Microprocessor – Microcontroller

W04 – Multi-state Systems and Function Sequences

Nguyen Tran Huu Nguyen
D: Computer Engineering
E: nthnguyen@hcmut.edu.vn

Introduction

- Two broad categories of multi-state systems
- Multi-State (Timed)
  - The transition between states will depend only on the passage of time
- Multi-State (Input/Timed)
  - More common form of system
  - Transition between states and behavior in each state will depend both on the passage of time and on system inputs

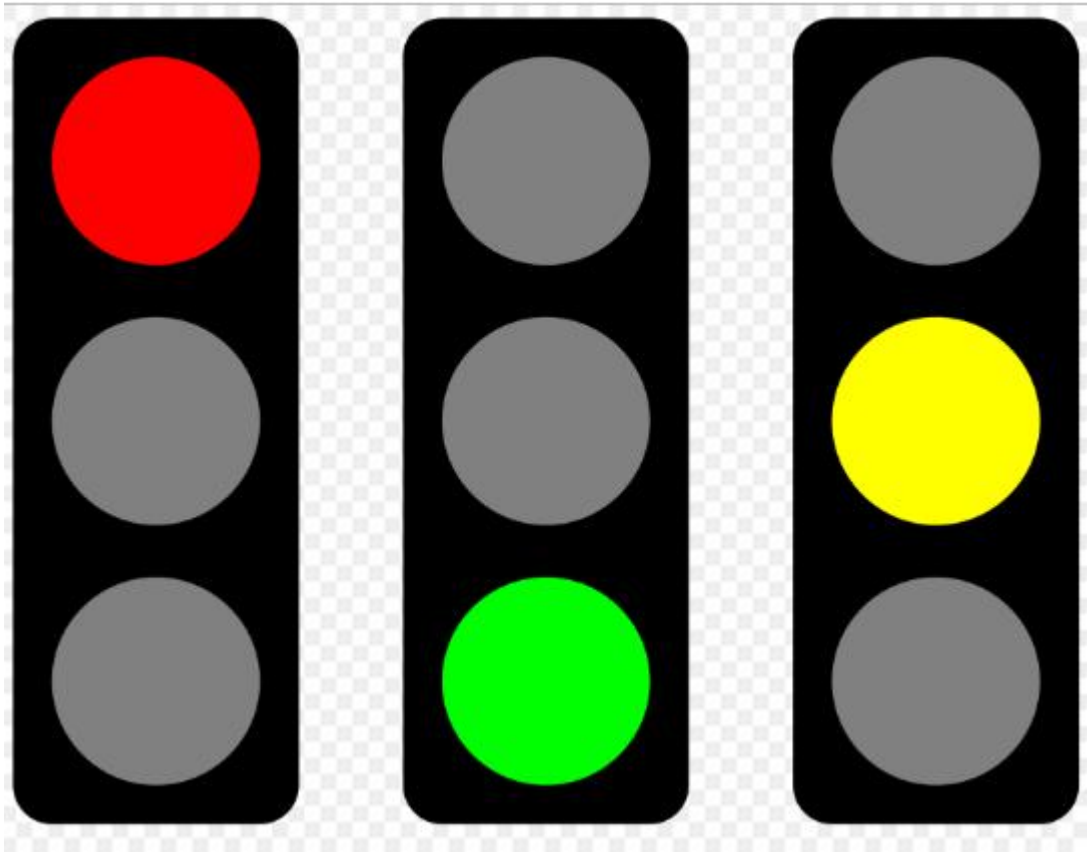
Multi-state (Timed) systems

- The system will operate in two or more states
- Each state may be associated with one or more function calls
- Transitions between states will be controlled by the passage of time
- Transitions between states may also involve function calls

- Please note that, in order to ease subsequent maintenance tasks, the system states should not be arbitrarily named, but should – where possible – reflect a physical state observable by the user and /or developer.

- Please also note that the system states will usually be represented by means of a switch statement

Traffic light



Traffic light

- The various states are easily identified
  - RED
  - GREEN
  - AMBER
- In the code, these states can be represented as follows:

```
/* Possible system state */
typedef enum {RED, GREEN, AMBER} eLightState
```

Traffic light

- The time to be spent in each state will be stored as follows:

```
/* Times are in second*/
#define RED_DURATION 20
#define GREEN_DURATION 10
#define YELLOW_DURATION 3
```

Traffic light

- In this simple case, we do not require function calls from or between system states.
- The required behavior will be implemented directly through control of the three port pins which - in the final system – would be connected to appropriate bulbs.

Main.c

```
void main(void){
    SYSTEM_Initialize();
    INTERRUPT_GlobalInterruptEnable();
    INTERRUPT_PeripheralInterruptEnable();
   // Prepare to run traffic sequence
   Traffic_Light_Init(RED);
   //Set up 50ms ticks
   Init_Timer0(50);
   while(1) {  //Supper loop
         Go_To_Sleep();
    }
}
```

Traffic_Light.h

```
#ifndef _T_LIGHT_H
#define _T_LIGHT_H
//Possible system states
typedef enum {RED, GREEN, AMBER } eLightState;

//public function prototype
void Traffic_Light_Init(const eLightState);
void Traffic_Light_Update(void);

#endif
```

Traffic_Light.c

- Working in group of four students
- What if the Traffic_Light_Update function is called in timer interrupt service routine of 50 ms.
- What if the Traffic_Light_Update function is called in main function.

TRAFFIC_LIGHTS_Update()

Must be called once per second.

```
void TRAFFIC_LIGHTS_Update(void){
    static tWord Time_in_state;
    switch (Light_state_G){
        case Red:
                Red_light = ON;
                Amber_light = OFF;
                Green_light = OFF;
                    if (++Time_in_state == RED_DURATION){
                        Light_state_G = Green;
                        Time_in_state = 0;
                    }
                 break;
```

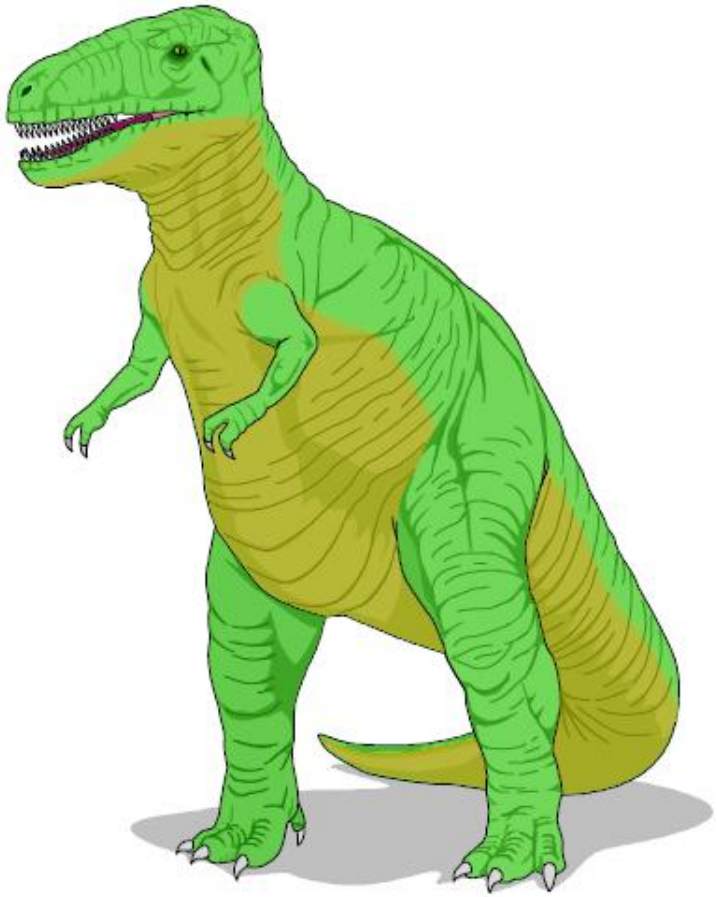
TRAFFIC_LIGHTS_Update()

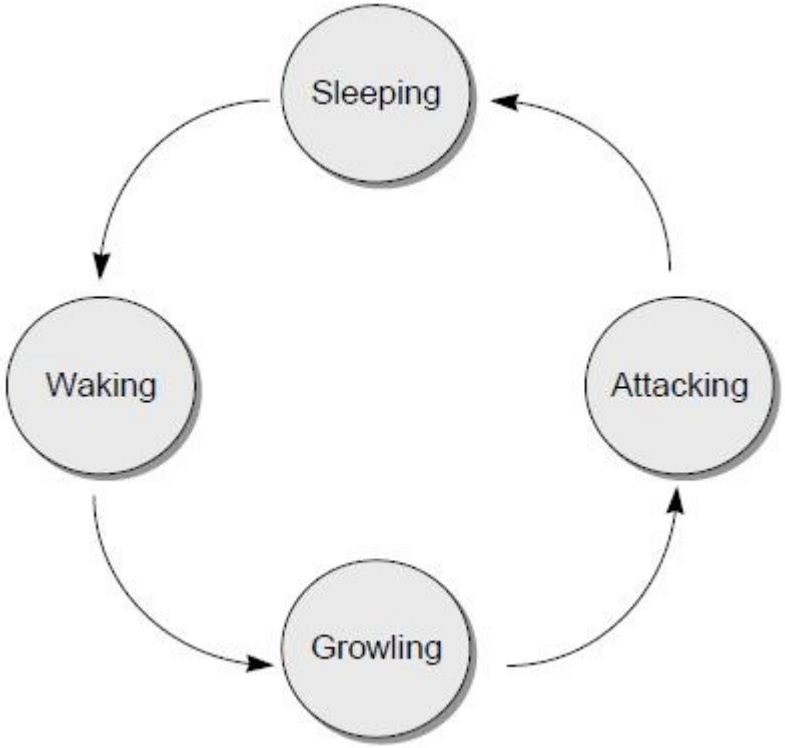
Must be called once per second.

```
case Green:
    Red_light = OFF;
    Amber_light = OFF;
    Green_light = ON;
     if (++Time_in_state == GREEN_DURATION){
        Light_state_G = Amber;
        Time_in_state = 0;
    }
    break;
```

```
case Amber:
    Red_light = OFF;
    Amber_light = ON;
    Green_light = OFF;
     if (++Time_in_state == AMBER_DURATION){
        Light_state_G = Red;
        Time_in_state = 0;
    }
    break;
```

Example: Animatronic dinosaur





The system states

- Sleeping
  - The dinosaur will be largely motionless, but will be obviously “breathing”. Irregular snoring noises, or slight movement during this time will add interest for the audience.
- Waking
  - The dinosaur will begin to wake up. Eyelids will begin to flicker. Breathing will become more rapid
- Growling
  - Eyes will suddently open, and the dinosaur will emit a very loud growl. Some further movement and growling will follow.
- Attacking
  - Rapid ‘random’ movements towards the audience. Lots of noise (you should be able to hear this from the next floor in the museum)

A Multi-State (Input/Timed) System

- The system will operate in two or more states
- Each states may be associated with one or more function calls
- Transitions between states may be controlled by the passage of time, by system inputs or a combination of time and inputs
- Transition between states may also involve function calls

Implementing State timeouts

- Consider the following informal system requirements
  - The pump should be run for 10 seconds.
  - If, during this time, no liquid is detected in the outflow tank, the pump should be switched off and ‘low water’ warning should be sounded.
  - If liquid is detected, the pump should be run for a further 45 seconds, or until the ‘high water’ sensor is activated (whichever is first)

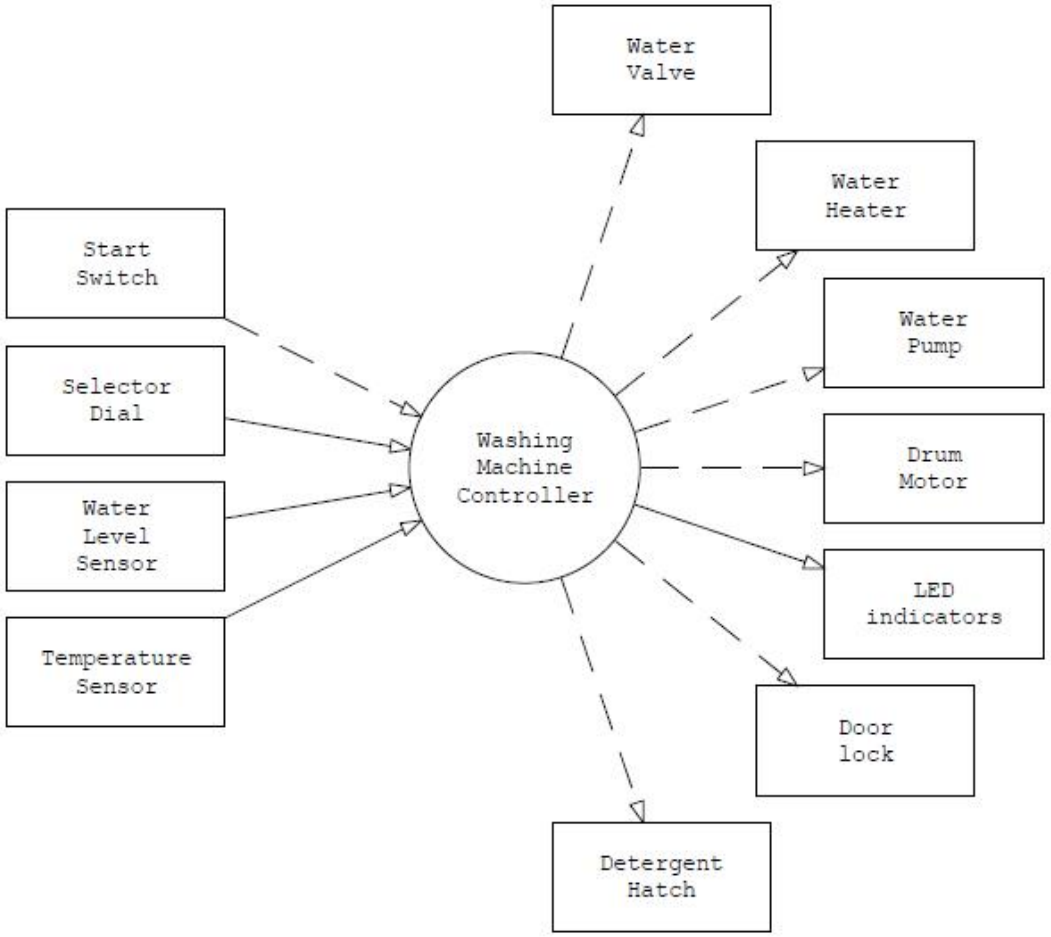
Implementing State timeouts

- After the front door is opened, the correct password must be entered on the control panel within 30 seconds or the alarm will sound.

Implementing State timeouts

- To meet this type of requirements, we can do two things
  - Keep track of the time in each system state;
  - If the time exceeds a predetermined error value, then the system should move to a different state

Example: Washing machine controller



System operations

1. The user selects a wash program (e.g. ‘Wool’, ‘Cotton’) on the selector dial.
2. The user presses the ‘Start’ button.
3. The door lock is engaged.
4. The water valve is opened to allow water into the wash drum.
5. If the wash program involves detergent, the detergent hatch is opened. When the detergent has been released, the detergent hatch is closed.
6. When the ‘full water level’ is sensed, the water valve is closed.
7. If the wash program involves warm water, the water heater is switched on. When the water reaches the correct temperature, the water heater is switched off.
8. The washer motor is turned on to rotate the drum. The motor then goes through a series of movements, both farward and reverse (at verious speeds) to wash the clothes. The precise set of movements carried out depends on the wash program the user has selected.) At the end of the wash cycle, the motor is stopped.
9. The pump is switched on to drain the drum. When the drum is empty, the pump is swithed off.

Washer_Update()

```
void WASHER_Update(void) {
    static tWord Time_in_state;
   switch (System_state_G) {
        case START:
            // Lock the door
              WASHER_Control_Door_Lock(ON);
            // Start filling the drum
              WASHER_Control_Water_Valve(ON);
              // Release the detergent (if any)
            if (Detergent_G[Program_G] == 1) {
                  WASHER_Control_Detergent_Hatch(ON);
            }
              // Ready to go to next state
            System_state_G = FILL_DRUM;
            Time_in_state_G = 0;
       break;
```

```
case FILL_DRUM: // Remain in this state until drum is full
    // NOTE: Timeout facility included here
    if (++Time_in_state_G>= MAX_FILL_DURATION){// Should have filled the drum by now
        System_state_G = ERROR;
    }
    // Check the water level
     if (WASHER_Read_Water_Level() == 1) {// Drum is full
         // Does the program require hot water?
         if (Hot_Water_G[Program_G] == 1) {
             WASHER_Control_Water_Heater(ON);
              // Ready to go to next state
             System_state_G = HEAT_WATER;
             Time_in_state_G = 0;
        }  else {  // Using cold water only
            // Ready to go to next state
              System_state_G = WASH_01;
              Time_in_state_G = 0;
         }
    }
break;
```

```
case HEAT_WATER:
        // Remain in this state until water is hot
        // NOTE: Timeout facility included here
        if (++Time_in_state_G >= MAX_WATER_HEAT_DURATION) {
        // Should have warmed the water by now...
                System_state_G = ERROR;
        }
        // Check the water temperature
        if (WASHER_Read_Water_Temperature() == 1) {
        // Water is at required temperature
        // Ready to go to next state
                System_state_G = WASH_01;
                Time_in_state_G = 0;
        }
break;
```

```
            case WASH_01:
                // All wash program involve WASH_01
                // Drum is slowly rotated to ensure clothes are fully wet
                WASHER_Control_Motor(ON);
                if (++Time_in_state >= WASH_01_DURATION) {
                        System_state_G = WASH_02;
                        Time_in_state = 0;
                }
            break;
            // REMAINING WASH PHASES OMITTED HERE ...
            case WASH_02:
            break;
            case ERROR:
            break;
        }
}
```