



TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

# TM4C123GH6PM Micro-controllers Programming Concepts

Dr. Munesh Singh

Indian Institute of Information Technology  
Design and Manufacturing,  
Kancheepuram  
Chennai-600127

March 12, 2019





# IAR Workbench Toolset

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## ARM Microcontroller

The screenshot shows the IAR Embedded Workbench IDE interface for an ARM Cortex-M processor. The workspace contains a project named "Lesson0 - Debug" with files main.c and delay.c. The assembly view shows the main() function and its loop, with the PC register highlighted at 0x00000006. The registers window displays the current values of all CPU registers. The memory window shows the RAM contents starting at address 0x00000000. The log window at the bottom shows the build and run process for the TM4C123GH6P target.

```

Log
Mon Feb 11, 2019 11:50:04 IAR Embedded Workbench 8.32.1 (C:\Program Files (x86)\IAR Systems\Embedded Workbench 8.3\arm\bin\armproc.dll)
Mon Feb 11, 2019 11:50:04 Loaded macro file C:\Program Files (x86)\IAR Systems\Embedded Workbench 8.3\arm\config\TexasInstruments\TM4C123.dmcrc
Mon Feb 11, 2019 11:50:04 Download complete
Mon Feb 11, 2019 11:50:04 Loaded debugger: C:\IAR\DebugEx\Lesson0.ost
Mon Feb 11, 2019 11:50:04 Target reset

```





# IAR Workbench Toolset

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Local versus Global Variable

The screenshot shows the IAR Embedded Workbench IDE interface. The assembly window displays the main() function:

```
int main()
{
    int count=0;
    while(count<21)
    {
        //some code
    }
    return 0;
}
```

The Registers window shows CPU registers R0 through R15, all initialized to 0x00000000.

The Locals window shows a variable 'count' with a value of 1 at memory location 0x20000000.

The Watch 1 window shows the same variable 'count' with a value of 1 at the same memory location.

The Memory window shows memory starting at address 0x20000000. The byte at address 0x20000000 is highlighted in red and has the value 0x01. The assembly code shows a MOVW instruction at address 0x00000000 that writes the value 0x00000001 to R0.

The Log window shows the following build and run logs:

```
Feb 11, 2019 12:11:01 IAR Embedded Workbench 8.32.1 [C:\Program Files (x86)\IAR Systems\Embedded Workbench 8.3\arm\bin\armproc.dll]
Feb 11, 2019 12:11:01 Loaded macro file C:\Program Files (x86)\IAR Systems\Embedded Workbench 8.3\arm\conf\debuger\TexasInstruments(TM4C123).dmc
Feb 11, 2019 12:11:01 Download complete.
Feb 11, 2019 12:11:01 Loaded debugger C:\API\Debug\Ex\Lesson0.out
Feb 11, 2019 12:11:01 Target reset
```

The status bar at the bottom right indicates "Activate Windows" and "Go to Settings to activate Windows".

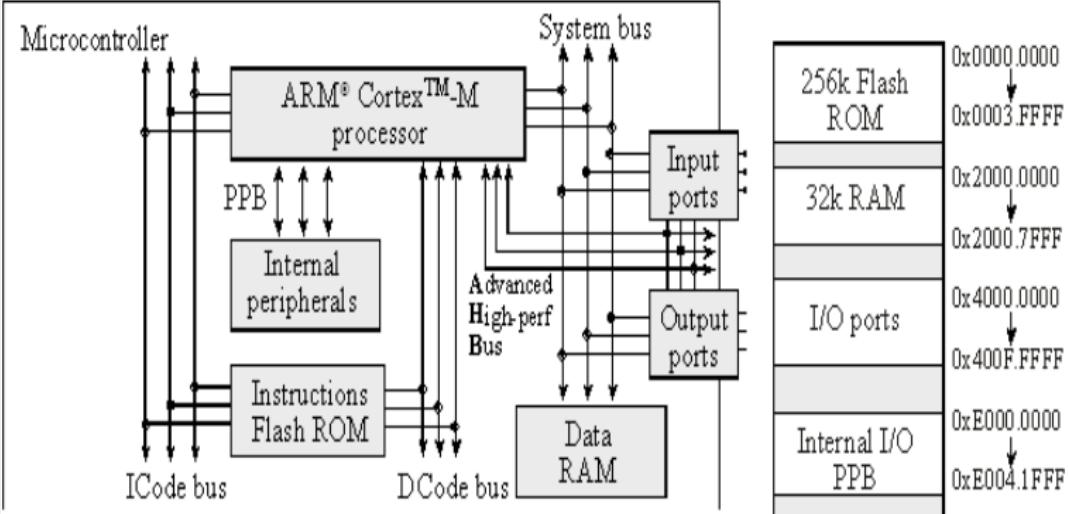


# Memory Map Buses

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh



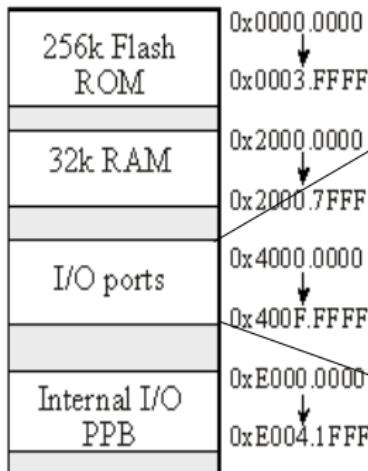


# Memory Map I/O Ports

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

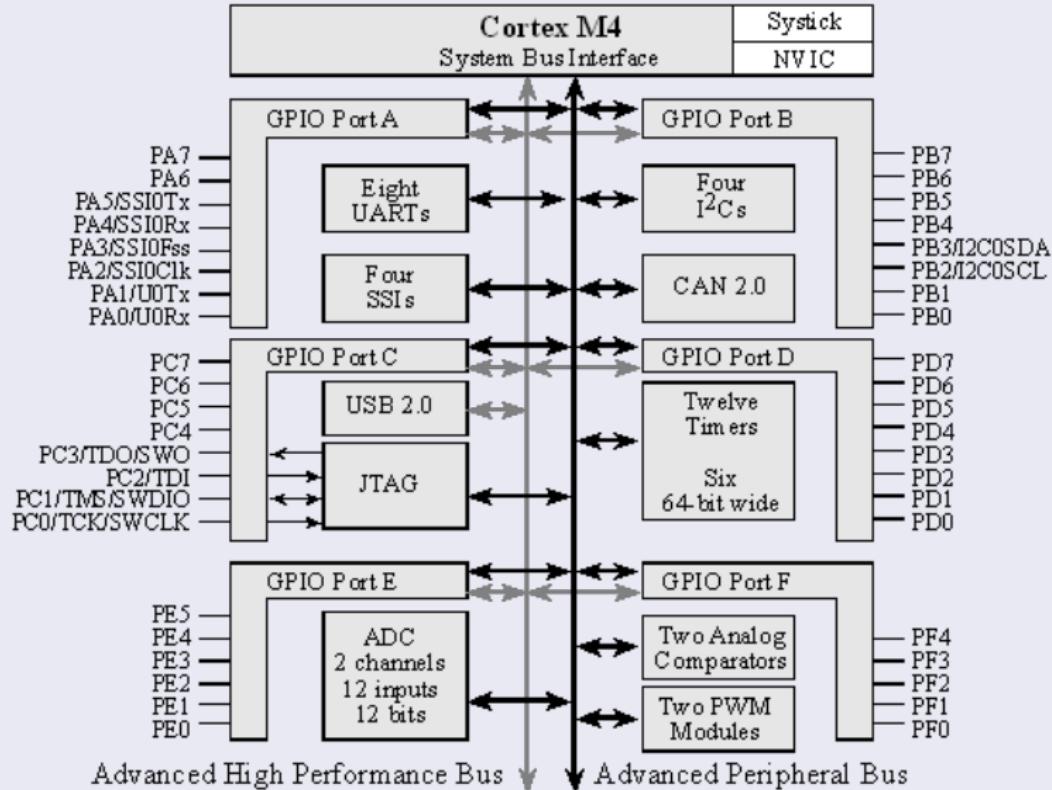
Dr. Munesh  
Singh



|             |             |             |
|-------------|-------------|-------------|
| 0x4000.4000 | 0x4000.4FFF | GPIO Port A |
| 0x4000.5000 | 0x4000.5FFF | GPIO Port B |
| 0x4000.6000 | 0x4000.6FFF | GPIO Port C |
| 0x4000.7000 | 0x4000.7FFF | GPIO Port D |
| 0x4002.4000 | 0x4002.4FFF | GPIO Port E |
| 0x4002.5000 | 0x4002.5FFF | GPIO Port F |



# Classification of I/O Ports





# I/O Port Initialization

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Seven Steps to Initialize a Port

- General-Purpose Input/Output Sleep Mode Clock Gating Control (SCGCGPIO)
- **Base 0x400F.E000**
- **Offset 0x608**
- Type RW, reset 0x0000.0000
- Actual address of this register calculated by adding the base address with offset
- Actual address= **0x400FE608**



# SCGCGPIO Register

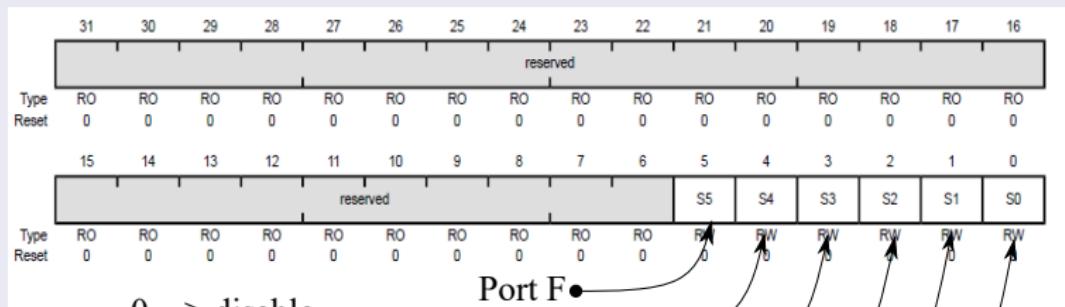
TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Register Information

- Activate the clock gating register



0---> disable

1---> enable

Port F

Port E

Port D

Port C

Port B

Port A



# Debugger Mode Memeory View SCGCGPIO Register and F Port

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## F Port Before Setting bit 5 of SCGCPIO register

Symbolic Memory

Go to: 0x400FE608    Memory    Previous    Next

| Location   | Data         | Variable | Value | Type |
|------------|--------------|----------|-------|------|
| 0x400FE5FC | 0x0000000000 |          |       |      |
| 0x400FE600 | 0x0000000000 |          |       |      |
| 0x400FE604 | 0x0000000000 |          |       |      |
| 0x400FE608 | 0x0000000000 |          |       |      |
| 0x400FE60C | 0x0000000000 |          |       |      |
| 0x400FE610 | 0x0000000000 |          |       |      |
| 0x400FE614 | 0x0000000001 |          |       |      |

Memory 1

Go to: 0x40025000    Memory

|            |       |
|------------|-------|
| 0x40024f90 | ----- |
| 0x40024fa0 | ----- |
| 0x40024fb0 | ----- |
| 0x40024fc0 | ----- |
| 0x40024fd0 | ----- |
| 0x40024fe0 | ----- |
| 0x40024ff0 | ----- |
| 0x40025000 | ----- |
| 0x40025010 | ----- |



# Debugger Mode Memory View SCGCGPIO Register and F Port

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## F Port after Setting bit 5 of SCGCPIO register

Symbolic Memory

| Location   | Data         | Variable | Value | Type |
|------------|--------------|----------|-------|------|
| 0x400FE5FC | 0x0000000000 |          |       |      |
| 0x400FE600 | 0x0000000000 |          |       |      |
| 0x400FE604 | 0x0000000000 |          |       |      |
| 0x400FE608 | 0x0000000020 |          |       |      |
| 0x400FE60C | 0x0000000000 |          |       |      |
| 0x400FE610 | 0x0000000000 |          |       |      |
| 0x400FE614 | 0x0000000001 |          |       |      |

Memory 1

| Go to      | Memory                                  | ▼ | ▲ | ↔ |
|------------|---|---|---|---|
| 0x40024f90 | -----                                   |   |   |   |
| 0x40024fa0 | -----                                   |   |   |   |
| 0x40024fb0 | -----                                   |   |   |   |
| 0x40024fc0 | -----                                   |   |   |   |
| 0x40024fd0 | -----                                   |   |   |   |
| 0x40024fe0 | -----                                   |   |   |   |
| 0x40024ff0 | -----                                   |   |   |   |
| 0x40025000 | 0000 0000 0000 0000 0000 0000 0000 0000 |   |   |   |
| 0x40025010 | 0000 0000 0000 0000 0000 0000 0000 0000 |   |   |   |



# GPIODIR Direction Register

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Register Information

|          | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Type     | RO |
| Reset    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Type     | RO | RW |
| Reset    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| DIR      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

0---> Corresponding pin input

1---> Corresponding pin output

):7 Bits

### GPIO Direction (GPIODIR)

GPIO Port A (APB) base: 0x4000.4000  
GPIO Port A (AHB) base: 0x4005.8000  
GPIO Port B (APB) base: 0x4000.5000  
GPIO Port B (AHB) base: 0x4005.9000  
GPIO Port C (APB) base: 0x4000.6000  
GPIO Port C (AHB) base: 0x4005.A000  
GPIO Port D (APB) base: 0x4000.7000  
GPIO Port D (AHB) base: 0x4005.B000  
GPIO Port E (APB) base: 0x4002.4000  
GPIO Port E (AHB) base: 0x4005.C000  
GPIO Port F (APB) base: 0x4002.5000  
GPIO Port F (AHB) base: 0x4005.D000  
Offset 0x400  
Type RW, reset 0x0000.0000



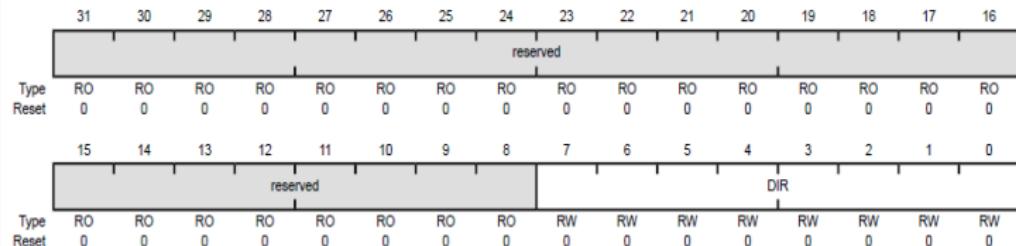
# GPIODEN Digital Function Register

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Register Information



## GPIO Digital Enable (GPIODEN)

0---> Disable Digital function  
1---> Enable Digital function

0:7 Bits

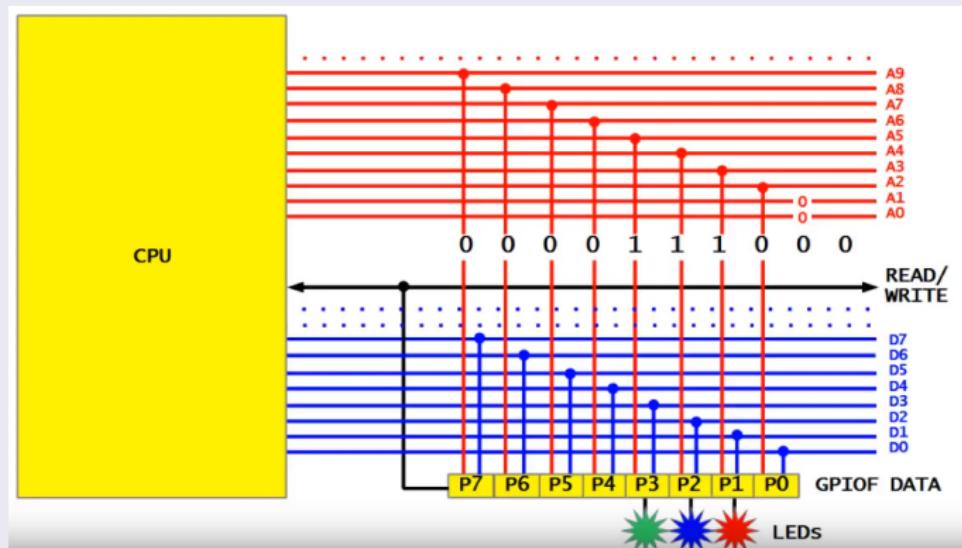
GPIO Port A (APB) base: 0x4000.4000  
GPIO Port A (AHB) base: 0x4005.8000  
GPIO Port B (APB) base: 0x4000.5000  
GPIO Port B (AHB) base: 0x4005.9000  
GPIO Port C (APB) base: 0x4000.6000  
GPIO Port C (AHB) base: 0x4005.A000  
GPIO Port D (APB) base: 0x4000.7000  
GPIO Port D (AHB) base: 0x4005.B000  
GPIO Port E (APB) base: 0x4002.4000  
GPIO Port E (AHB) base: 0x4005.C000  
GPIO Port F (APB) base: 0x4002.5000  
GPIO Port F (AHB) base: 0x4005.D000  
Offset 0x51C  
Type RW, reset -



# GPIO DATA Register

## Register Information

- In order to write to GPIO DATA, the corresponding bits in the mask, resulting from the address bus bits [9:2], must be set.





# GPIO DATA Register

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Register Information

|          |    |    |    |    |    |    |    |     |    |    |    |    |    |    |    |
|----------|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|
| 31       | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23  | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| reserved |    |    |    |    |    |    |    |     |    |    |    |    |    |    |    |
| Type     | RO  | RO | RO | RO | RO | RO | RO | RO |
| Reset    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 15       | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7   | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
| reserved |    |    |    |    |    |    |    | DIR |    |    |    |    |    |    |    |
| Type     | RO | RW  | RW | RW | RW | RW | RW | RW | RW |
| Reset    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

0:7 Bits

2 1 8 4 2 1 8 4 2 1  
9 8 7 6 5 4 3 2 1 0      Addr  
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓  
7 6 5 4 3 2 1 0      Pin

All seven port if want to use

3FC

0x400253FC

## GPIO Data (GPIO DATA)

GPIO Port A (APB) base: 0x4000.4000

GPIO Port A (AHB) base: 0x4005.8000

GPIO Port B (APB) base: 0x4000.5000

GPIO Port B (AHB) base: 0x4005.9000

GPIO Port C (APB) base: 0x4000.6000

GPIO Port C (AHB) base: 0x4005.A000

GPIO Port D (APB) base: 0x4000.7000

GPIO Port D (AHB) base: 0x4005.B000

GPIO Port E (APB) base: 0x4002.4000

GPIO Port E (AHB) base: 0x4005.C000

GPIO Port F (APB) base: 0x4002.5000

GPIO Port F (AHB) base: 0x4005.D000

Offset 0x000

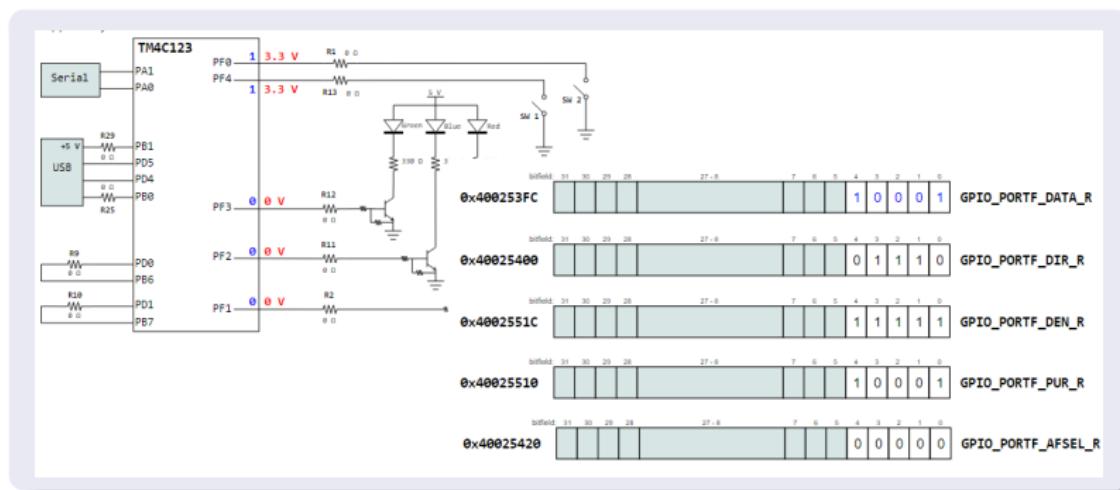
Type RW, reset 0x0000.0000



# Pin Diagram of TM4C123

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh





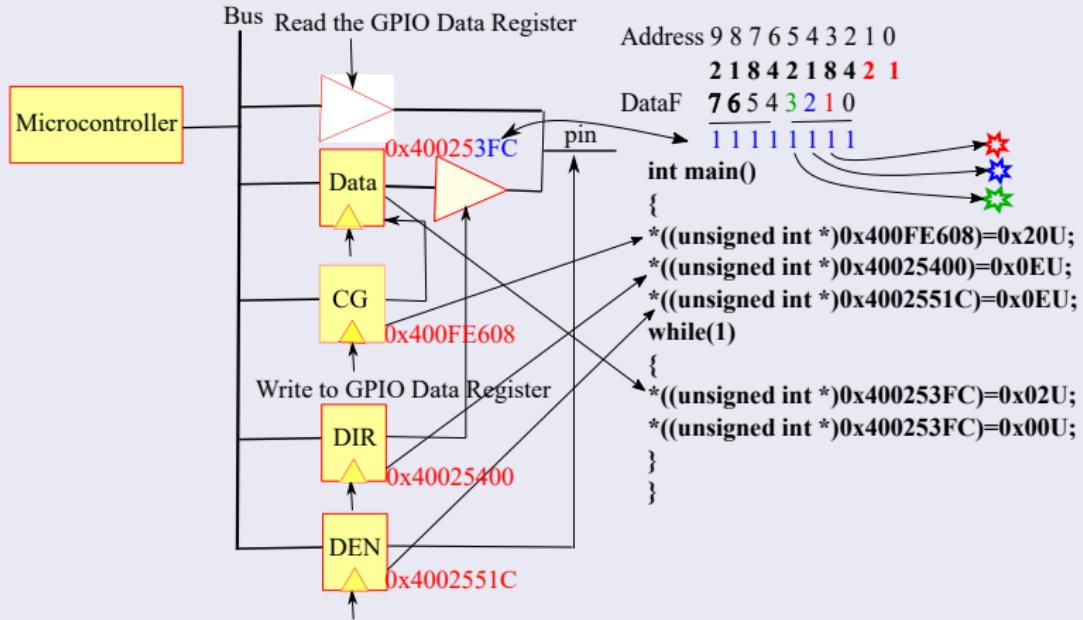
# Red LED Glow Program

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Simple Program





# PULLUP and PULL DOWN Resistance

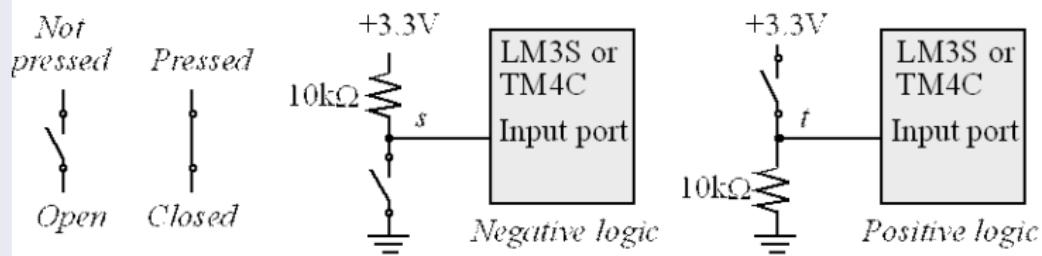
TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Switch Interfacing

- Pull-up and Pull-down resistors are used to correctly bias the inputs of digital gates to stop them from floating about randomly when there is no input condition





# PULLUP Register

## Register Information

- Pull up register is write protected
- To remove the write protection, we need GPIOCR to set before PULLUP Register

|          |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 31       | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Type     | RO |
| Reset    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 15       | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
| Type     | RO | RW |
| Reset    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

GPIO Pull-Down Select (GPIOPDR)

0:7 Bits

PortF PULL UP Register Set

0x40025514=0x10

GPIO Port A (APB) base: 0x4000.4000

GPIO Port A (AHB) base: 0x4005.8000

GPIO Port B (APB) base: 0x4000.5000

GPIO Port B (AHB) base: 0x4005.9000

GPIO Port C (APB) base: 0x4000.6000

GPIO Port C (AHB) base: 0x4005.A000

GPIO Port D (APB) base: 0x4000.7000

GPIO Port D (AHB) base: 0x4005.B000

GPIO Port E (APB) base: 0x4002.4000

GPIO Port E (AHB) base: 0x4005.C000

GPIO Port F (APB) base: 0x4002.5000

GPIO Port F (AHB) base: 0x4005.D000

Offset 0x514

Type RW, reset 0x0000.0000



# GPIOCR Register

## Register Information

- The GPIOCR register is the commit register.

| reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Type     | RO |
| Reset    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Type     | RO | RW |
| Reset    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| DIR      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

GPIO Commit (GPIOCR)

GPIO Port A (APB) base: 0x4000.4000  
GPIO Port A (AHB) base: 0x4005.8000  
GPIO Port B (APB) base: 0x4000.5000  
GPIO Port B (AHB) base: 0x4005.9000  
GPIO Port C (APB) base: 0x4000.6000  
GPIO Port C (AHB) base: 0x4005.A000  
GPIO Port D (APB) base: 0x4000.7000  
GPIO Port D (AHB) base: 0x4005.B000  
GPIO Port E (APB) base: 0x4002.4000  
GPIO Port E (AHB) base: 0x4005.C000  
GPIO Port F (APB) base: 0x4002.5000  
GPIO Port F (AHB) base: 0x4005.D000  
Offset 0x524  
Type -, reset -

0:7 Bits

Default value is 0xFF

0x40025524=0xFF

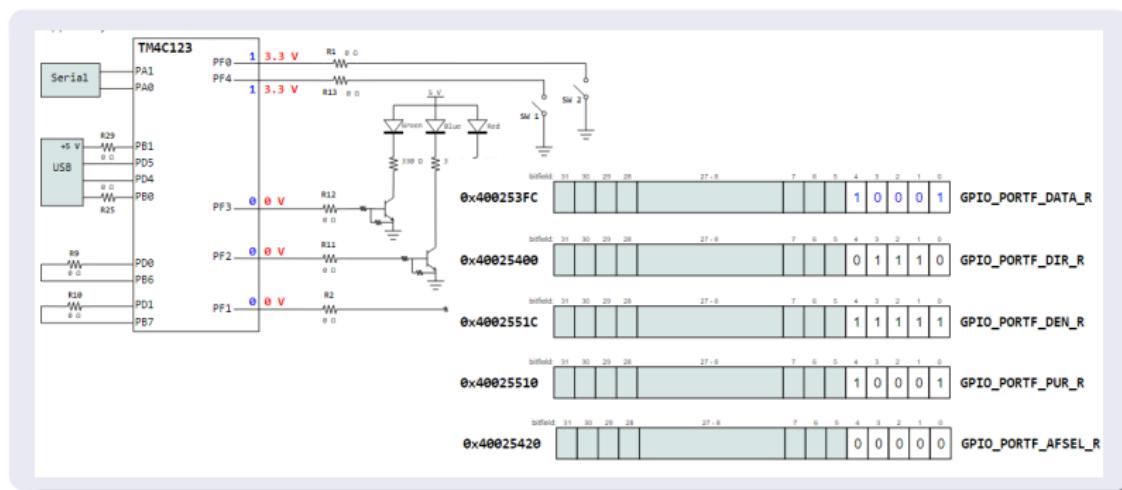
- 0 The corresponding **GPIOAFSEL**, **GPIOPUR**, **GPIOPDR**, or **GPIODEN** bits cannot be written.
- 1 The corresponding **GPIOAFSEL**, **GPIOPUR**, **GPIOPDR**, or **GPIODEN** bits can be written.



# Pin Diagram of TM4C123

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

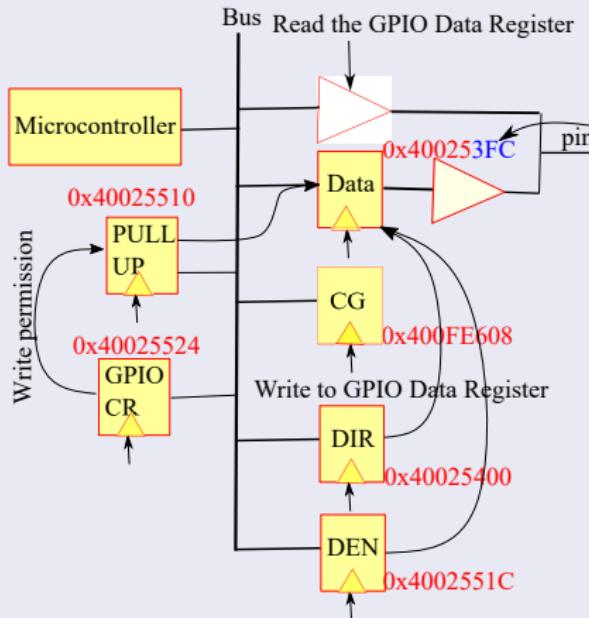
Dr. Munesh  
Singh





# Blue LED Control using Switch at Pin 4

## Simple Program



Address 9 8 7 6 5 4 3 2 1 0  
2 1 8 4 2 1 8 4 2 1  
7 6 5 4 3 2 1 0  
1 1 1 1 1 1 1 1 1 1  
locked

int main()  
{  
\*((unsigned int \*)0x400FE608)=0x20U;  
\*((unsigned int \*)0x40025400)=0x0EU;  
\*((unsigned int \*)0x4002551C)=0x1FU;  
\*((unsigned int \*)0x40025524)=0xFF;  
\*((unsigned int \*)0x40025510)=0x10;  
while(1)  
{  
if(\*((unsigned int \*)0x40025040)==0) {  
\*((unsigned int \*)0x40025010)=0x04;  
}  
else  
\*((unsigned int \*)0x40025010)=0x00;  
}  
return 0;  
}



# Unlock Special Function Pin

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Register Setting

- To enable the use of these pins, we have to set two registers **GPIOLOCK** and **GPIOCR**

Table 10-1. GPIO Pins With Special Considerations

| GPIO Pins | Default Reset State | GPIOAFSEL | GPIODEN | GPIOPDR | GPIOPUR | GPIOPCTL | GPIOCR |
|-----------|---------------------|-----------|---------|---------|---------|----------|--------|
| PA[1:0]   | UART0               | 0         | 0       | 0       | 0       | 0x1      | 1      |
| PA[5:2]   | SSIO                | 0         | 0       | 0       | 0       | 0x2      | 1      |
| PB[3:2]   | I <sup>2</sup> C0   | 0         | 0       | 0       | 0       | 0x3      | 1      |
| PC[3:0]   | JTAG/SWD            | 1         | 1       | 0       | 1       | 0x1      | 0      |
| PD[7]     | GPIO <sup>a</sup>   | 0         | 0       | 0       | 0       | 0x0      | 0      |
| PF[0]     | GPIO <sup>a</sup>   | 0         | 0       | 0       | 0       | 0x0      | 0      |

a. This pin is configured as a GPIO by default but is locked and can only be reprogrammed by unlocking the pin in the **GPIOLOCK** register and uncommitting it by setting the **GPIOCR** register.



# GPIO LOCK Register

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## GPIOLOCK

|       | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| LOCK  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Type  | RW |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| LOCK  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Type  | RW |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  |

0:31 Bits

To Unlock the Special Port  
 $0x40025520=0x4C4F434B$

Default value=0x4C4F434B

## GPIO Lock (GPIOLOCK)

GPIO Port A (APB) base: 0x4000.4000  
GPIO Port A (AHB) base: 0x4005.8000  
GPIO Port B (APB) base: 0x4000.5000  
GPIO Port B (AHB) base: 0x4005.9000  
GPIO Port C (APB) base: 0x4000.6000  
GPIO Port C (AHB) base: 0x4005.A000  
GPIO Port D (APB) base: 0x4000.7000  
GPIO Port D (AHB) base: 0x4005.B000  
GPIO Port E (APB) base: 0x4002.4000  
GPIO Port E (AHB) base: 0x4005.C000  
GPIO Port F (APB) base: 0x4002.5000  
GPIO Port F (AHB) base: 0x4005.D000  
Offset 0x520  
Type RW, reset 0x0000.0001



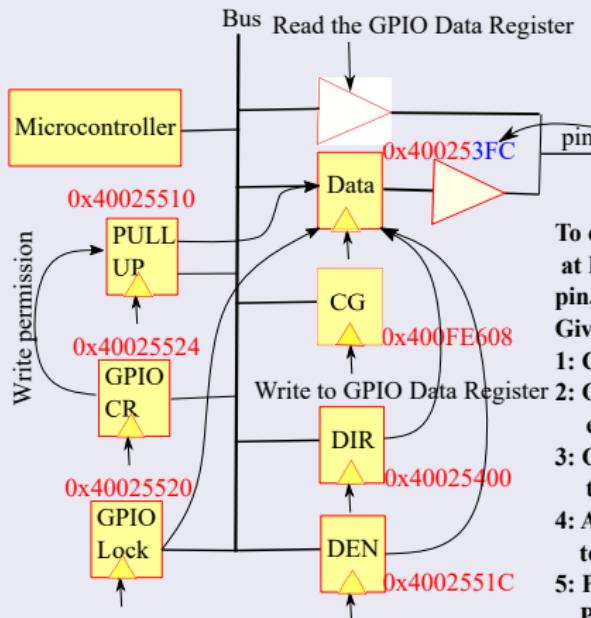
# Control RGB LED using On Board Switches

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Write a Program



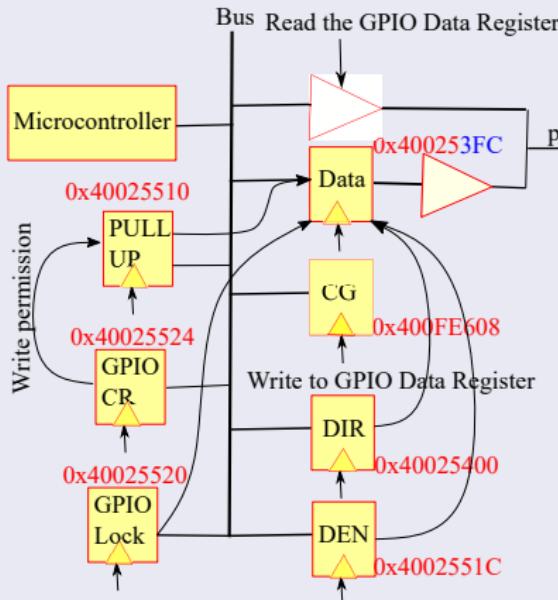
To control all the on Board LED using two switch at Port F write the program. For each pin, i need specific address to be calculated.  
Given

- 1: CG register 5th bit initialize the port memory
- 2: GPIOLock default value =0x4C4F434B to enable locked switch
- 3: GPIOCR register Default value=0xFF enable the write permission
- 4: All other register set the Data register bit to enable functions
- 5: PF0->sw1 (locked), PF4->sw2, PF1->Red, PF2->Blue, PF3->Green.



# Control all on Board LED using Switches

## Write a Program



```
int main()
{
    *((unsigned int *)0x400FE608)=0x20;
    *((unsigned int *)0x40025400)=0x0E;
    *((unsigned int *)0x4002551C)=0x1F;
    *((unsigned int *)0x40025520)=0x4C4F434B;
    *((unsigned int *)0x40025524)=0xFF;
    *((unsigned int *)0x40025510)=0x11;
    while(1) {
        if(*((unsigned int *)0x40025044)==0x00){
            *((unsigned int *)0x40025010)=0x04;
        }
        else if(*((unsigned int *)0x40025044)==0x10)
        {
            *((unsigned int *)0x40025008)=0x02;
            else if(*((unsigned int *)0x40025044)==0x01)
        {
            *((unsigned int *)0x40025020)=0x08;
        }
        else{
            *((unsigned int *)0x40025010)=0x00;
            *((unsigned int *)0x40025008)=0x00;
            *((unsigned int *)0x40025020)=0x00;
        }
    }
}
```



# Simplify the Coding # directive and Volatile

## #define volatile Datatype

- The earlier written program uses the dereferencing of pointer address to access the port memory
- Same can be defined with a new name of user choice using Preprocessor Directive
- In the address pointer type casting, we have to include volatile keyword
  - **volatile** keyword use with register where R/W permission of bit is allowed
  - It is used to tell the compiler that register object changes frequently
  - volatile is a qualifier
- **int** and **long** data type in Arm Instruction classes is of 32 Bit.



# Use of # directive and Volatile

## #define volatile Datatype

```
#define CG *((volatile unsigned int *)0x400FE608) // clock gating
#define DIR *((volatile unsigned int *)0x40025400)// direction register (b01110->portF
#define DEN *((volatile unsigned int *)0x4002551C) //digital enable (b11111)->portF
#define LOCK *((volatile unsigned int *)0x40025520)//Unlock the pinF[0]
#define CR *((volatile unsigned int *)0x40025524)//Commit register allow write permission
#define PULLUP *((volatile unsigned int *)0x40025510) //pull up register(b10001);
#define DATAF *((unsigned int *)0x40025044) //Data register of port F
#define RED_LED *((unsigned int *)0x40025008) // Red LED Address Pin
#define BLUE_LED *((unsigned int *)0x40025010) //Blue LED Address Pin
#define GREEN_LED *((unsigned int *)0x40025020) //Green LED Address Pin
int main()
{
    CG=0x20; DIR=0x0E; DEN=0x1F; LOCK=0x4C4F434B; CR=0xFF; PULLUP=0x11;
    while(1) {
        if(DATAF==0x00) {
            BLUE_LED=0x04;
        }
        else if(*((unsigned int *)0x40025044)==0x10) {
            RED_LED=0x02;
        }
        else if(*((unsigned int *)0x40025044)==0x01) {
            GREEN_LED=0x08;
        }
        else{
            RED_LED=0x00;
            BLUE_LED=0x00;
            GREEN_LED=0x00;
        }
    }
    return 0;
}
```



# Use of Bitwise Operator of C

## Bitwise Operators

- $c = a | b$ ; OR
- $c = a \& b$ ; AND
- $c = a \wedge b$ ; Exclusive OR
- $c = b >> 1$ ; right shift
- $c = b << 1$ ; left shift
- $c = \neg b$ ; NOT
- Lower bit hex calculation is easier, but high order bit calculation of hex is time taking
- Let say i want to set bit 1 of GIPOF Data ports For RED, BLUE, GREEN LED.
  - `#define RED (1U<<2)`
  - `#define BLUE (1U<<3)`
  - `#define GREEN (1U<<4)`



# Use of Bitwise Operators

TM4C123GH6P

Micro-  
controllers

Programming  
Concepts

Dr. Munesh  
Singh

## Bitwise

```
#define CG *((volatile unsigned int *)0x400FE608) // clock gating
#define DIR *((volatile unsigned int *)0x40025400)// direction register (b01110)->portF
#define DEN *((volatile unsigned int *)0x4002551C) //digital enable (b11111)->portF
#define DATAF *((volatile unsigned int *)0x400253FC) //Data register of port F
#define RED (1U<<1)
#define BLUE (1U<<2)
#define GREEN (1U<<3)
int main()
{
    CG=0x20;
    DIR=0xE; // DIR|=(RED|BLUE|GREEN) = DIR|=((1U<<1)|(1U<<2)|(1U<<3))
    DEN=0xE; // DEN|=(RED|BLUE|GREEN)= DEN|=((1U<<1)|(1U<<2)|(1U<<3))
    while(1)
    {
        DATAF|=RED; // DATAF|=(1U<<2)
        DATAF&=~RED; //DATAF&=~(1U<<2)
    }
    return 0;
}
```

OR to Set Bit

|        |   |   |   |   |   |   |
|--------|---|---|---|---|---|---|
| XXXXXX | T | X |   |   |   |   |
| 0      | 0 | 0 | 0 | 0 | 1 | 0 |
| XXXXXX | 1 | X |   |   |   |   |

reg  
mask  
reg set

AND to Clear

|        |   |   |   |   |   |   |
|--------|---|---|---|---|---|---|
| XXXXXX | T | X |   |   |   |   |
| 1      | 1 | 1 | 1 | 1 | 1 | 0 |
| XXXXXX | 0 | X |   |   |   |   |

reg  
mask  
reg clear



# Read/Write/Modify Issue During the Interrupt

## Bitwise Operators

- Read/write/modify is fast enough in most cases
- But during the interrupt the read/modify/write operation stuck the current execution and program counter jump to the new instruction address.
- During the interrupt service routine GPIO DATA register bits may be modified.
- After the completion of ISR, the processor resume its current execution.
- Resume instruction further changes the GPIO DATA bits, which lost the ISR bit change
- Through the pointer arithmetic this problem can be minimized to a certain extend



# Array and Pointer

## Bitwise Operators

- Arm instruction read/write/modify instruction is replaced with an atomic write operation
- Each bit of GPIO DATA register is addressable ( 8 bit GPIO DATA register is addressed by 256 registers)
- If we write the instruction to write the bit at the particular address, then we can access that address through pointer arithmetic or array base address



# Array and Pointer Use

## Bitwise Operators

```
#define CG *((unsigned int *)0x400FE608U) //clock gating
#define DIR *((unsigned int *)0x40025400U) // Direction register
#define DEN *((unsigned int *)0x4002551CU) //Digital enable function
#define GPIODATAF *((unsigned int *)0x400253FCU) // GPIOF Register
#define GPIODATABITF ((unsigned int *)0x40025000U)//Base address of GPIOF Register
#define RED_LED (1U<<1)
#define BLUE_LED (1U<<2)
#define GREEN_LED (1U<<3)
```

```
int main()
{
    CG=0x20U;
    DIR=0x0EU;
    DEN=0x0EU;
    while(1){
        int count=0;
        //GPIODATAF|=RED_LED;
        *(GPIODATABITF+RED_LED)=RED_LED;
        // GPIODATABITF[RED_LED]=RED_LED;

        while(count<1000000){
            ++count;
        }
        count=0;
        GPIODATAF&=~RED_LED;
        while(count<1000000){
            ++count;
        }
        return 0;
    }
}
```

|                     |        |                 |
|---------------------|--------|-----------------|
| 0x53: 0x480d        | LDR.N  | R0, [PC, #0x34] |
| 0x54: 0x6801        | LDI    | R1, [R0]        |
| 0x56: 0xf051 0x0102 | ORRS.W | R1, R1, #2      |
| 0x5a: 0x6001        | STR    | R1, [R0]        |

|   |              |       |                     |
|---|--------------|-------|---------------------|
| *(GPIODATABITF+RED_LED)=RED_LED; // GPIODATABITF[RED_L... | 0x54: 0x2102 | MRS   | R1, #2              |
|   | 0x56: 0x4a0b | LDR.N | R2, [PC, #0x2c] ... |
|   | 0x58: 0x6011 | STR   | R1, [R2]            |



# Function and Stack

## Eliminate the Repaeated Code

```
#define CG *((unsigned int *)0x400FE608U) //clock gating
#define DIR *((unsigned int *)0x40025400U) // Direction register
#define DEN *((unsigned int *)0x4002551CU) //Digital enable function
#define GPIODATABITF ((unsigned int *)0x40025000U) //Base address of GPIOF Register
#define RED_LED (1U<<1)
#define BLUE_LED (1U<<2)
#define GREEN_LED (1U<<3)

void delay() {
    int volatile counter=0;
    while(counter<1000000){
        ++counter; } }

int main()
{
    CG|=(1U<<5);
    DIR|=(RED_LED|BLUE_LED|GREEN_LED);
    DEN|=(RED_LED|BLUE_LED|GREEN_LED);
    while(1){
        GPIODATABITF[BLUE_LED]=BLUE_LED;
        delay();
        GPIODATABITF[BLUE_LED]=~BLUE_LED;
        delay();
    }
    return 0;
}
```

|                                   |                  |
|-----------------------------------|------------------|
| PC                                | 0x00000080       |
| LR                                | 0x00000085       |
| delay():                          |                  |
| 0x80: 0xf7ff 0xffde               | BL delay         |
| GPIODATABITF[BLUE_LED]=~BLUE_LED; |                  |
| 0x84: 0xf07f 0x0004               | MVNS.W R0, #4    |
| 0x88: 0x6020                      | STR R0, [R4]     |
| SP                                | 0x20000FF8       |
| delay:                            |                  |
| 0x40: 0xb081                      | SUB SP, SP, #0x4 |
| 0x56: 0xb001                      | ADD SP, SP, #0x4 |
| 0x58: 0x4770                      | BX LR            |



# GPIO Timers

- General-Purpose Timer Module (GPTM) contains six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks
- The GPT Module is one timing resource other System Timer (SysTick) and the PWM timer in the PWM modules
- GPIO Timer operates in different modes
  - One short/periodic mode
  - Real time clock timer mode
  - Input edge count mode
  - Input edge time mode
  - PWM mode
  - Wait for trigger mode



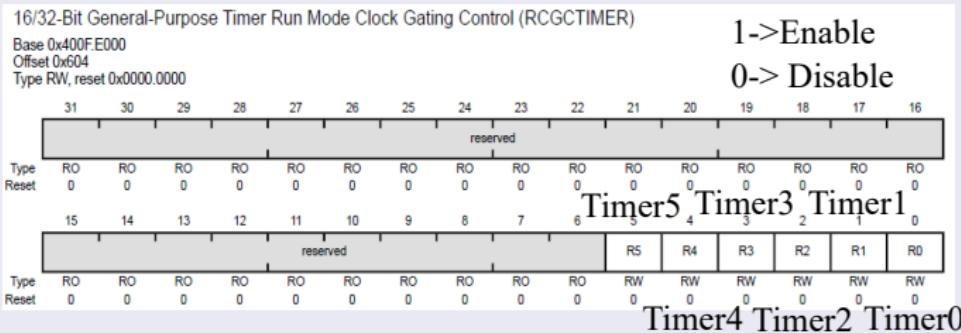
# GPIO Timers Clock Gating Register

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

- General-Purpose Timer Run Mode Clock Gating Control (RCGCTIMER)





# GPIO Timers Initialization

## One Short/ Periodic Timer

```
#include"tm4c123gh6pm.h"
#define CG *((volatile unsigned int *)0x400FE608) // clock gating
#define DIR *((volatile unsigned int *)0x40025400)// direction register (b01110)->portF
#define DEN *((volatile unsigned int *)0x4002551C) //digital enable (b11111)->portF
#define DATAF *((unsigned int *)0x400253FC) //Data register of port F
#define DATABITF ((volatile unsigned int *)0x40025000) //Data register of port F
int main()
{
    CG=0x20;
    SYSTCL_RCGCTIMER_R|=(1U<<0); //clock gating on timer circuit
    TIMER0_CTL_R &=~(1<<0); // disable the timer0
    TIMER0_CFG_R= 0x00000000; // configure resiter for periodic
    TIMER0_TAMR_R|= (0x2<<0); // configure periodic mode
    TIMER0_TAMR_R&= ~(1<<4); // configure down counter
    TIMER0_TAILR_R= 0x00F42400; // set the value 16000000
    TIMER0_CTL_R|= (1<<0); // enable the timer0
    DIR|=((1U<<1)|(1U<<2)|(1U<<3)) //set the direction of port
    DEN|=((1U<<1)|(1U<<2)|(1U<<3)) //enable the digital function
    while(1)
    {
        if((TIMER0_RIS_R & 0x00000001) == 1)//0 TIMER A HAS NOT TIME OUT
        {
            TIMER0_ICR_R |= (1<<0);
            DATAF^= (1<<2);
        }
    }
    return 0;
}
```



# Exception Handler

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Types of Exception

- Nested Vectored Interrupt Controller (NVIC) prioritize and handle all exceptions in Handler Mode
- The processor state is automatically stored to the stack on an exception and automatically restored from the stack at the end of the Interrupt Service Routine (ISR)
- Exception States:
  - Inactive
  - Pending
  - Active
  - Active and Pending



# Interrupt Vector Table

| Exception number | IRQ number | Offset | Vector                  |
|------------------|------------|--------|-------------------------|
| 154              | 138        | 0x0268 | IRQ131                  |
| .                | .          | .      | .                       |
| 18               | 2          | 0x004C | IRQ2                    |
| 17               | 1          | 0x0048 | IRQ1                    |
| 16               | 0          | 0x0044 | IRQ0                    |
| 15               | -1         | 0x0040 | Systick                 |
| 14               | -2         | 0x003C | PendsV                  |
| 13               |            | 0x0038 | Reserved                |
| 12               |            |        | Reserved for Debug      |
| 11               | -5         | 0x002C | SVCall                  |
| 10               |            |        | Reserved                |
| 9                |            |        | Reserved                |
| 8                |            |        |                         |
| 7                |            |        |                         |
| 6                | -10        | 0x0018 | Usage fault             |
| 5                | -11        | 0x0014 | Bus fault               |
| 4                | -12        | 0x0010 | Memory management fault |
| 3                | -13        | 0x000C | Hard fault              |
| 2                | -14        | 0x0008 | NMI                     |
| 1                |            | 0x0004 | Reset                   |
|                  |            | 0x0000 | Initial SP value        |

generated by user  
program or peripheral

set timer generated  
software interrupt (ICSR)

debug when no halting  
supervised from OS to kernel

user generated fault  
memory related fault  
memory protection  
error during exception processing  
software interrupt (ICSR)  
Reset the processor



# Types of Exception Handler

## Exception handlers

- Processor handles the exception using
  - **Interrupt Service Routines (ISRs)** Interrupts (IRQx) are the exceptions handled by ISRs.
  - **Fault Handlers** Hard fault, memory management fault, usage fault, and bus fault are fault exceptions handled by the fault handlers.
  - **System Handlers** NMI, PendSV, SVCall, SysTick, and the fault exceptions are all system exceptions that are handled by system handlers.



# Interrupt Vector Table Entry

| Vector Number | Interrupt Number (Bit in Interrupt Registers) | Vector Address or Offset  | Description             |
|---------------|---|---------------------------|-------------------------|
| 0-15          | -   | 0x0000.0000 - 0x0000.003C | Processor exceptions    |
| 16            | 0   | 0x0000.0040               | GPIO Port A             |
| 17            | 1   | 0x0000.0044               | GPIO Port B             |
| 18            | 2   | 0x0000.0048               | GPIO Port C             |
| 19            | 3   | 0x0000.004C               | GPIO Port D             |
| 20            | 4   | 0x0000.0050               | GPIO Port E             |
| 21            | 5   | 0x0000.0054               | UART0                   |
| 22            | 6   | 0x0000.0058               | UART1                   |
| 23            | 7   | 0x0000.005C               | SSI0                    |
| 24            | 8   | 0x0000.0060               | I <sup>2</sup> C0       |
| 25            | 9   | 0x0000.0064               | PWM0 Fault              |
| 26            | 10  | 0x0000.0068               | PWM0 Generator 0        |
| 27            | 11  | 0x0000.006C               | PWM0 Generator 1        |
| 28            | 12  | 0x0000.0070               | PWM0 Generator 2        |
| 29            | 13  | 0x0000.0074               | QEIO                    |
| 30            | 14  | 0x0000.0078               | ADC0 Sequence 0         |
| 31            | 15  | 0x0000.007C               | ADC0 Sequence 1         |
| 32            | 16  | 0x0000.0080               | ADC0 Sequence 2         |
| 33            | 17  | 0x0000.0084               | ADC0 Sequence 3         |
| 34            | 18  | 0x0000.0088               | Watchdog Timers 0 and 1 |
| 35            | 19  | 0x0000.008C               | 16/32-Bit Timer 0A      |
| 36            | 20  | 0x0000.0090               | 16/32-Bit Timer 0B      |



# Interrupt Vector Table Entry in cstartup\_M

- There are four 32 bit interrupt registers, that maps 128 types of interrupts.
- Initialization steps for interrupt:
  - In cstartup\_M.c file we add the declaration of interrupt function
  - Add the function name in the interrupt table with correct order
  - Order of the interrupt gets in the interrupt vector table
  - Lastly, function definition.



# Function Addition in cstartup\_M.c vector file

## Function declaration

Debug

| Files           |   |
|-----------------|---|
| Lesson0 - Debug | • |
| core_cm4.h      |   |
| cstartup_M.c    | ✓ |
| main.c          |   |
| tm4c123gh6pm.h  |   |
| Output          |   |

```
#pragma language=extended
#pragma segment="CSTACK"

extern void __iar_program_start( void );

extern void NMI_Handler( void );
extern void HardFault_Handler( void );
extern void MemManage_Handler( void );
extern void BusFault_Handler( void );
extern void UsageFault_Handler( void );
extern void SVC_Handler( void );
extern void DebugMon_Handler( void );
extern void PendSV_Handler( void );
extern void SysTick_Handler( void );
extern void ADC1SS3_Handler(void);
extern void TIMER0A_Handler( void ); function declaration
typedef void( *intfunc )( void );
typedef union { intfunc __fun; void * __ptr; } intvec_elem;
```



## Function Entry in Vector Table



# Function Addition in cstartup\_M.c vector file

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Function Definition

| Debug           |
|-----------------|
| Files           |
| Lesson0 - Debug |
| core_cm4.h      |
| cstartup_M.c    |
| main.c          |
| tm4c123gh6pm.h  |
| Output          |

```
#pragma call_graph_root = "interrupt"
__weak void BusFault_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void UsageFault_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void SVC_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void DebugMon_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void PendSV_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void SysTick_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void ADC1SS3_Handler( void ) { while (1) {} }

#pragma call_graph_root = "interrupt"
__weak void TIMER0A_Handler( void ) { while (1) {} }
```



# Timer Interrupt Based LED Blink

## Complete Program

```
#include"tm4c123gh6pm.h"
#define CG *((volatile unsigned int *)0x400FE608) // clock gating
#define DIR *((volatile unsigned int *)0x40025400)// direction register (b01110)->portF
#define DEN *((volatile unsigned int *)0x4002551C) //digital enable (b11111)->portF
#define DATAF *((unsigned int *)0x400253FC) //Data register of port F
#define DATABITF ((volatile unsigned int *)0x40025000) //Data register of port F

void TIMER0A_Handler( void )
{
    TIMER0_ICR_R |= (1<<0);
    DATAF^= (1<<2);
}

int main()
{
    CG=0x20;
    SYSCTL_RCGCTIMER_R|=(1U<<0); //clock gating on timer circuit
    TIMER0_CTL_R &=~(1<<0); // disable the timer0
    TIMER0_CFG_R= 0x00000000; // configure resiter for periodic
    TIMER0_TAMR_R|=(0x2<<0); // periodic mode timer
    TIMER0_TAMR_R&= ~(1<<4); // configure down counter
    TIMER0_TAILR_R= 0x00F42400; // set the value 16000000
    TIMER0_IMR_R|=(1<<0); // enable the interrupt for timer0
    NVIC_EN0_R |=(1<<19); //Enter the entry in NVIC Register
    TIMER0_CTL_R|=(1<<0); // enable the timer0

    DIR|=((1U<<1)|(1U<<2)|(1U<<3)) // Set the direction of GPIOF Port
    DEN|=((1U<<1)|(1U<<2)|(1U<<3)) //Enable Digital Function of GPIOF Port
    while(1) {
    }
    return 0;
}
```

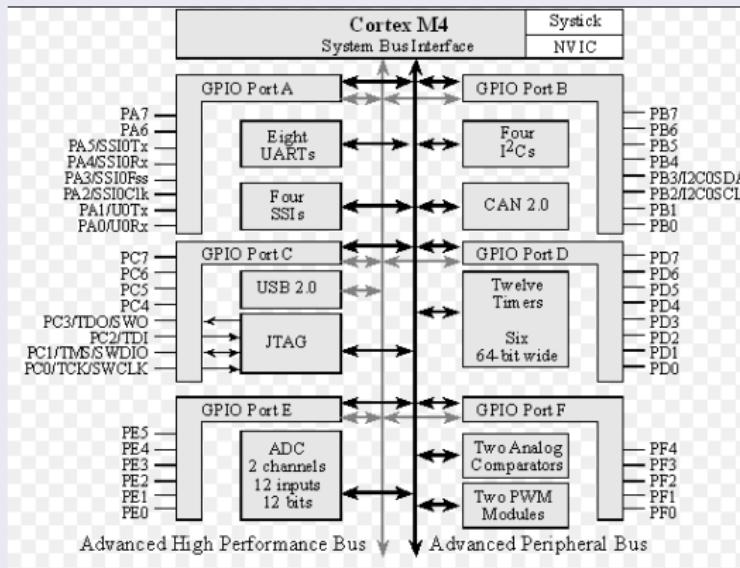


# Analog To Digital Convertors

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

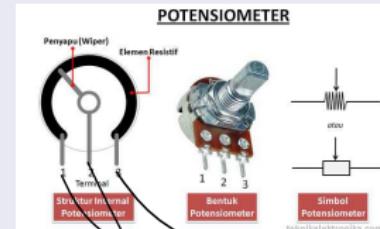




# Analog To Digital Convertors Connections

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh



12 bit ADC

Range=0 to  $(2^{12}-1)=0$  to 4095=(0-3.3v)

Resolution=(3.3/4096) = 0.8 mv





# Analog To Digital Convertors Process

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

- An analog-to-digital converter (ADC) is a peripheral that converts a continuous analog voltage to a discrete digital number.
- Important Terms
  - Amplitude
  - Amplitude Range
  - Time Quantization
  - Time Interval
- A 12-bit ADC converts 0 to 3.3V on its input into a digital number from 0 to 4095.
- 12 Bit Successive Approximation



# Initialization of ADC Port

- Enable the ADC clock using the RCGCADC register.

Analog-to-Digital Converter Run Mode Clock Gating Control (RCGCADC)

Base 0x400F.E000

Offset 0x638

Type RW, reset 0x0000.0000

|          | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Type     | RO |
| Reset    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Type     | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
| Reset    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | R1 | R0 |

## Value Description

0 ADC module 0 is disabled.

1 Enable and provide a clock to ADC module 0 in Run mode.



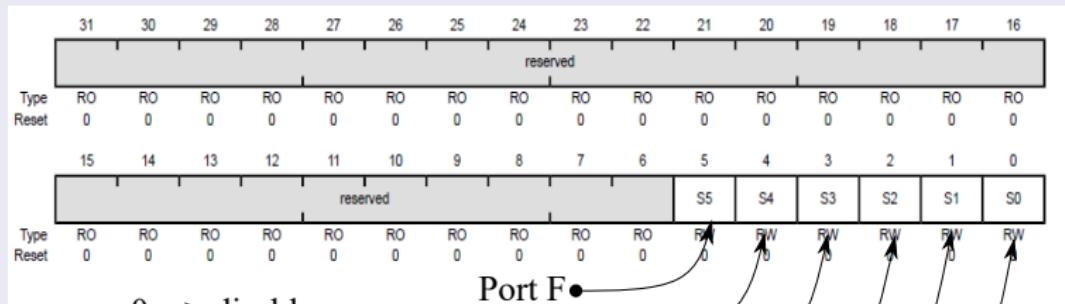
# Initialization of GPIO PORT

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

- Enable the clock to the appropriate GPIO modules via the RCGCGPIO register



0---> disable

1---> enable

Port F

Port E

Port D

Port C

Port B

Port A



# GPIODIR Direction Register

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Register Information

|          | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Type     | RO |
| Reset    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Type     | RO | RW |
| Reset    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| DIR      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

0---> Corresponding pin input

1---> Corresponding pin output

):7 Bits

### GPIO Direction (GPIODIR)

GPIO Port A (APB) base: 0x4000.4000

GPIO Port A (AHB) base: 0x4005.8000

GPIO Port B (APB) base: 0x4000.5000

GPIO Port B (AHB) base: 0x4005.9000

GPIO Port C (APB) base: 0x4000.6000

GPIO Port C (AHB) base: 0x4005.A000

GPIO Port D (APB) base: 0x4000.7000

GPIO Port D (AHB) base: 0x4005.B000

GPIO Port E (APB) base: 0x4002.4000

GPIO Port E (AHB) base: 0x4005.C000

GPIO Port F (APB) base: 0x4002.5000

GPIO Port F (AHB) base: 0x4005.D000

Offset 0x400

Type RW, reset 0x0000.0000



# GPIODEN Digital Function Register

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Register Information

|          |    |    |    |    |    |    |    |     |    |    |    |    |    |    |    |
|----------|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|
| 31       | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23  | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| reserved |    |    |    |    |    |    |    |     |    |    |    |    |    |    |    |
| Type     | RO  | RO | RO | RO | RO | RO | RO | RO |
| Reset    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| reserved |    |    |    |    |    |    |    |     |    |    |    |    |    |    |    |
| 15       | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7   | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
| Type     | RO  | RW |
| Reset    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| DIR      |    |    |    |    |    |    |    | DIR |    |    |    |    |    |    |    |

## GPIO Digital Enable (GPIODEN)

0---> Disable Digital function  
1---> Enable Digital function

0:7 Bits

GPIO Port A (APB) base: 0x4000.4000

GPIO Port A (AHB) base: 0x4005.8000

GPIO Port B (APB) base: 0x4000.5000

GPIO Port B (AHB) base: 0x4005.9000

GPIO Port C (APB) base: 0x4000.6000

GPIO Port C (AHB) base: 0x4005.A000

GPIO Port D (APB) base: 0x4000.7000

GPIO Port D (AHB) base: 0x4005.B000

GPIO Port E (APB) base: 0x4002.4000

GPIO Port E (AHB) base: 0x4005.C000

GPIO Port F (APB) base: 0x4002.5000

GPIO Port F (AHB) base: 0x4005.D000

Offset 0x51C

Type RW, reset -



## GPIO Alternate Function Select (GPIOAFSEL)

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

## Register Information

|       | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23       | 22 | 21 | 20 | 19 | 18 | 17 | 16 |    |  |  |  |  |  |  |
|-------|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|----|----|----|--|--|--|--|--|--|
| Type  | RO | reserved |    |    |    |    |    |    |    |    |  |  |  |  |  |  |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0        | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |  |  |  |  |  |  |
|       | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7        | 6  | 5  | 4  | 3  | 2  | 1  | 0  |    |  |  |  |  |  |  |
| Type  | RO | reserved |    |    |    |    |    |    |    |    |  |  |  |  |  |  |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 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  |    |  |  |  |  |  |  |
| Type  | RO | AFSEL    |    |    |    |    |    |    |    |    |  |  |  |  |  |  |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | -        | -  | -  | -  | -  | -  | -  | -  | -  |  |  |  |  |  |  |
|       | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7        | 6  | 5  | 4  | 3  | 2  | 1  | 0  |    |  |  |  |  |  |  |

## GPIO Alternate Function Select (GPIOAFSEL)

GPIO Port A (APB) base: 0x4000\_4000

GPIO Port A (AHB) base: 0x4005.8000

GPIO Port B (APB) base: 0x4000.5000

GPIO Port B (AHB) base: 0x4005.9000

GPIO Port C (APB) base: 0x4000.6000

GPIO Port C (AHB) base: 0x4005.A000

GPIO Port D (APB) base: 0x4000.7000

GPIO Port D (AHB) base: 0x4005.B000

GPIO Port E (APB) base: 0x40024000

GPIO Port E (AHB) base: 0x4005.C000

GPIO Port F (APB) base: 0x4002\_5000

GPIO Port F (AHB) base: 0x4005.D000  
Offset 0x420

Type RW reset -

Type KW, Reset -

Digitized by srujanika@gmail.com

- 0 The associated pin functions as a GPIO and is controlled by the GPIO registers.
  - 1 The associated pin functions as a peripheral signal and is controlled by the alternate hardware function.

The reset value for this register is 0x0000.0000 for GPIO ports that are not listed in Table 10-1 on page 650.



# GPIO Analog Mode Select (GPIOAMSEL)

## Register Information

| reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Type     | RO |
| Reset    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Type     | RO | RW |
| Reset    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

### GPIO Analog Mode Select (GPIOAMSEL)

GPIO Port A (APB) base: 0x4000\_4000

GPIO Port A (AHB) base: 0x4005\_8000

GPIO Port B (APB) base: 0x4000\_5000

GPIO Port B (AHB) base: 0x4005\_9000

GPIO Port C (APB) base: 0x4000\_6000

GPIO Port C (AHB) base: 0x4005\_A000

GPIO Port D (APB) base: 0x4000\_7000

GPIO Port D (AHB) base: 0x4005\_B000

GPIO Port E (APB) base: 0x4002\_4000

GPIO Port E (AHB) base: 0x4005\_C000

GPIO Port F (APB) base: 0x4002\_5000

GPIO Port F (AHB) base: 0x4005\_D000

Offset 0x528

Type RW, reset 0x0000.0000

#### Value Description

0 The analog function of the pin is disabled, the isolation is enabled, and the pin is capable of digital functions as specified by the other GPIO configuration registers.

1 The analog function of the pin is enabled, the isolation is disabled, and the pin is capable of analog functions.



# Analog Input with GPIOE Port

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Register Information

| Pin Name | Pin Number | Pin Mux / Pin Assignment | Pin Type | Buffer Type <sup>a</sup> | Description                          |
|----------|------------|--------------------------|----------|--------------------------|--------------------------------------|
| AIN0     | 6          | PE3                      | I        | Analog                   | Analog-to-digital converter input 0. |
| AIN1     | 7          | PE2                      | I        | Analog                   | Analog-to-digital converter input 1. |
| AIN2     | 8          | PE1                      | I        | Analog                   | Analog-to-digital converter input 2. |
| AIN3     | 9          | PE0                      | I        | Analog                   | Analog-to-digital converter input 3. |
| AIN4     | 64         | PD3                      | I        | Analog                   | Analog-to-digital converter input 4. |
| AIN5     | 63         | PD2                      | I        | Analog                   | Analog-to-digital converter input 5. |
| AIN6     | 62         | PD1                      | I        | Analog                   | Analog-to-digital converter input 6. |
| AIN7     | 61         | PD0                      | I        | Analog                   | Analog-to-digital converter input 7. |
| AIN8     | 60         | PE5                      | I        | Analog                   | Analog-to-digital converter input 8. |



# Sample Sequencier

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

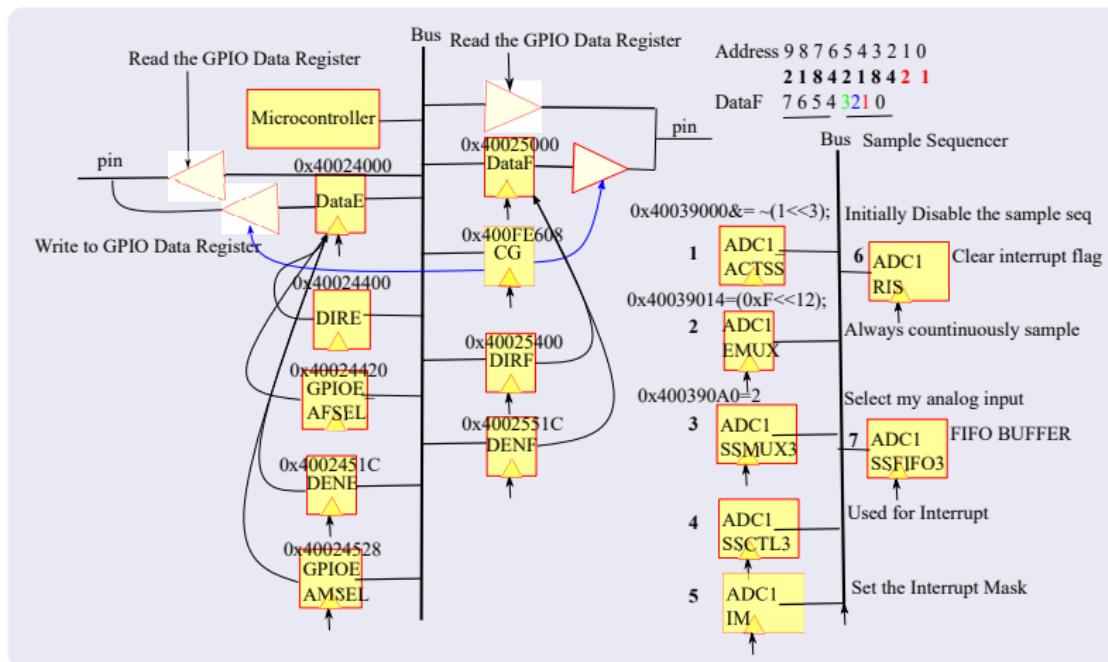
Dr. Munesh  
Singh

## Types of Sample Sequencier

| Sequencer | Number of Samples | Depth of FIFO |
|-----------|-------------------|---------------|
| SS3       | 1                 | 1             |
| SS2       | 4                 | 4             |
| SS1       | 4                 | 4             |
| SS0       | 8                 | 8             |



# Initialization of Analog to Digital Convertor





# Program to Control the LED Using Potentiometer

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

```
#include "lm4f120h5qr.h"
volatile static uint32_t adcResult = 0;
void ADC1SS3_Handler(void){
    adcResult = ADC1->SSFIFO3;
    ADC1->ISC = (1<<3); }

int main()
{
    SYSCTL->RCGCADC = (1<<1); //Step 1
    SYSCTL->RCCGCGPIO = (1<<4)|(1<<5); //Step 2
    GPIOE->DIR &= ~(1<<1);
    GPIOF->DEN = 0xFF; //Enable digital functions for the corresponding pin
    GPIOF->AFSEL = 0x00; //Disable alternate functions, we are using digital
    GPIOF->DIR = 0xFF; //Setting the pins makes them an output
    GPIOF->DATA = (1<<1);
    GPIOE->AFSEL = (1<<1); //Step 3
    GPIOE->DEN &= ~(1<<1); //Step 4
    GPIOE->AMSEL = (1<<1); //Step 5
    ADC1->ACTSS &= ~(1<<3); //Step 1 sample sequencer setting
    ADC1->EMUX = (0xF<<12); //Step 2
    ADC1->SSMUX3 = 2; //Step 3
    ADC1->SSCTL3 = 0x6; //Step 4
    ADC1->IM = (1<<3); //Step 5
    ADC1->ACTSS |= (1<<3); //Step 6
    ADC1->ISC = (1<<3);
    NVIC_EnableIRQ(ADC1SS3_IRQn);

    while(1) {
        if(adcResult > 2047){
            GPIOF->DATA |= (1<<1);}
```



## Interrupt Setting in Vector Table

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

## cstartup\_M.c vector file



# UART Communication To Communicate with Computer

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh



No.of bits transmitted per second from sender to receiver.

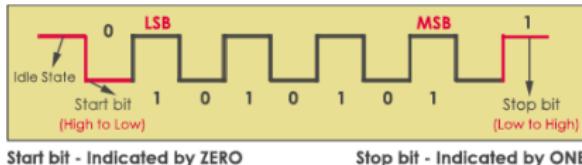
**Rule 1:** Baud Rate



**Rule 2:** Data length selection



**Rule 3:** Synchronization



**Rule 4:** Error Checking

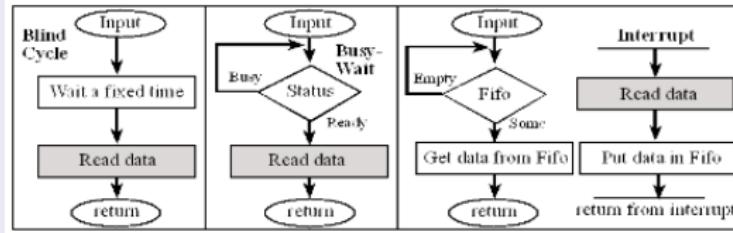
Parity bit is '1' for even number of binary ones and '0' for odd number of binary ones. According to rule 3 it is set to 1.

Codrey Electronics



# I/O Synchronization

- **Blind Cycle** is a method where the software simply wait for fixed amount of time
- **Busy-Wait** is a software loop that checks the I/O status waiting for the done state.
- **Interrupt** uses hardware to cause special software execution. With input device hardware will request when input device has new data.





# I/O Synchronization

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

- **Periodic Polling** use a clock interrupt to periodically check the I/O status.
  - With an input device, a ready flag is set when input device has new data.
  - At the next periodic interrupt after the input flag is set, the software will read the data and save into global RAM.
  - With the output device, a ready flag is set when output device is idle.
  - At the next periodic interrupt after the ready flag is set, the software will read the data from the global RAM



# I/O Synchronization

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

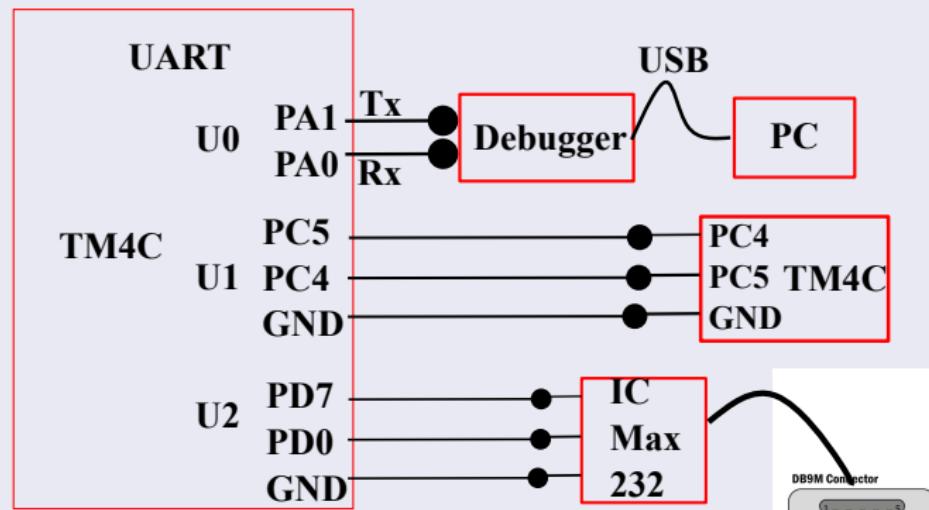
- **DMA** is an interface approach that transfer data directly to/from memory
  - With an input device, the hardware request to DMA transfer when input device has new data.
  - Without the software knowledge and permission data read and save in global RAM
  - With an output device, the hardware will request a DMA transfer when the output device is idle.
  - The DMA controller will get data from memory, and then write it to the device



# UART Communication Port On TM4C

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh





# UART Communication

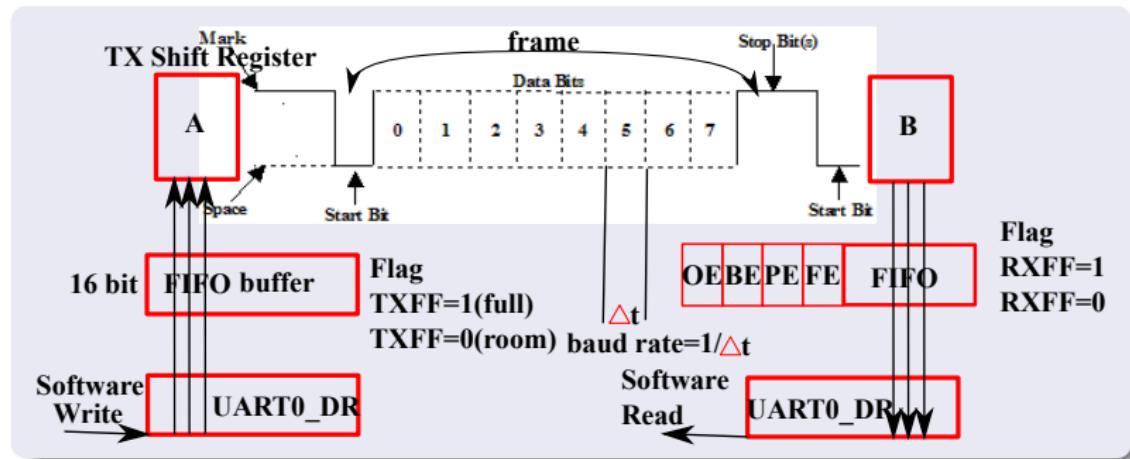
- Software write the data into the USRT0\_DR register
- Data later be transferred to FIFO buffer before it transfer to the TX shift register
- Later the each bit is transferred
- At the receiving end these bits are further transferred to the FIFO buffer.
  - **Overrun Error(OE)** OE, is set when input data are lost because the FIFO is full and more input frames are arriving at the receiver.
  - **Break Error(BE)** The break error, BE, is set when the input is held low for more than a frame.
  - **Parity Error(PE)** PE bit is set on a parity error.
  - **Frame Error(FE)** The framing error, FE, is set when the stop bit is incorrect



# UART Communication

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh





# UART Communication Port Configuration

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

The image displays five windows illustrating the setup process:

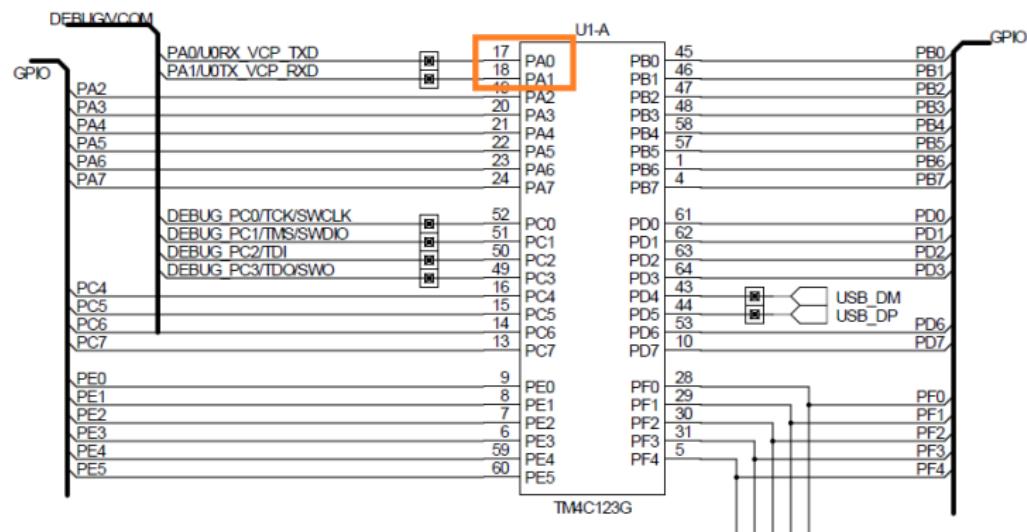
- Alternative binary files** (1): A download page for Putty.exe, stating "The installer packages above will provide all of these (except PuTTYtel), but you can download them individually." It lists 32-bit and 64-bit versions with download links and signatures.
- Putty Configuration** (2): The configuration dialog for Putty, showing the "Session" category selected. It asks for a "Host Name (or IP address)" and "Port" (set to 22). The "Connection type" is set to SSH. There are buttons for "Load", "Save", and "Delete".
- Device Manager** (3): The Windows Device Manager interface. The "Ports (COM & LPT)" section is expanded, showing "Stellaris Virtual Serial Port (COM3)". Other options like "Intel(R) Dual Band Wireless-AC 8265" and "Reutek PCIe IEEE Family Controller" are also visible.
- Stellaris Virtual Serial Port (COM3) Properties** (4): The properties dialog for the Stellaris Virtual Serial Port. It shows settings for "Bits per second" (9600), "Data bits" (8), "Parity" (None), "Stop bits" (1), and "Flow control" (None). Buttons for "Advanced...", "Restore Defaults", "OK", and "Cancel" are at the bottom.
- COM3 - PuTTY** (5): The PuTTY terminal window showing a black screen, indicating the connection is established.



# UART Communication Port On TM4C

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh





# UART Communication Port On TM4C

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## TOP VIEW



**Energia**

Flash 256 KB  
SRAM 32 KB

| I2C      |             |
|----------|-------------|
| CS (0)   | I2C SDA (0) |
| RX (0)   | A11         |
| TX (1)   | PB_5        |
| RX (0)   | PB_0        |
| TX (1)   | PB_1        |
| RX (0)   | SCL (2)     |
| TX (1)   | SDA (2)     |
| SCK (2)  | A9          |
| MOSI (0) | A8          |
| SCL (1)  | PE_4        |
| SDA (1)  | PE_5        |
| SCK (2)  | PE_6        |
| MOSI (0) | PE_7        |
| SCL (1)  | PE_8        |
| SDA (1)  | PE_9        |
| SCK (2)  | PE_10       |
| MOSI (0) | PE_11       |
| SCL (1)  | PE_12       |
| SDA (1)  | PE_13       |
| SCK (2)  | PE_14       |
| MOSI (0) | PE_15       |
| SCL (1)  | PE_16       |
| SDA (1)  | PE_17       |
| SCK (2)  | PE_18       |
| MOSI (0) | PE_19       |
| SCL (1)  | PE_20       |
| SDA (1)  | PE_21       |
| SCK (2)  | PE_22       |
| MOSI (0) | n           |

User pins present only!  
Quadrature Encoder - TTL level  
only on TM4C123GH6PM

**LaunchPad with LM4F120H5QR**  
**LaunchPad with TM4C123GH6PM**  
Revision 1

**Hardware Pin number**  
Other pin number

| Pin       | Pin number | Other pin number |
|-----------|------------|------------------|
| J1        | J1         | J1               |
| J2        | J2         | J2               |
| RESET     | RESET      | RESET            |
| GREEN LED | PF_3       | PF_3             |
| RED LED   | PF_2       | PF_2             |
| GROUND    | GROUND     | GROUND           |
| SCL (0)   | SCL (0)    | SCL (0)          |
| SDA (0)   | SDA (0)    | SDA (0)          |
| Index (1) | Index (1)  | Index (1)        |
| TX (1)    | TX (1)     | TX (1)           |
| TX (0)    | TX (0)     | TX (0)           |
| TX (1)    | TX (1)     | TX (1)           |
| TX (0)    | TX (0)     | TX (0)           |
| RESET     | RESET      | RESET            |
| SDA (2)   | SDA (2)    | SDA (2)          |
| MOSI (2)  | MOSI (2)   | MOSI (2)         |
| SCL (3)   | SCL (3)    | SCL (3)          |
| C8 (0)    | C8 (0)     | C8 (0)           |
| C9 (0)    | C9 (0)     | C9 (0)           |
| C10 (0)   | C10 (0)    | C10 (0)          |
| C11 (0)   | C11 (0)    | C11 (0)          |
| C12 (0)   | C12 (0)    | C12 (0)          |
| C13 (0)   | C13 (0)    | C13 (0)          |
| C14 (0)   | C14 (0)    | C14 (0)          |
| C15 (0)   | C15 (0)    | C15 (0)          |
| C16 (0)   | C16 (0)    | C16 (0)          |
| C17 (0)   | C17 (0)    | C17 (0)          |
| C18 (0)   | C18 (0)    | C18 (0)          |
| C19 (0)   | C19 (0)    | C19 (0)          |
| C20 (0)   | C20 (0)    | C20 (0)          |
| C21 (0)   | C21 (0)    | C21 (0)          |
| C22 (0)   | C22 (0)    | C22 (0)          |
| C23 (0)   | C23 (0)    | C23 (0)          |
| C24 (0)   | C24 (0)    | C24 (0)          |
| GROUND    | GROUND     | GROUND           |
| PA_0      | PA_0       | PA_0             |
| PA_1      | PA_1       | PA_1             |
| PA_2      | PA_2       | PA_2             |
| PA_3      | PA_3       | PA_3             |
| PA_4      | PA_4       | PA_4             |
| PA_5      | PA_5       | PA_5             |
| PA_6      | PA_6       | PA_6             |
| PA_7      | PA_7       | PA_7             |
| PA_8      | PA_8       | PA_8             |
| PA_9      | PA_9       | PA_9             |
| PA_10     | PA_10      | PA_10            |
| PA_11     | PA_11      | PA_11            |
| PA_12     | PA_12      | PA_12            |
| PA_13     | PA_13      | PA_13            |
| PA_14     | PA_14      | PA_14            |
| PA_15     | PA_15      | PA_15            |
| PA_16     | PA_16      | PA_16            |
| PA_17     | PA_17      | PA_17            |
| PA_18     | PA_18      | PA_18            |
| PA_19     | PA_19      | PA_19            |
| PA_20     | PA_20      | PA_20            |
| PA_21     | PA_21      | PA_21            |
| PA_22     | PA_22      | PA_22            |
| PA_23     | PA_23      | PA_23            |
| PA_24     | PA_24      | PA_24            |
| PA_25     | PA_25      | PA_25            |
| PA_26     | PA_26      | PA_26            |
| PA_27     | PA_27      | PA_27            |
| PA_28     | PA_28      | PA_28            |
| PA_29     | PA_29      | PA_29            |
| PA_30     | PA_30      | PA_30            |
| PA_31     | PA_31      | PA_31            |
| PA_32     | PA_32      | PA_32            |
| PA_33     | PA_33      | PA_33            |
| PA_34     | PA_34      | PA_34            |
| PA_35     | PA_35      | PA_35            |
| PA_36     | PA_36      | PA_36            |
| PA_37     | PA_37      | PA_37            |
| PA_38     | PA_38      | PA_38            |
| PA_39     | PA_39      | PA_39            |
| PA_40     | PA_40      | PA_40            |
| PA_41     | PA_41      | PA_41            |
| PA_42     | PA_42      | PA_42            |
| PA_43     | PA_43      | PA_43            |
| PA_44     | PA_44      | PA_44            |
| PA_45     | PA_45      | PA_45            |
| PA_46     | PA_46      | PA_46            |
| PA_47     | PA_47      | PA_47            |
| PA_48     | PA_48      | PA_48            |
| PA_49     | PA_49      | PA_49            |
| PA_50     | PA_50      | PA_50            |
| PA_51     | PA_51      | PA_51            |
| PA_52     | PA_52      | PA_52            |
| PA_53     | PA_53      | PA_53            |
| PA_54     | PA_54      | PA_54            |
| PA_55     | PA_55      | PA_55            |
| PA_56     | PA_56      | PA_56            |
| PA_57     | PA_57      | PA_57            |
| PA_58     | PA_58      | PA_58            |
| PA_59     | PA_59      | PA_59            |
| PA_60     | PA_60      | PA_60            |
| PA_61     | PA_61      | PA_61            |
| PA_62     | PA_62      | PA_62            |
| PA_63     | PA_63      | PA_63            |
| PA_64     | PA_64      | PA_64            |
| PA_65     | PA_65      | PA_65            |
| PA_66     | PA_66      | PA_66            |
| PA_67     | PA_67      | PA_67            |
| PA_68     | PA_68      | PA_68            |
| PA_69     | PA_69      | PA_69            |
| PA_70     | PA_70      | PA_70            |
| PA_71     | PA_71      | PA_71            |
| PA_72     | PA_72      | PA_72            |
| PA_73     | PA_73      | PA_73            |
| PA_74     | PA_74      | PA_74            |
| PA_75     | PA_75      | PA_75            |
| PA_76     | PA_76      | PA_76            |
| PA_77     | PA_77      | PA_77            |
| PA_78     | PA_78      | PA_78            |
| PA_79     | PA_79      | PA_79            |
| PA_80     | PA_80      | PA_80            |
| PA_81     | PA_81      | PA_81            |
| PA_82     | PA_82      | PA_82            |
| PA_83     | PA_83      | PA_83            |
| PA_84     | PA_84      | PA_84            |
| PA_85     | PA_85      | PA_85            |
| PA_86     | PA_86      | PA_86            |
| PA_87     | PA_87      | PA_87            |
| PA_88     | PA_88      | PA_88            |
| PA_89     | PA_89      | PA_89            |
| PA_90     | PA_90      | PA_90            |
| PA_91     | PA_91      | PA_91            |
| PA_92     | PA_92      | PA_92            |
| PA_93     | PA_93      | PA_93            |
| PA_94     | PA_94      | PA_94            |
| PA_95     | PA_95      | PA_95            |
| PA_96     | PA_96      | PA_96            |
| PA_97     | PA_97      | PA_97            |
| PA_98     | PA_98      | PA_98            |
| PA_99     | PA_99      | PA_99            |
| PA_100    | PA_100     | PA_100           |
| PA_101    | PA_101     | PA_101           |
| PA_102    | PA_102     | PA_102           |
| PA_103    | PA_103     | PA_103           |
| PA_104    | PA_104     | PA_104           |
| PA_105    | PA_105     | PA_105           |
| PA_106    | PA_106     | PA_106           |
| PA_107    | PA_107     | PA_107           |
| PA_108    | PA_108     | PA_108           |
| PA_109    | PA_109     | PA_109           |
| PA_110    | PA_110     | PA_110           |
| PA_111    | PA_111     | PA_111           |
| PA_112    | PA_112     | PA_112           |
| PA_113    | PA_113     | PA_113           |
| PA_114    | PA_114     | PA_114           |
| PA_115    | PA_115     | PA_115           |
| PA_116    | PA_116     | PA_116           |
| PA_117    | PA_117     | PA_117           |
| PA_118    | PA_118     | PA_118           |
| PA_119    | PA_119     | PA_119           |
| PA_120    | PA_120     | PA_120           |
| PA_121    | PA_121     | PA_121           |
| PA_122    | PA_122     | PA_122           |
| PA_123    | PA_123     | PA_123           |
| PA_124    | PA_124     | PA_124           |
| PA_125    | PA_125     | PA_125           |
| PA_126    | PA_126     | PA_126           |
| PA_127    | PA_127     | PA_127           |
| PA_128    | PA_128     | PA_128           |
| PA_129    | PA_129     | PA_129           |
| PA_130    | PA_130     | PA_130           |
| PA_131    | PA_131     | PA_131           |
| PA_132    | PA_132     | PA_132           |
| PA_133    | PA_133     | PA_133           |
| PA_134    | PA_134     | PA_134           |
| PA_135    | PA_135     | PA_135           |
| PA_136    | PA_136     | PA_136           |
| PA_137    | PA_137     | PA_137           |
| PA_138    | PA_138     | PA_138           |
| PA_139    | PA_139     | PA_139           |
| PA_140    | PA_140     | PA_140           |
| PA_141    | PA_141     | PA_141           |
| PA_142    | PA_142     | PA_142           |
| PA_143    | PA_143     | PA_143           |
| PA_144    | PA_144     | PA_144           |
| PA_145    | PA_145     | PA_145           |
| PA_146    | PA_146     | PA_146           |
| PA_147    | PA_147     | PA_147           |
| PA_148    | PA_148     | PA_148           |
| PA_149    | PA_149     | PA_149           |
| PA_150    | PA_150     | PA_150           |
| PA_151    | PA_151     | PA_151           |
| PA_152    | PA_152     | PA_152           |
| PA_153    | PA_153     | PA_153           |
| PA_154    | PA_154     | PA_154           |
| PA_155    | PA_155     | PA_155           |
| PA_156    | PA_156     | PA_156           |
| PA_157    | PA_157     | PA_157           |
| PA_158    | PA_158     | PA_158           |
| PA_159    | PA_159     | PA_159           |
| PA_160    | PA_160     | PA_160           |
| PA_161    | PA_161     | PA_161           |
| PA_162    | PA_162     | PA_162           |
| PA_163    | PA_163     | PA_163           |
| PA_164    | PA_164     | PA_164           |
| PA_165    | PA_165     | PA_165           |
| PA_166    | PA_166     | PA_166           |
| PA_167    | PA_167     | PA_167           |
| PA_168    | PA_168     | PA_168           |
| PA_169    | PA_169     | PA_169           |
| PA_170    | PA_170     | PA_170           |
| PA_171    | PA_171     | PA_171           |
| PA_172    | PA_172     | PA_172           |
| PA_173    | PA_173     | PA_173           |
| PA_174    | PA_174     | PA_174           |
| PA_175    | PA_175     | PA_175           |
| PA_176    | PA_176     | PA_176           |
| PA_177    | PA_177     | PA_177           |
| PA_178    | PA_178     | PA_178           |
| PA_179    | PA_179     | PA_179           |
| PA_180    | PA_180     | PA_180           |
| PA_181    | PA_181     | PA_181           |
| PA_182    | PA_182     | PA_182           |
| PA_183    | PA_183     | PA_183           |
| PA_184    | PA_184     | PA_184           |
| PA_185    | PA_185     | PA_185           |
| PA_186    | PA_186     | PA_186           |
| PA_187    | PA_187     | PA_187           |
| PA_188    | PA_188     | PA_188           |
| PA_189    | PA_189     | PA_189           |
| PA_190    | PA_190     | PA_190           |
| PA_191    | PA_191     | PA_191           |
| PA_192    | PA_192     | PA_192           |
| PA_193    | PA_193     | PA_193           |
| PA_194    | PA_194     | PA_194           |
| PA_195    | PA_195     | PA_195           |
| PA_196    | PA_196     | PA_196           |
| PA_197    | PA_197     | PA_197           |
| PA_198    | PA_198     | PA_198           |
| PA_199    | PA_199     | PA_199           |
| PA_200    | PA_200     | PA_200           |
| PA_201    | PA_201     | PA_201           |
| PA_202    | PA_202     | PA_202           |
| PA_203    | PA_203     | PA_203           |
| PA_204    | PA_204     | PA_204           |
| PA_205    | PA_205     | PA_205           |
| PA_206    | PA_206     | PA_206           |
| PA_207    | PA_207     | PA_207           |
| PA_208    | PA_208     | PA_208           |
| PA_209    | PA_209     | PA_209           |
| PA_210    | PA_210     | PA_210           |
| PA_211    | PA_211     | PA_211           |
| PA_212    | PA_212     | PA_212           |
| PA_213    | PA_213     | PA_213           |
| PA_214    | PA_214     | PA_214           |
| PA_215    | PA_215     | PA_215           |
| PA_216    | PA_216     | PA_216           |
| PA_217    | PA_217     | PA_217           |
| PA_218    | PA_218     | PA_218           |
| PA_219    | PA_219     | PA_219           |
| PA_220    | PA_220     | PA_220           |
| PA_221    | PA_221     | PA_221           |
| PA_222    | PA_222     | PA_222           |
| PA_223    | PA_223     | PA_223           |
| PA_224    | PA_224     | PA_224           |
| PA_225    | PA_225     | PA_225           |
| PA_226    | PA_226     | PA_226           |
| PA_227    | PA_227     | PA_227           |
| PA_228    | PA_228     | PA_228           |
| PA_229    | PA_229     | PA_229           |
| PA_230    | PA_230     | PA_230           |
| PA_231    | PA_231     | PA_231           |
| PA_232    | PA_232     | PA_232           |
| PA_233    | PA_233     | PA_233           |
| PA_234    | PA_234     | PA_234           |
| PA_235    | PA_235     | PA_235           |
| PA_236    | PA_236     | PA_236           |
| PA_237    | PA_237     | PA_237           |
| PA_238    | PA_238     | PA_238           |
| PA_239    | PA_239     | PA_239           |
| PA_240    | PA_240     | PA_240           |
| PA_241    | PA_241     | PA_241           |
| PA_242    | PA_242     | PA_242           |
| PA_243    | PA_243     | PA_243           |
| PA_244    | PA_244     | PA_244           |
| PA_245    | PA_245     | PA_245           |
| PA_246    | PA_246     | PA_246           |
| PA_247    | PA_247     | PA_247           |
| PA_248    | PA_248     | PA_248           |
| PA_249    | PA_249     | PA_249           |
| PA_250    | PA_250     | PA_250           |
| PA_251    | PA_251     | PA_251           |
| PA_252    | PA_252     | PA_252           |
| PA_253    | PA_253     | PA_253           |
| PA_254    | PA_254     | PA_254           |
| PA_255    | PA_255     | PA_255           |
| PA_256    | PA_256     | PA_256           |
| PA_257    | PA_257     | PA_257           |
| PA_258    | PA_258     | PA_258           |
| PA_259    | PA_259     | PA_259           |
| PA_260    | PA_260     | PA_260           |
| PA_261    | PA_261     | PA_261           |
| PA_262    | PA_262     | PA_262           |
| PA_263    | PA_263     | PA_263           |
| PA_264    | PA_264     | PA_264           |
| PA_265    | PA_265     | PA_265           |
| PA_266    | PA_266     | PA_266           |
| PA_267    | PA_267     | PA_267           |
| PA_268    | PA_268     | PA_268           |
| PA_269    | PA_269     | PA_269           |
| PA_270    | PA_270     | PA_270           |
| PA_271    | PA_271     | PA_271           |
| PA_272    | PA_272     | PA_272           |
| PA_273    | PA_273     | PA_273           |
| PA_274    | PA_274     | PA_274           |
| PA_275    | PA_275     | PA_275           |
| PA_276    | PA_276     | PA_276           |
| PA_277    | PA_277     | PA_277           |
| PA_278    | PA_278     | PA_278           |
| PA_279    | PA_279     | PA_279           |
| PA_280    | PA_280     | PA_280           |
| PA_281    | PA_281     | PA_281           |
| PA_282    | PA_282     | PA_282           |
| PA_283    | PA_283     | PA_283           |
| PA_284    | PA_284     | PA_284           |
| PA_285    | PA_285     | PA_285           |
| PA_286    | PA_286     | PA_286           |
| PA_287    | PA_287     | PA_287           |
| PA_288    | PA_288     | PA_288           |
| PA_289    | PA_289     | PA_289           |
| PA_290    | PA_290     | PA_290           |
| PA_291    | PA_291     | PA_291           |
| PA_292    | PA_292     | PA_292           |
| PA_293    | PA_293     | PA_293           |
| PA_294    | PA_294     | PA_294           |
| PA_295    | PA_295     | PA_295           |
| PA_296    | PA_296     | PA_296           |
| PA_297    | PA_297     | PA_297           |
| PA_298    | PA_298     | PA_298           |
| PA_299    | PA_299     | PA_299           |
| PA_300    | PA_300     | PA_300           |
| PA_301    | PA         |                  |

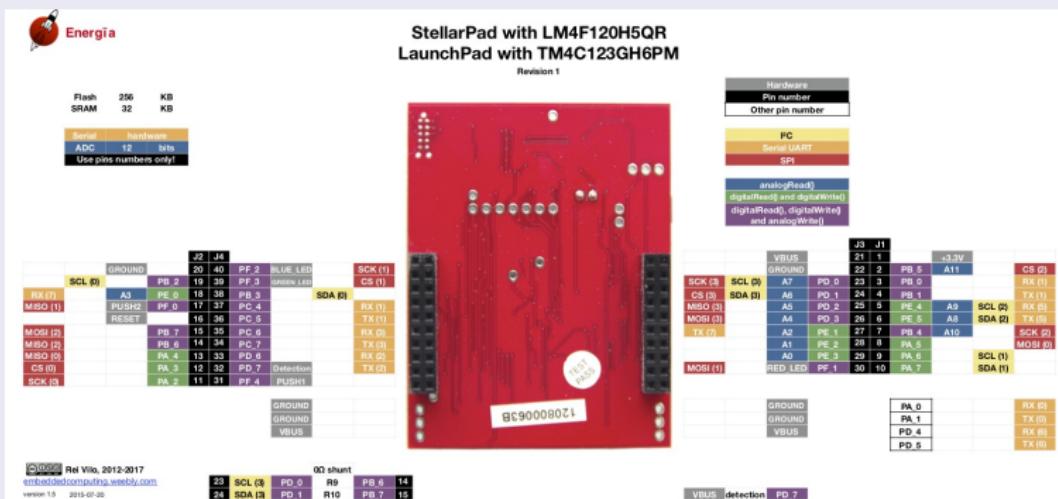


# UART Communication Port On TM4C

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## BOTTOM VIEW





# UART Enable Steps

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

- ① Enable the UART module using the RCGCUART register  
 $SYSCTL \rightarrow RCGCUART = (1 << 0);$
- ② Enable the clock to the appropriate GPIO module A  
 $SYSCTL \rightarrow RCGCGPIO = (1 << 0);$
- ③ Set the GPIO AFSEL bits for the appropriate pins  
 $GPIOA \rightarrow AFSEL = (1 << 1)|(1 << 0);$
- ④ Configure the PMCn fields in the GPIOPCTL register to assign the UART signals to the appropriate pins  
 $GPIOA \rightarrow PCTL = (1 << 0)|(1 << 4);$





# UART Enable Steps

- ① Enable the Digital function on Port A  
 $GPIOA \rightarrow DEN = (1 << 0) | (1 << 1);$
- ② Find the Baud-Rate Divisor  
 $BRD = 16,000,000 / (16 * 9600) = 104.16666$
- ③ Configure the UART to Operate at 9600 baud rate  
 $UARTFBRD = \text{integer}(0.166667 * 64 + 0.5) = 11$
- ④ Disable the UART by clearing the UARTEN bit in the  
UARTCTL register  $UART0 \rightarrow CTL\& = \neg(1 << 0);$
- ⑤ Write the integer portion of the BRD  
 $UART0 \rightarrow IBRD = 104;$
- ⑥ Write the fractional portion of the BRD  
 $UART0 \rightarrow FBRD = 11;$



# UART Enable Steps

- ① Write the desired serial parameters to the **UARTLCRH** register  $UART0 \rightarrow LCRH = (0x3 << 5)|(1 << 4);$
- ② Configure the UART clock source by writing to the **UARTCC** register  
 $UART0 \rightarrow CC = 0x0;$
- ③ Enable the UART by setting the **UARTEN** bit in the **UARTCTL** register  
 $UART0 \rightarrow CTL = (1 << 0)|(1 << 8)|(1 << 9);$



# UART Communication To Light The LED

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

- ① Configure LED pins on Port F

*SYSCTL -> RCGCGPIO = (1 << 5);  
GPIOF -> DIR = (1 << 1)|(1 << 2)|(1 << 3);  
GPIOF -> DEN = (1 << 1)|(1 << 2)|(1 << 3);  
GPIOF -> DATA& = ((1 << 1)|(1 << 2)|(1 << 3));*

- ② Write a function to print string on com port
- ③ Write read char function to read the user send char on serial
- ④ Write the Switch and case statement to light the appropriate led
- ⑤ We need UART0\_FR and UART0\_DATA register.



# Synchronous Serial Interface

## SSI

- ① The Serial Peripheral Interface (SPI) is a synchronous serial communication interface
- ② The interface was developed by Motorola in the mid 1980s
- ③ SPI devices communicate in full duplex mode using a master-slave architecture with a single master.
- ④ Multiple slave devices are supported through selection with individual slave select (SS) lines.

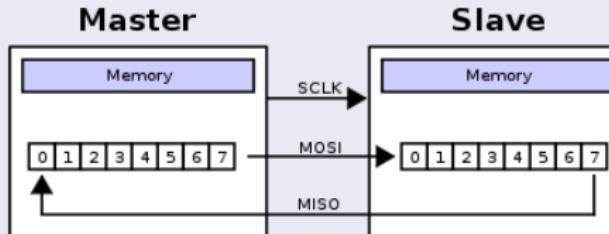




# Synchronous Serial Interface

## Data Transmission

- ① The Bus master configures the clock using a frequency supported by the slave device
- ② The master then selects the slave device with logic 0 on select line
- ③ Transmissions normally involve two shift registers of some given word size (8 bit)
- ④ These shift register in master and slave devices are virtual connected in ring topology





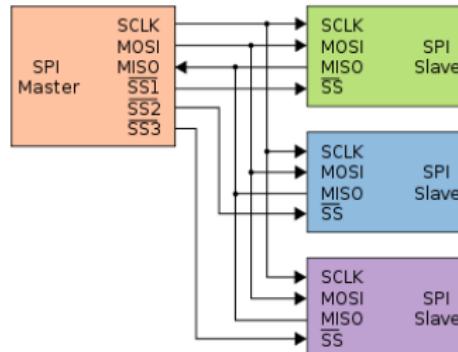
# Synchronous Serial Interface

TM4C123GH6P

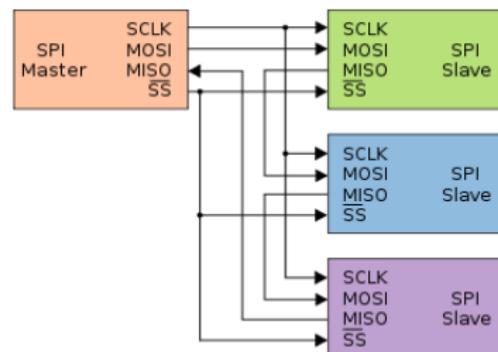
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Connecting with Multiple SPI device



Typical SPI bus: master and three independent slaves



Daisy-chained SPI bus: master and cooperative slaves



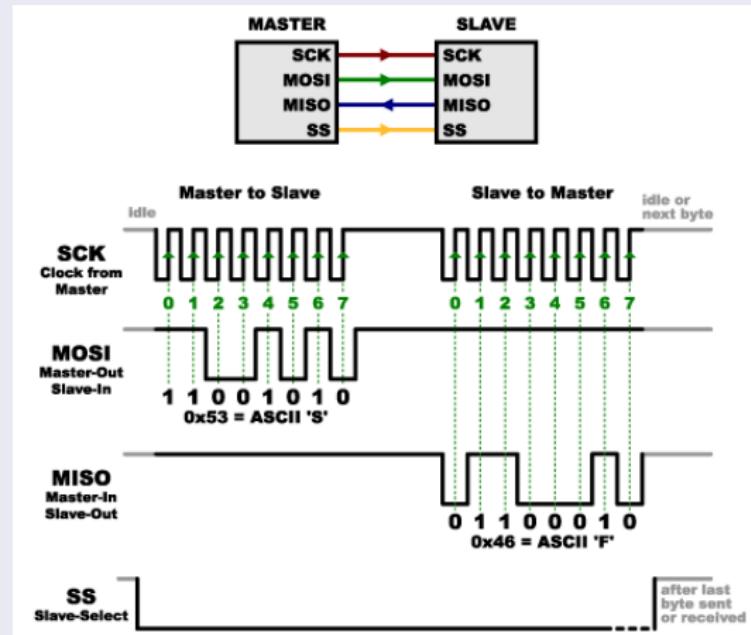
# Synchronous Serial Interface

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Operation

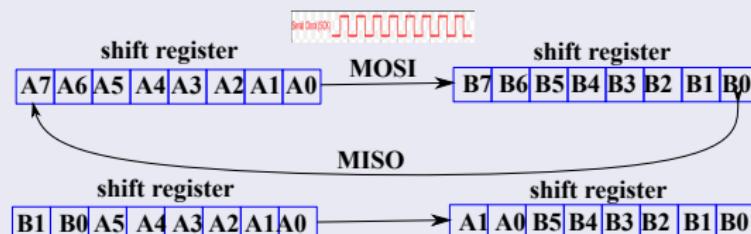




# Synchronous Serial Interface

## Configuration

- SPI communication uses simple shift register

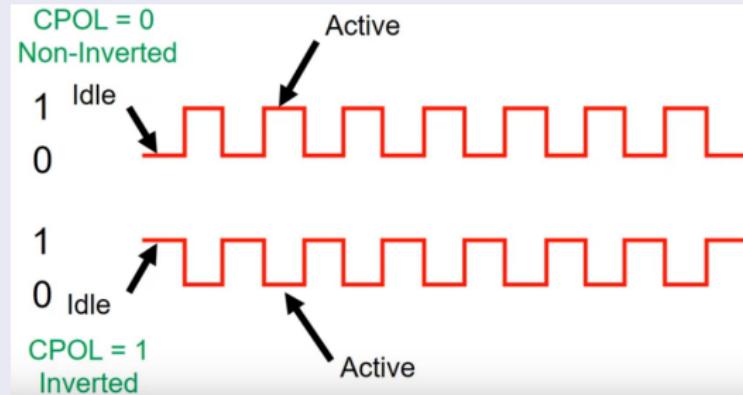




# Synchronous Serial Interface

## Configuration

- SPI Mode Configuration required to set two register
  - Clock Polarity (CPOL)
  - Clock Phase(CPHASE)
    - When the data has to be toggled
    - When the data has to be sampled





# Synchronous Serial Interface

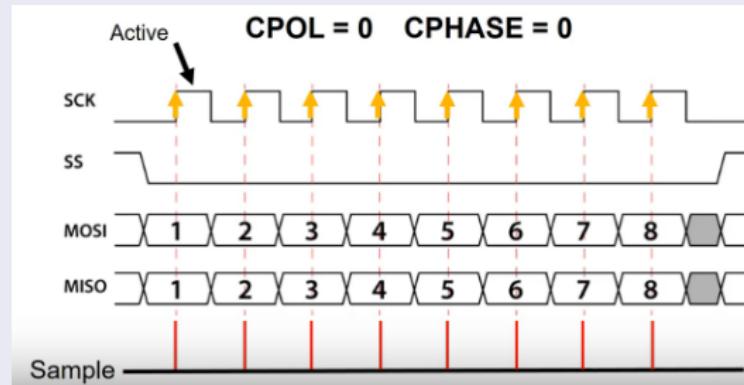
TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Configuration

- **CPHASE=0** Data will be sampled at the leading edge of the clock
- **CPHASE=1** Data will be sampled at the trailing edge of the clock





# Synchronous Serial Interface

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Modes

| Mode   | CPOL | CPHASE | Comments   |
|--------|------|--------|--|
| Mode 0 | 0    | 0      | Active state of clock is 1<br>Data sampling at leading edge  |
| Mode 1 | 0    | 1      | Active state of clock is 1<br>Data sampling at trailing edge |
| Mode 2 | 1    | 0      | Active state of clock is 0<br>Data sampling at leading edge  |
| Mode 3 | 1    | 1      | Active state of clock is 0<br>Data sampling at trailing edge |



# I2C Communication Port

## What is I2C

- Inter-integrated Circuit
- Bidirectional Data Transfer
- Half Duplex (have only one data-line)
- Synchronous bus so data is clocked with clock signal
- Clock is controlled when data line is changed

## Speed of I2C

- Low (under 100 kbps)
- Fast (400 Kbps)
- High speed (3.4 mbps) (I2C V2.0)
- 2 Wire communication:
- SDA and SCL



# I2C Communication Port

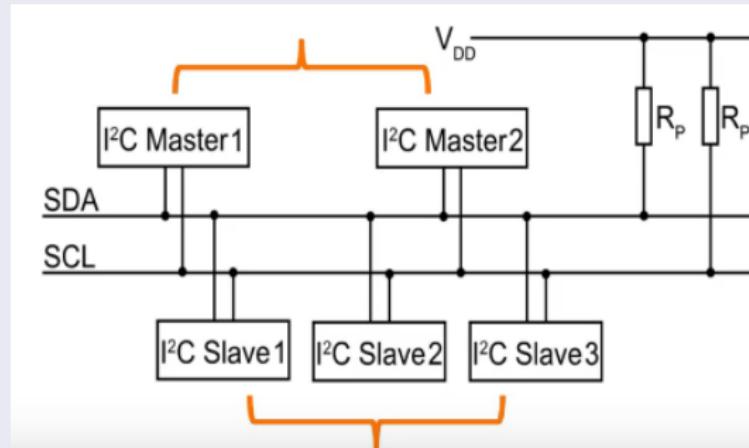
TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## What is I2C



## Basic protocol is master slave protocol





# I2C Communication Port

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Packet Format



## Basic protocol is master slave protocol

- Master controls the clock
- Slave device may hold the clock low to prevent data transfer
- No data transfer is present when clock is low
- It is a kind of wired and connection
- Need to put pullup resistor
- Default state is high when no device is pulling it low



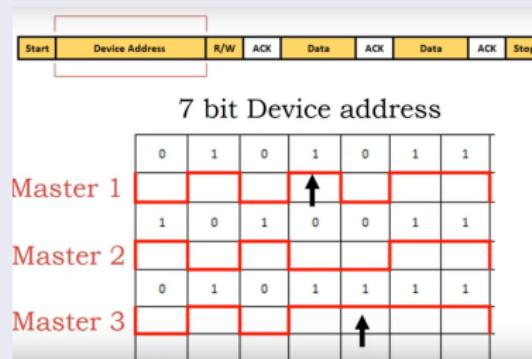
# I2C Concepts

## Arbitration

- Process to get the bus access
- Related to SDA line.

## Clock Stretching

- Process to slow down the communication
- Related to SCL line.





# I2C Driver Development

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Case 1 "Master Want to Write 1 Byte"





# Control Area Networks

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## CAN

- Controller Area Network (CAN bus)
- Developed by Robert Bosch in 1986
- No Host required
- CAN bus is a broadcast type of bus
- Message based protocol
- Serial half-duplex Asynchronous Communication
- Differential Two wired communication.
- Designed originally for automobiles, but also used in many other contexts.



# Use In Automobile

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## CAN Protocol

Engine  
Control Unit

Transmission  
Control

Airbag

Ignition

Dashboard

ABS

Power  
Window

Mirror  
Adjustment

Doors



# Application of CAN Protocol

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## CAN Protocol

- Automotive(passenger vehicles, trucks, buses)
- Electronic equipment for aviation and navigation
- Industrial automation and mechanical control
- Elevators & escalators
- Building automation
- Medical instruments and equipment
- Marine, Military, Industrial, Medical

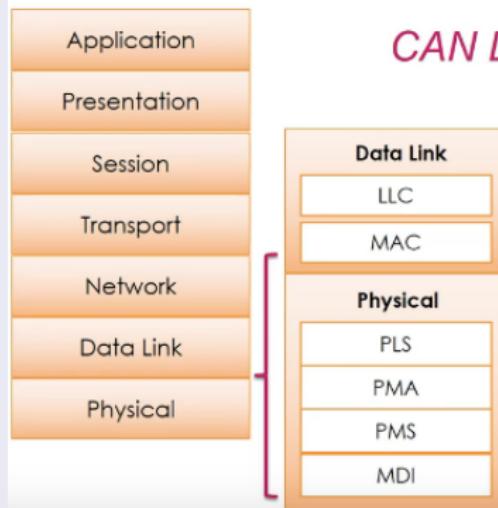


# CAN Architecture

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## CAN



### *CAN Layered Architecture*

#### MAC : Medium Access Control

- Medium Access
- Encapsulation/Decapsulation
- Error Detection
- Signalling

#### LLC : Logical Link Control

- Frame Acceptance Filtering
- Overload Notification
- Recovery Management

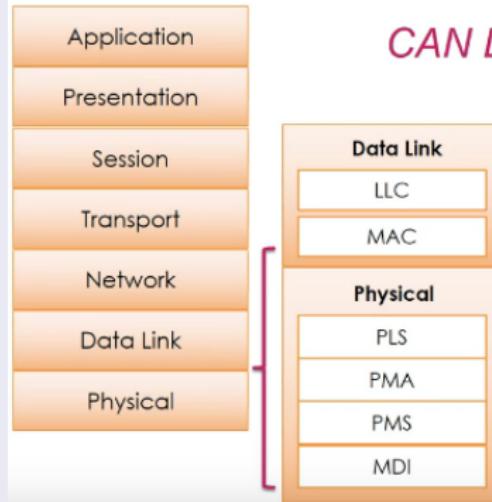


# CAN Architecture

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## CAN



### *CAN Layered Architecture*

LLC : Logical Link Control

MAC : Medium Access Control

PLS : Physical Layer Signalling

PMA : Physical Medium Attachment

PMS : Physical Medium Specification

MDI : Medium Dependent Interface



# CAN Architecture

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## CAN

- Datalink Layer (CAN Protocol)
- Physical Layer (CAN Physical Layer)
- CAN Datalink Layer has standard (ISO 11898-1)
- CAN Physical Layer has tree standard
  - ISO 11898-2 (signalling) (datarate-1 Mbps)
  - ISO 11898-3 (medium) (datarate-125kb)
  - CIA (CAN in Automation) DS-102 (Types of connectors)
- CAN controller and CAN tranciver are implemented in software with the help of operating system and network functions
- CAN latest version is CAN 2.0
  - CAN 2.0 A (11 bit identifier)
  - CAN 2.0 B (29 bit identifier)



# CAN Prioritization of Message

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

- CAN is not an address/node oriented protocol
- CAN is message based protocol
- High priority message will be transmitted first
- Priority will be decided based on identifier
- Lower the value of identifier, priority will be higher and vice versa
- An example
  - ECU1: Have a message with identifier 0x1A1
  - ECU2: Have a message with identifier 0x100
  - ECU3: Have a message with identifier 0x21A
  - Binary representation of 1A1: **001 1010 0001**
  - Binary representation of 1A1: **001 0000 0000**
  - Binary representation of 1A1: **010 0001 1010**
- As we know the lowest value here is 0x100 (duw to wired AND logic), so this is high priority message(ECU2)



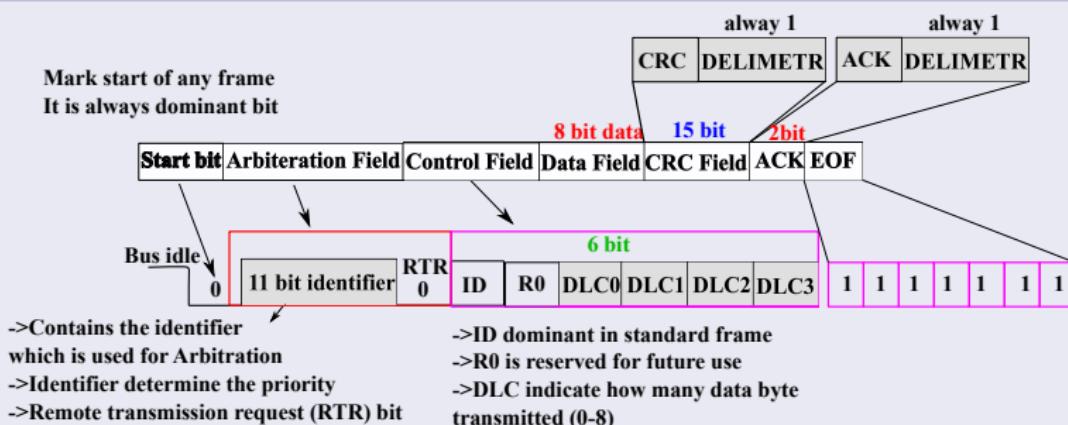
# CAN Frame Format

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## CAN





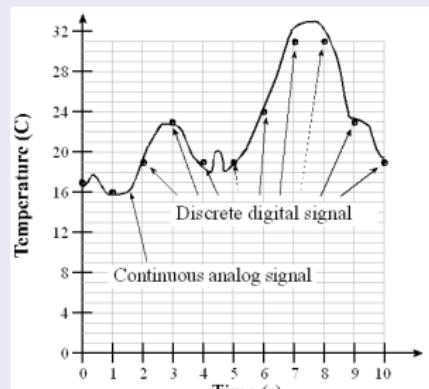
# Digital to Analog Conversion

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

- Analog world to Digitalization
  - Amplitude Quantization (Levels the amplitude)
  - Time Quantization (Sampling rate)
- The Nyquist Theorem states that if the signal is sampled with a frequency of  $f_s$ , then to correctly digitalizes the signal we need to sample the signal twice of sampling frequency



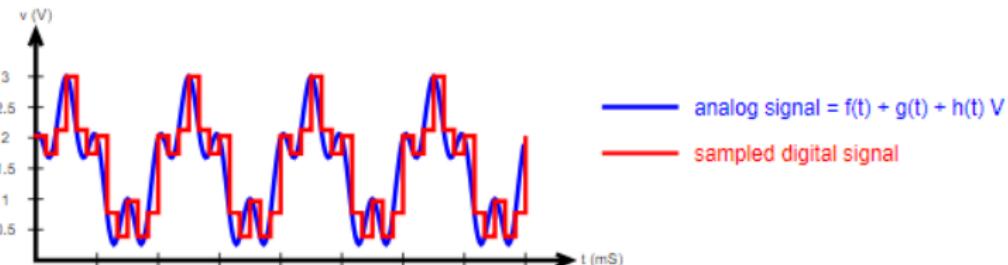
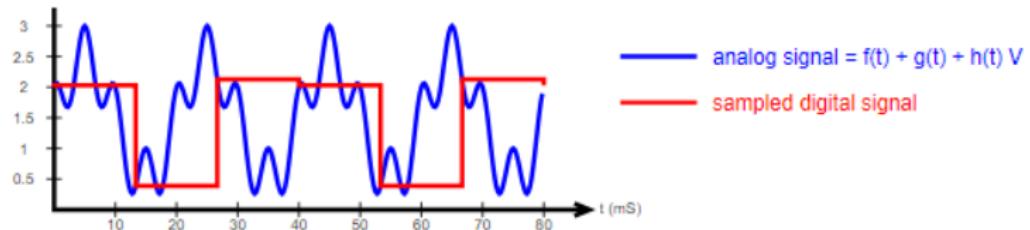
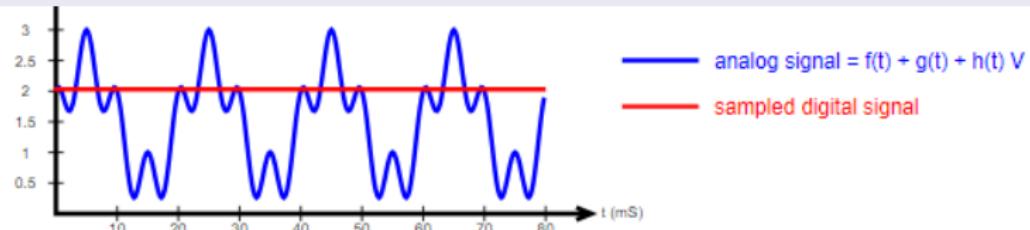


# Nyquist Theorem

TM4C123GH6P  
Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

## Sampling frequency





# Digital to Analog Converter

TM4C123GH6P

Micro-  
controllers  
Programming  
Concepts

Dr. Munesh  
Singh

- The DAC precision is the number of distinguishable DAC outputs
- Lets say 4 bit digital to analog converter we have to design
- Maximum voltage range is from 0 to 3.3 v
- Number of levels  $2^4$  is 16 bits

