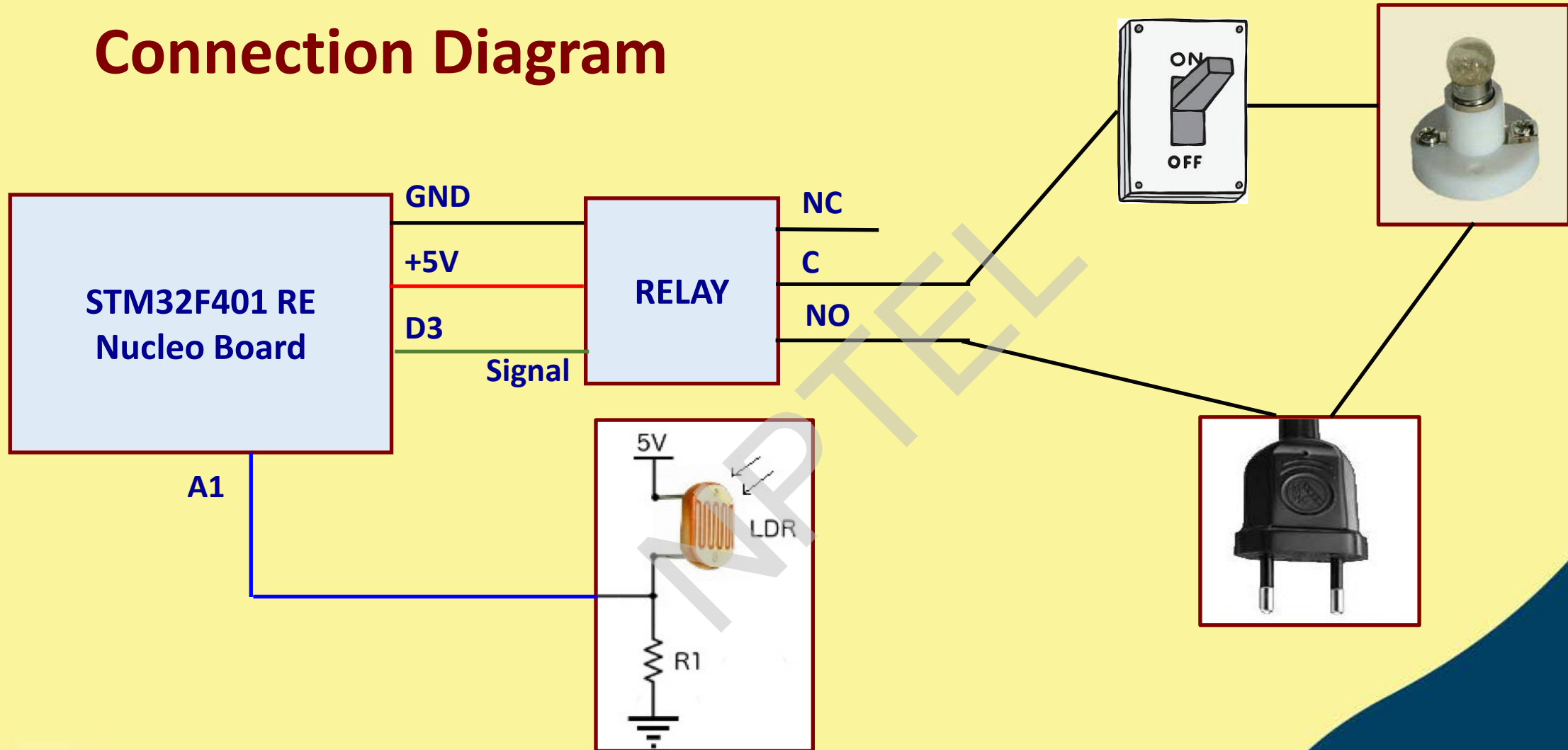
bulb.write (0.2);    // duty cycle = 0.2;
wait (3.0);          // wait for 3 second

bulb.write (0.0);    // duty cycle = 0.0;
wait (3.0);          // wait for 3 second
}
}
```

Experiment 3

- Interface an electric bulb through relay, and turn it ON or OFF depending on ambient light as sensed using a LDR.
 - The relay is connected through digital output pin D3 as usual.
 - A LDR light sensing circuit, that generates an analog output voltage, is connected to the analog input pin A1.

Connection Diagram



Mbed C Code

```
#include "mbed.h"

PwmOut bulb (D3); // D3 is set as a PWM controlled digital output
AnalogIn ldr (A1); // LDR circuit output is connected to pin A1
int main()
{
    float light;
    int value;
    bulb.period (0.02);
    while(1) {
        light = ldr.read();
        value = light * 5000;
        if (value > 4000)
            bulb.write (0.0);
        else
            bulb.write (1.0);
    }
}
```



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 32: Experiments on Speed Control of DC Motor

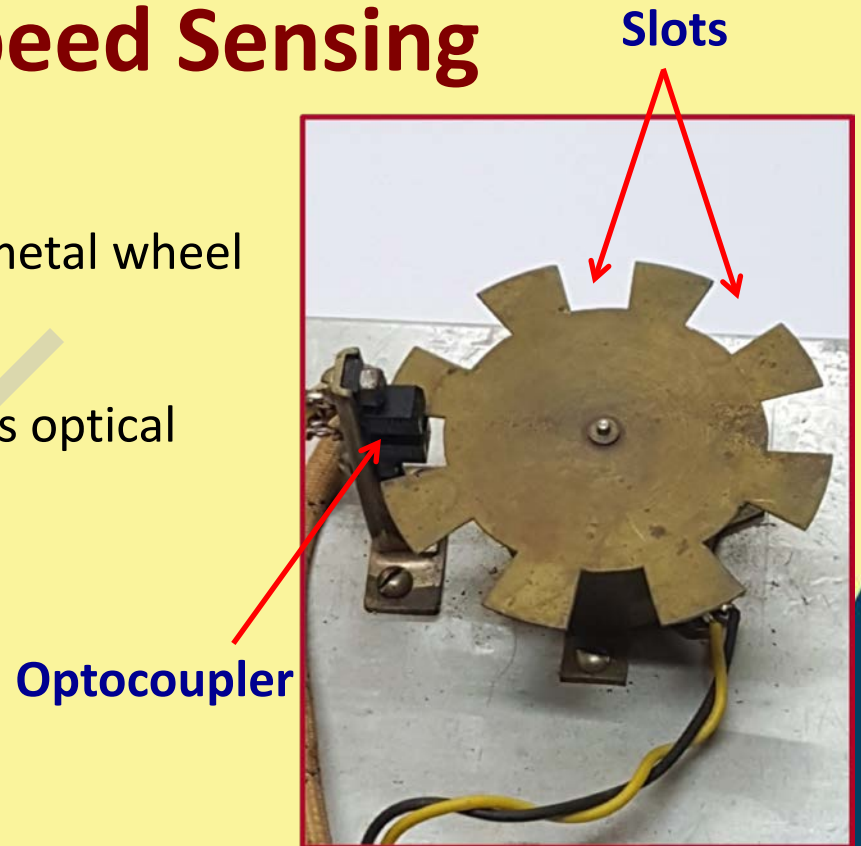
CONCEPTS COVERED

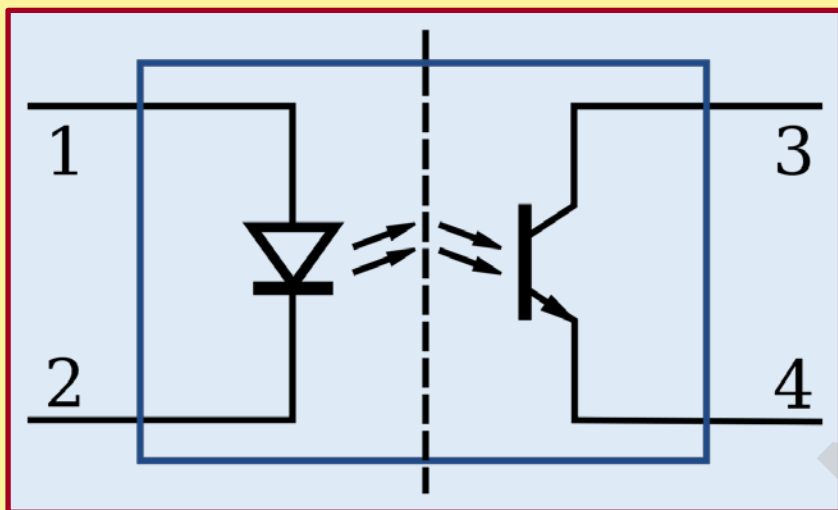
- ☐ Motor interface and speed sensing
- ☐ Experiments with motor interface
- ☐ Demonstration



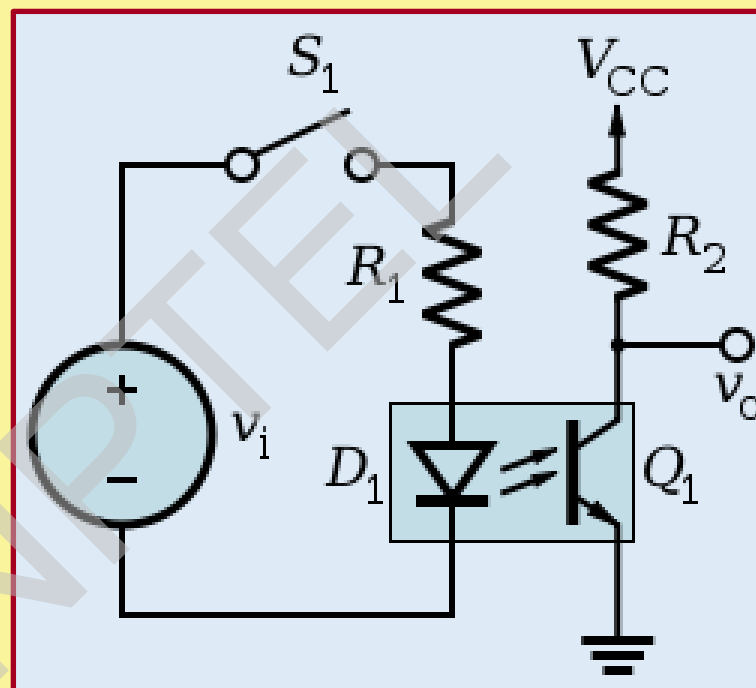
About the Motor Interface and Speed Sensing

- A DC motor is used, where the rotating shaft is connected to a metal wheel cut with 8 slots.
- The wheel is attached to an optocoupler circuit, which generates optical interruptions whenever the wheel rotates.
 - 8 optical interrupts for every single rotation of the wheel.
 - An optocoupler circuit generates a pulse for every interruption.
- How is the motor driven?
 - Directly from the PWM port of the microcontroller.
 - By changing the PWM duty cycle, the speed of the motor can be varied.





Optocoupler

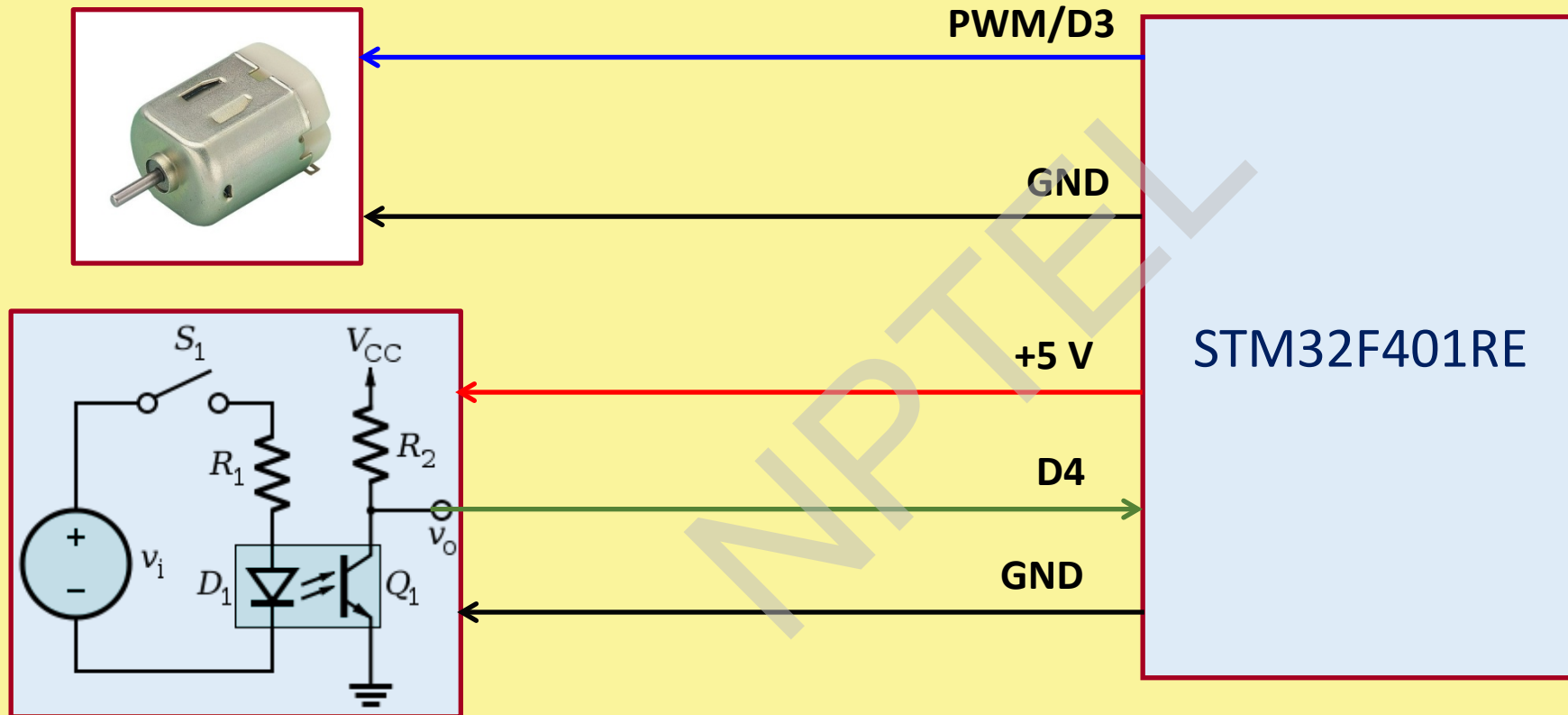


Sensing Optical Interruptions



MOC7811

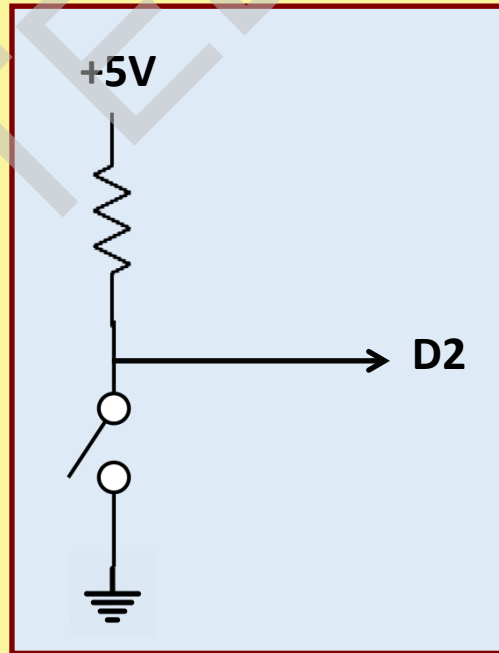
Connection Diagram – STM32



Experiment 1

- Interface the motor and a push switch. On successive presses of the switch, the following will alternate repeatedly:
 - a) Run the motor at high speed
 - b) Run the motor at low speed
 - c) Turn off the motor

- The push switch is interfaced to port line D2.
- PWM control is used to drive the motor.



Embed C Code – STM32

```
#include "mbed.h"

PwmOut motor (D3);           // D3 is set as a PWM controlled digital output
DigitalIn push_switch (D2);  // Switch output is connected to pin D2
int main()
{
    int state = 1;
    motor.period (0.10);
    motor = 1.0;               // Duty cycle = 1.0
    while(1) {
        if (push_switch == 0) { // Check for switch press
            state++;
            if (state == 4) state = 1;
        }
        while (push_switch == 0) ; // Wait for key release
    }
}
```

```
switch (state) {           // Set duty cycle depending on state
    case 1: motor = 1.0;
               break;
    case 2: motor = 0.8;
               break;
    case 3: motor = 0.0;
               break;
}
}
```

Experiment 2

- Interface the motor and read the pulses generated by the rotary sensor. Count the pulses and display the RPM on a LCD display unit.
- The pulses generated by the optocoupler are fed to interrupt input D4.
 - PWM control is used to drive the motor.
 - LCD display unit is also interfaced.

Embed C Code – STM32

```
#include "mbed.h"
#include "TextLCD.h"
#include "TextLCDScroll.h"

PwmOut motor (D3);           // D3 is set as a PWM controlled digital output
InterruptIn wheel (D4);      // Optocoupler output is connected to pin D4

TextLCDScroll lcd(D13, D12, D11, D10, D9, D8, TextLCD::LCD16x2);

int pulses = 0;

void count()
{
    pulses = pulses + 1;
}
```

Embed C Code – STM32

```
int main()
{
    int RPM; char msg[20];
    wheel.rise (&count);
    motor.period (0.10);
    motor = 0.9;
    lcd.cls();

    while (1) {
        wait (1.0);
        RPM = pulses * 60 / 8;    // In a minute; 8 slots in the wheel
        sprintf (msg, "%6d", RPM);
        lcd.setLine (0, "The RPM is:");
        lcd.setLine (1, msg);
        pulses = 0;
    }
}
```



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 33: Experiment with Multiple Sensors and Relay

CONCEPTS COVERED

- ❑ Interfacing multiple sensor and multiple output devices
- ❑ Demonstration



Introduction

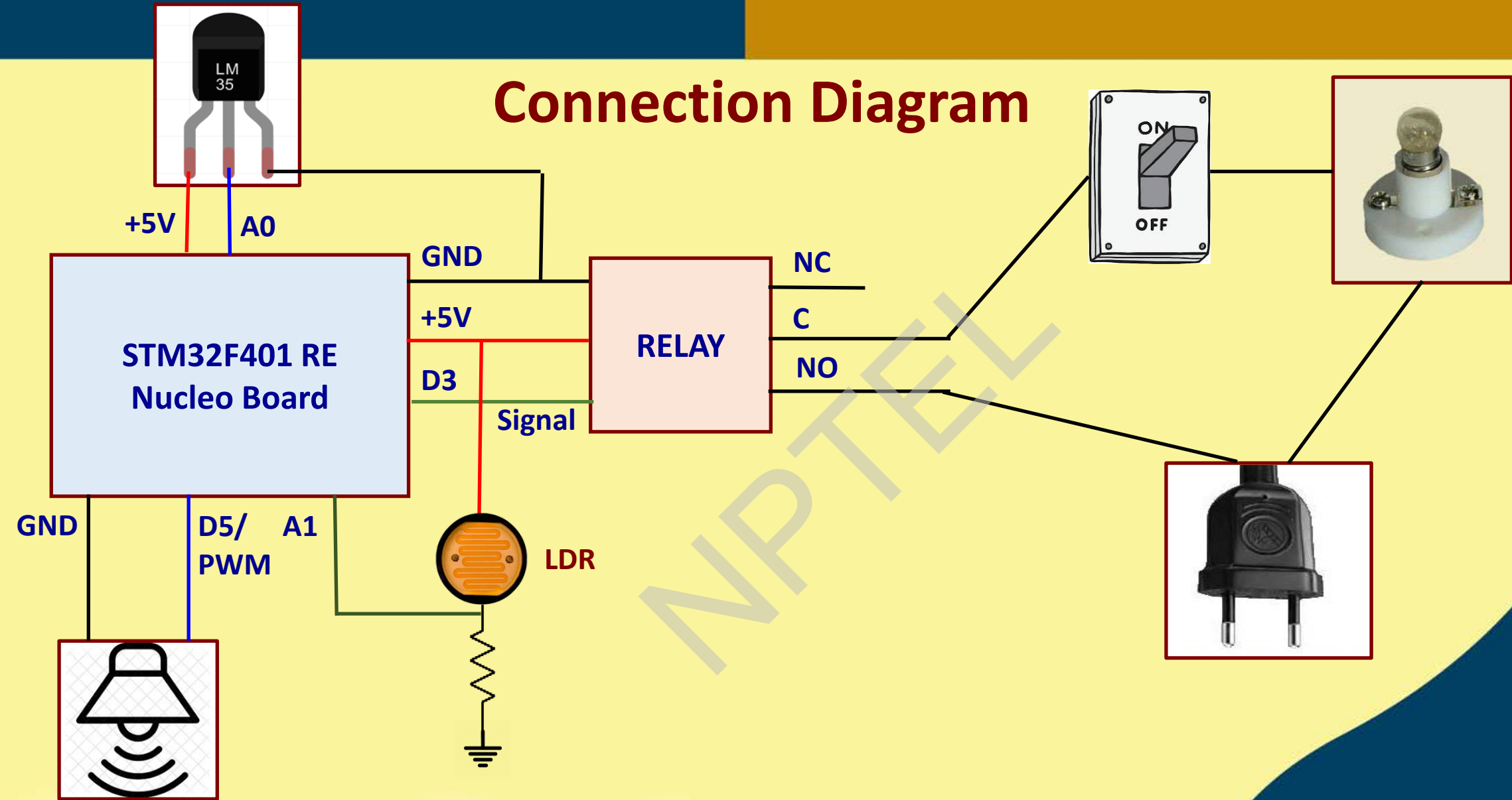
- In this experiment, we shall consider the interfacing of multiple sensors and multiple output devices.
 - Input devices: LDR, LM35 temperature sensor
 - Output devices: Relay driving a bulb, Speaker
- We shall demonstrate how the same microcontroller can be used to perform multiple tasks in a time multiplexed way.
- The experiment is also realistic enough such that it can be related to practical scenarios of home automation.

The Experiment

Interface relay, speaker, LDR, and LM35 to the STM32F401 Nucleo board:

- The relay circuit is used to drive a bulb (*driven by D3*).
- The speaker is interfaced to generate an audible output (*driven by D5 using PWM*).
- The LDR circuit is used to sense the level of ambient light (*connected to A1*).
- The LM35 sensor is used to measure the temperature (*connected to A0*).
- If the ambient light falls below a threshold, the bulb will turn on.
- Whenever the temperature crosses a threshold, an alarm will sound.

Connection Diagram



Basic Program Logic

- The steps as shown must run in a repetitive loop.
- There can be a delay before the loop repeats.
- First, the LDR generated sensor voltage is read through A0.
- Then, LM35 generated temperature data is read through A1.

```
begin
  if (value(A0) > thres_light)
    D3 = 0;           // Turn OFF relay (bulb)
  else
    D3 = 1;           // Turn ON relay (bulb)
  if (value(A1) > thres_temp)
    D5 = PWM tone;    // Turn ON alarm
  else
    D5 = No PWM tone; // Turn OFF alarm
end
```

Mbed C Code for STM32F401

```
#include "mbed.h"
PwmOut bulb (D3);      // Relay connected to PWM output D3
PwmOut speaker (D6);    // Speaker connected to PWM output D5
AnalogIn ldr (A1);      // LDR circuit output is connected to pin A1
AnalogIn lm35 (A0);     // LM35 output is connected to pin A0

int main()
{
    float light, temper;
    int lvalue, tvalue;
    bulb.period (0.02);
    speaker.period_ms (3);
```

```

while(1) {
    light = ldr.read();           // First sense the ambient light
    lvalue = light * 5000;
    if (lvalue > 4000)           // Turn on bulb if required
        bulb.write (0.0);
    else
        bulb.write (1.0);

    temper = lm35.read();        // Next sense the temperature
    tvalue = temper * 1000;
    if (tvalue > 96) {           // Generate fire alarm if required
        speaker = 0.5;          // 333 Hz, 50% duty cycle tome
    }
    else {
        speaker.period_us (3000);
        speaker = 0.0;          // 0% duty cycle (NO TONE)
    }
}
}

```



NPTEL ONLINE CERTIFICATION COURSES

**Thank
you!**