

# Input and Output



**Arijit Mondal**

Dept. of Computer Science & Engineering  
Indian Institute of Technology Patna  
arijit@iitp.ac.in

# Things to consider

- Mechanical and electrical properties of the interfaces are important
- Drawing too much current may result in malfunction
- In physical world most of the things run in parallel, software is sequential
- Incorrect interaction between sequential code and concurrent event in physical world may lead to catastrophe

# Interfaces

- Pulse width modulation (PWM)
  - Used to deliver variable power
    - Speed of motor, brightness of LED
  - Duty cycle is one of the key parameters
  - Typically operates using memory-mapped register
- General purpose digital I/O (GPIO)
  - A number of general purpose I/O pins are available in most microcontrollers
  - Voltage level in the pins are read/written to represent logic 0 or 1
  - Active high vs active low logic
  - External physical devices can be connected
    - Need to check current level
    - May require power amplifier
  - Electrical isolation
  - Schmitt triggered, Tristate

# Connection

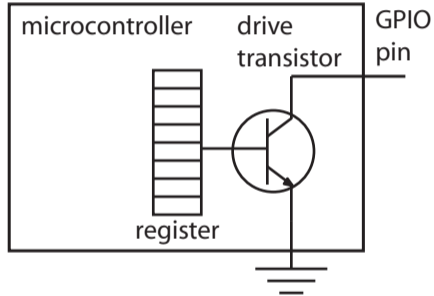
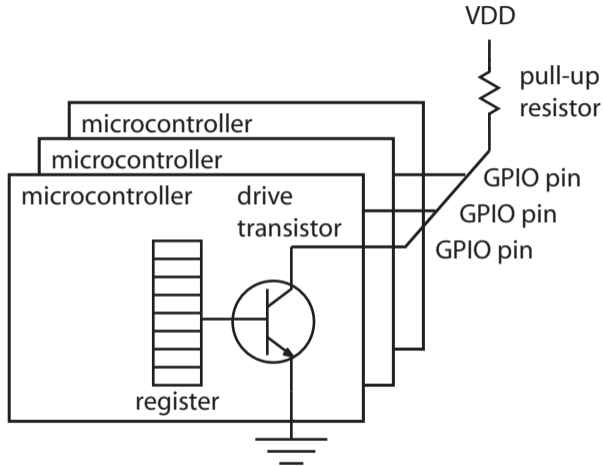


image source: [Introduction to Embedded Systems book](#)

# Connection



# Serial interface

- Embedded processor requires physical small package and low power consumption
  - Number of pins need to be reduced
- Send information serially as sequence of bits
- RS232 - one of the popular standard
  - Sender and receiver first agree on transmission rate
  - Sender initiates transmission of byte with a start bit that alerts receiver
  - Sender sends the data with agreed upon rate
  - There will be one or two stop bits
  - Receiver reset upon receiving start bit and samples the data using agreed upon rate
- USB - 3.0-4.8 GBits/sec
- I<sup>2</sup>C, SPI, PCI express

# Parallel interface

- It uses multiple lines to simultaneously send data
  - Each line is a serial interface
- Printer port (IEEE-1284)
- GPIO pins can be used to realize parallel interface
- Challenges are to maintain synchrony

# Buses

- Interface shared among multiple devices
  - USB - serial bus
  - SCSI - parallel bus
- ISA bus, PCI
- Architecture must include media access control (MAC)
  - MAC has single master that connect with slaves
  - Time triggered bus, token ring



# Interrupt and exception

- Interrupt - pausing of execution of whatever processor is currently doing and start executing predefined code sequence
  - Interrupt service routine (ISR)
  - Can be triggered by software or external hardware
- Exception is triggered by internal hardware that detects a fault
- For hardware or software interrupt program resumes its normal execution after completion of ISR
- Exception has the highest priority

# Example

```
volatile uint timerCount=0;
void countDown(void){
    if (timerCount != 0){
        timerCount- -;
    }
}
```

```
SysTickPeriodSet (SysCtlClockGet()/1000);
SysTickIntReg(&countDown);
SysTickEnable();
SysTickIntEnable();
```

```
int main(){
    timerCount = 2000;
    ...
    while(timerCount != 0){
        ... code run for 2 sec...
    }
}
```

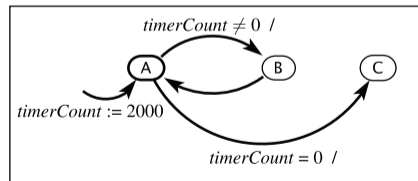
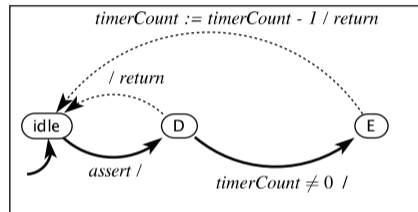
# Interrupt modeling

```
volatile uint timerCount = 0;
void ISR(void) {
  D → ... disable interrupts
  E → if(timerCount != 0) {
      timerCount--;
    }
    ... enable interrupts
}
int main(void) {
  // initialization code
  SysTickIntRegister(&ISR);
  ... // other init
  A → timerCount = 2000;
  B → while(timerCount != 0) {
      ... code to run for 2 seconds
    }
  C → }
    ... whatever comes next
```

**variables:** *timerCount*: uint

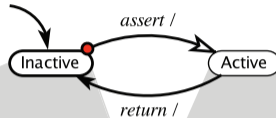
**input:** *assert*: pure

**output:** *return*: pure



# Interrupt modeling

input: *assert*, return: *pure*



```
int main(void) {  
    // initialization code  
    SysTickIntRegister(&ISR);  
    ... // other init  
    timerCount = 2000;  
    while(timerCount != 0) {  
        ... code to run for 2 seconds  
    }  
}
```

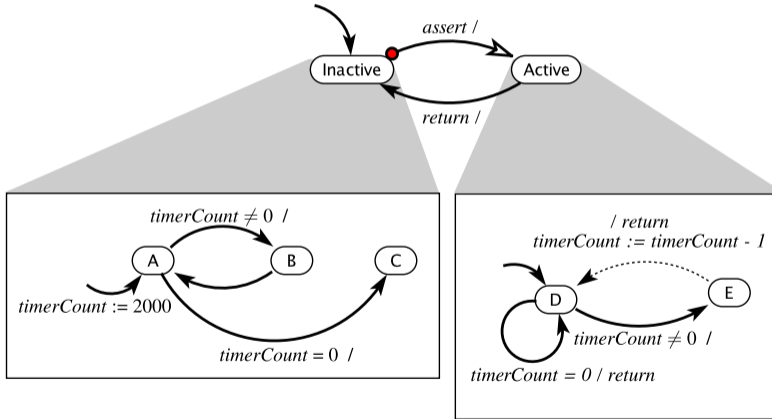
A →  
B →  
C →

```
volatile uint timerCount = 0;  
void ISR(void) {  
    ... disable interrupts  
    if(timerCount != 0) {  
        timerCount--;  
    }  
    ... enable interrupts  
}
```

D →  
E →

# Interrupt modeling

**variables:** *timerCount*: uint  
**input:** *assert*: pure, *return*: pure  
**output:** *return*: pure



# Interrupt modeling

**variables:** *timerCount*: uint  
**input:** *assert*: pure

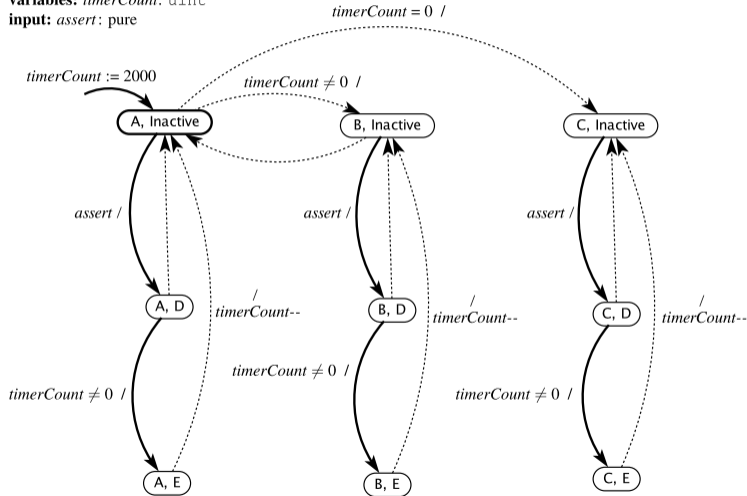


image source: [Introduction to Embedded Systems book](#)