TM4C123GH6PM Micro-controllers Programming Concepts

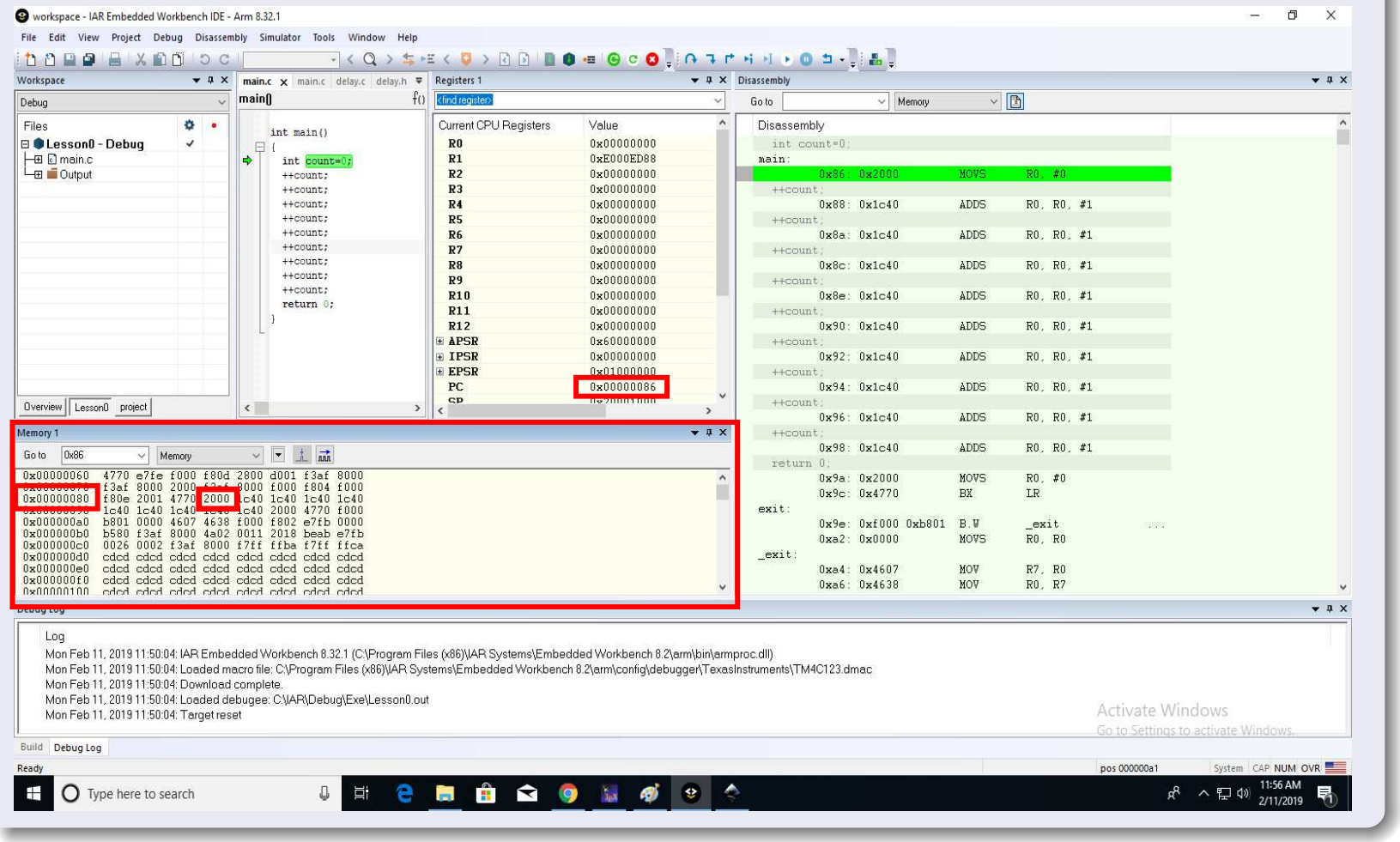
Dr. Munesh Singh

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

March 12, 2019

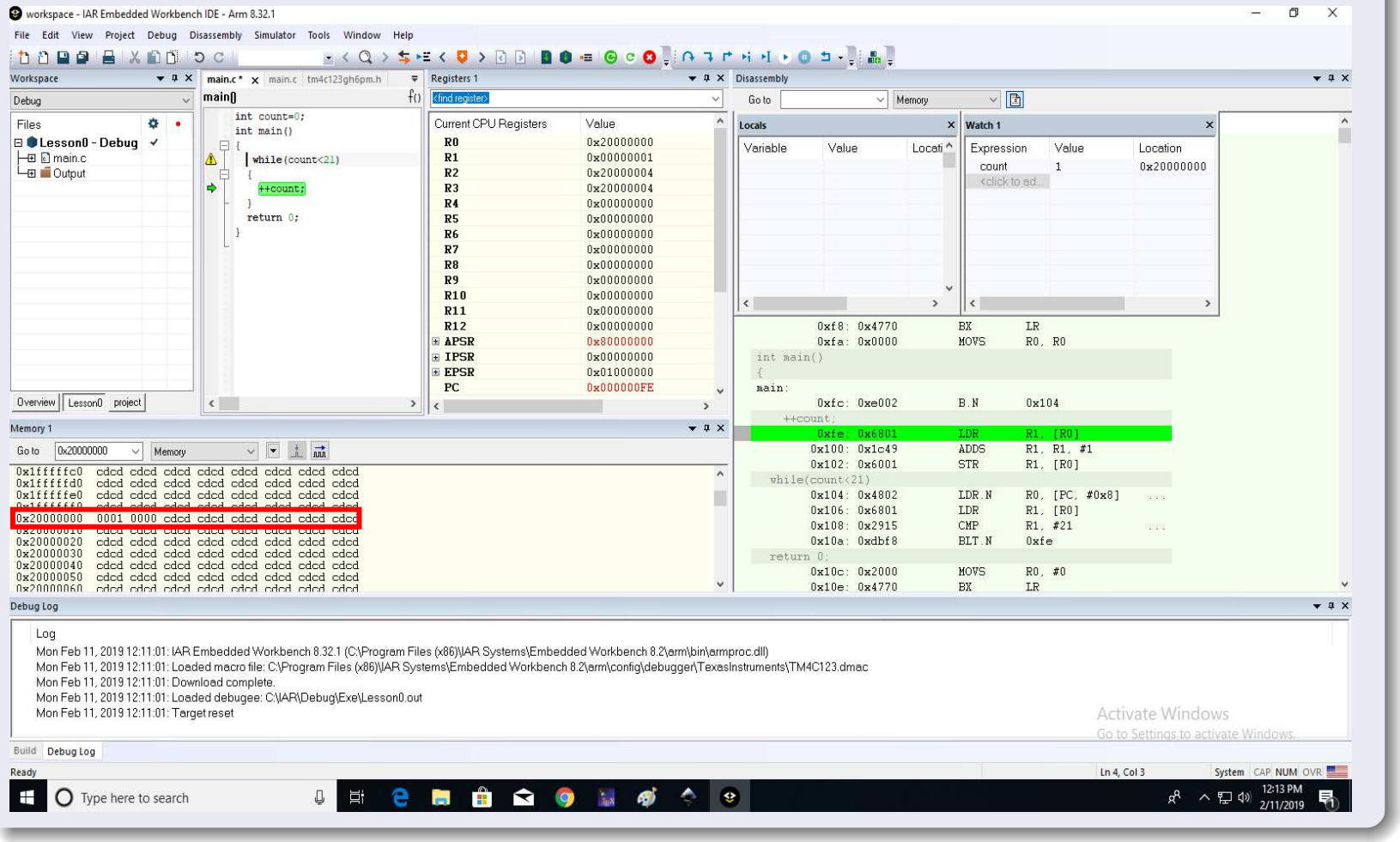
IAR Workbench Toolset

ARM Microcontroller

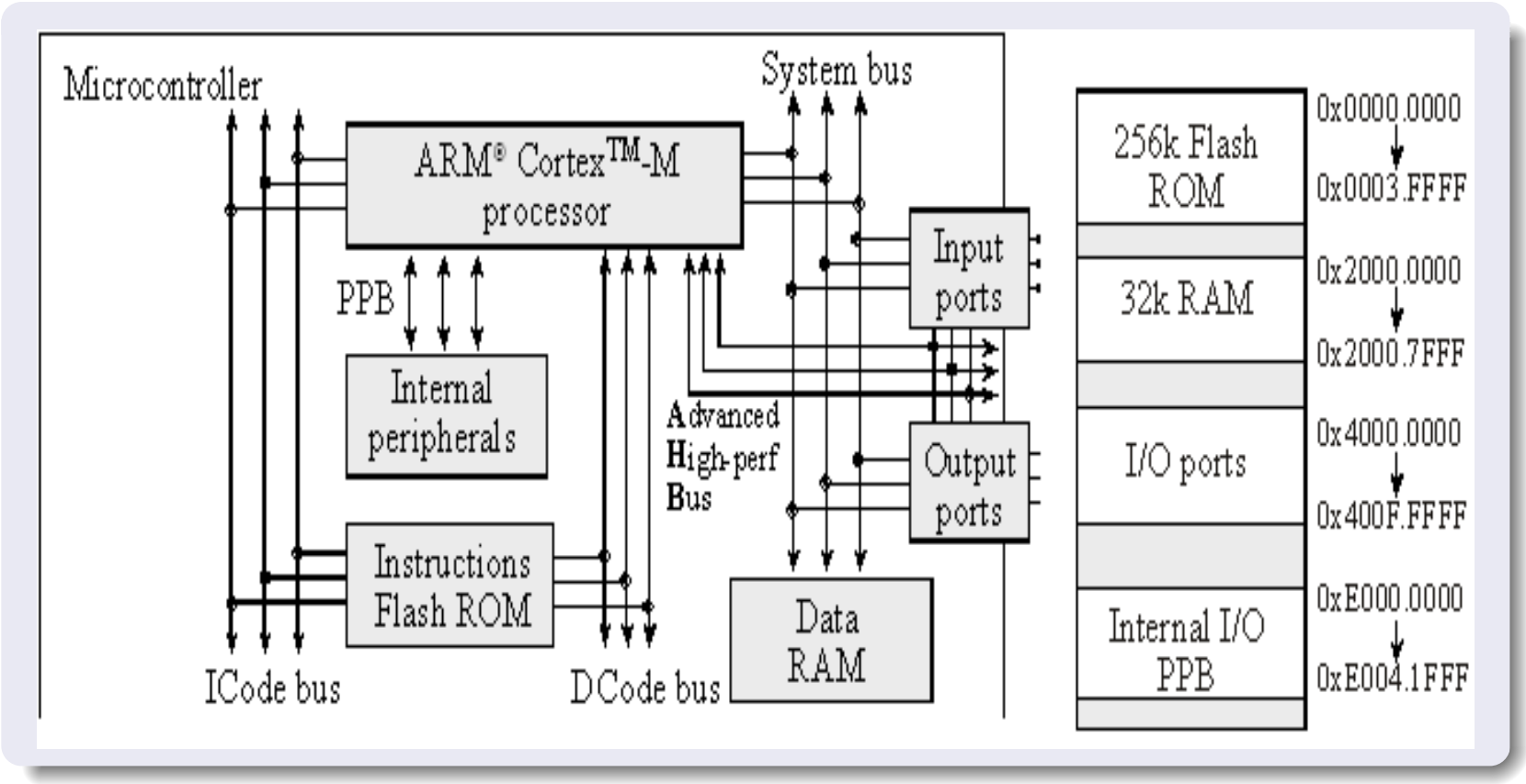


IAR Workbench Toolset

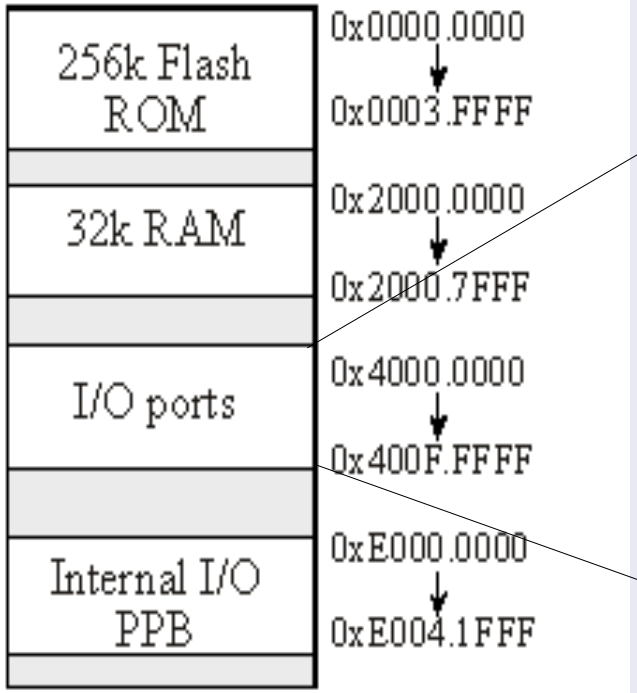
Local versus Global Variable



Memory Map Buses

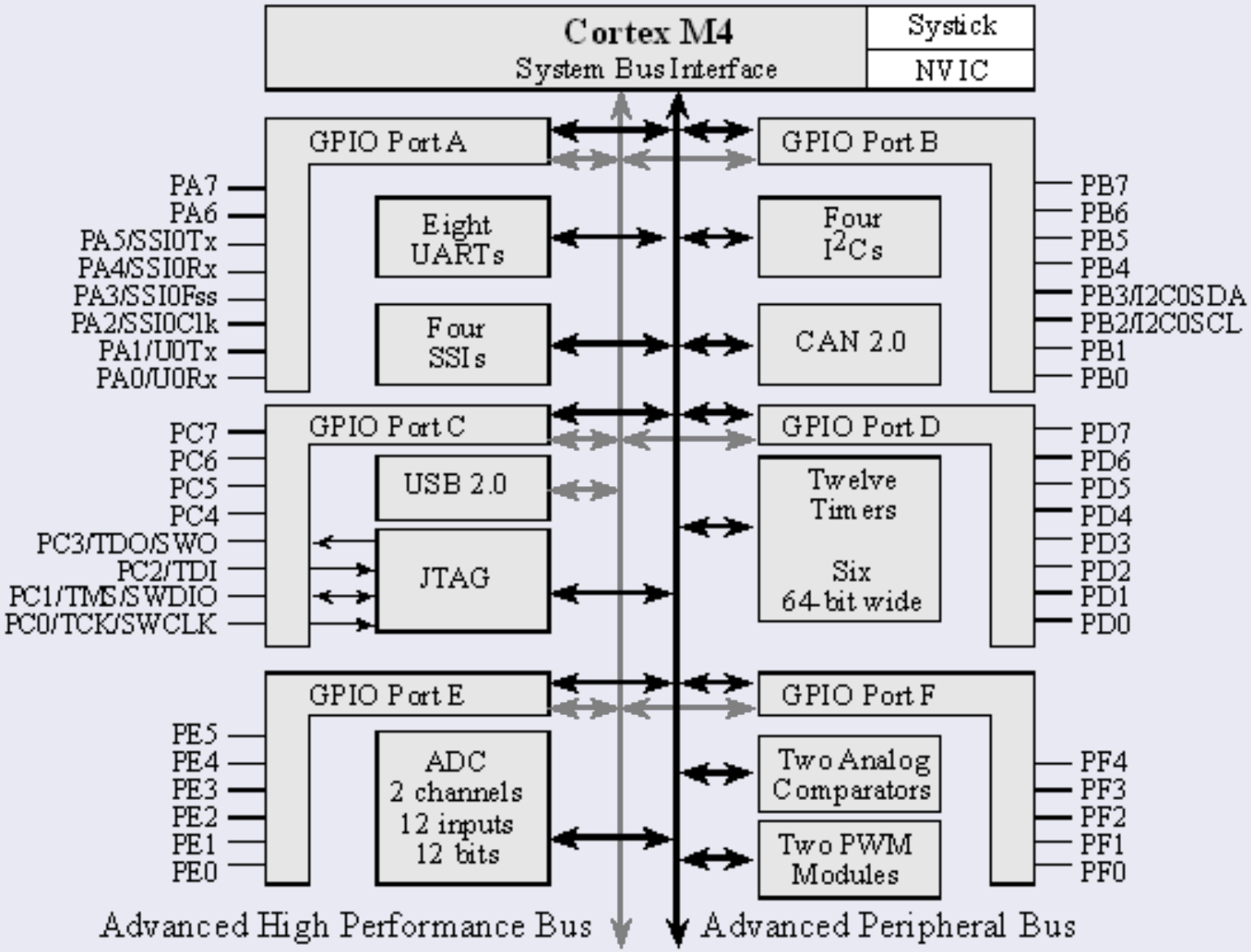


Memory Map I/O Ports



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

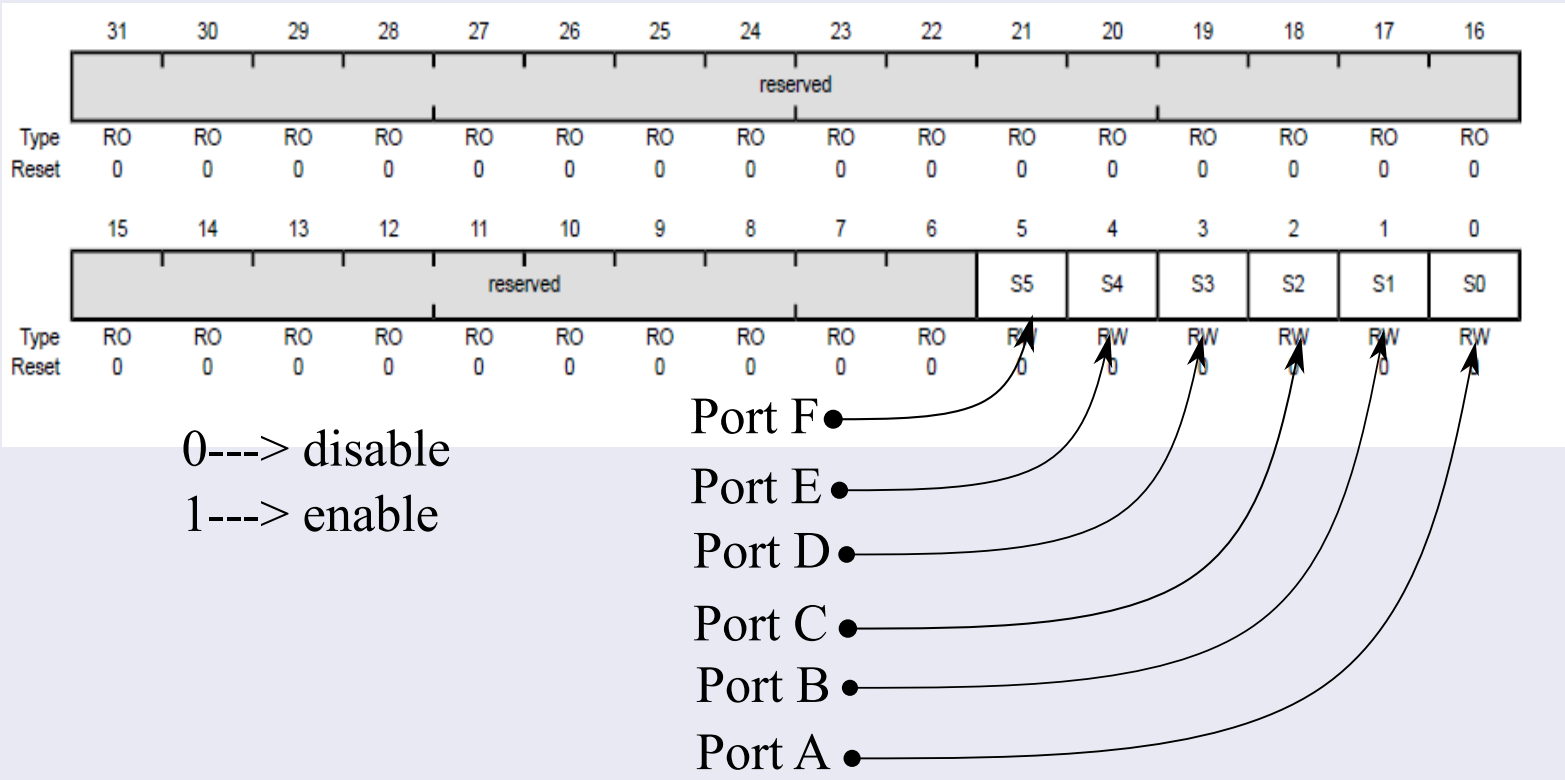
Seven Steps to Initialize a Port

- 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

Register Information

- Activate the clock gating register



Debugger Mode Memeory View SCGCGPIO Register and F Port

F Port Before Setting bit 5 of SCGCPIO register

Symbolic Memory

Go to: 0x400FE608 | Memory | Previous | Next

Location	Data	Variable	Value	Type
0x400FE5FC	0x00000000			
0x400FE600	0x00000000			
0x400FE604	0x00000000			
0x400FE608	0x00000000			
0x400FE60C	0x00000000			
0x400FE610	0x00000000			
0x400FE614	0x00000001			

Memory 1

Go to 0x40025000 | Memory

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

Dibugger Mode Memeory View SCGCGPIO Register and F Port

F Port after Setting bit 5 of SCGCPIO register

Symbolic Memory

Go to: 0x400FE608 | Memory | Previous | Next

Location	Data	Variable	Value	Type
0x400FE5FC	0x00000000			
0x400FE600	0x00000000			
0x400FE604	0x00000000			
0x400FE608	0x00000020			
0x400FE60C	0x00000000			
0x400FE610	0x00000000			
0x400FE614	0x00000001			

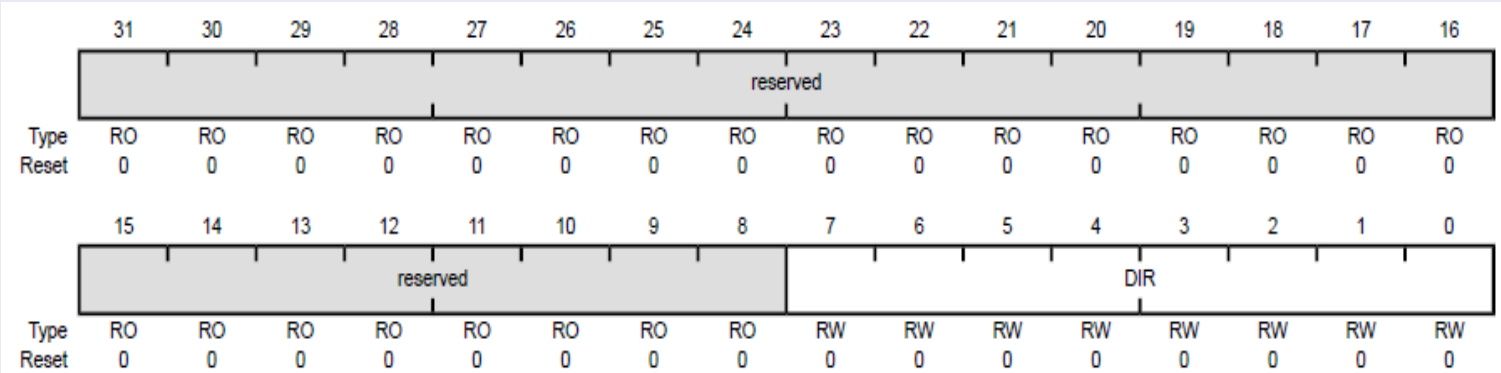
Memory 1

Go to 0x40025000 | 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

Register Information



0---> Cooresponding pin input

1---> Cooresponding pin output

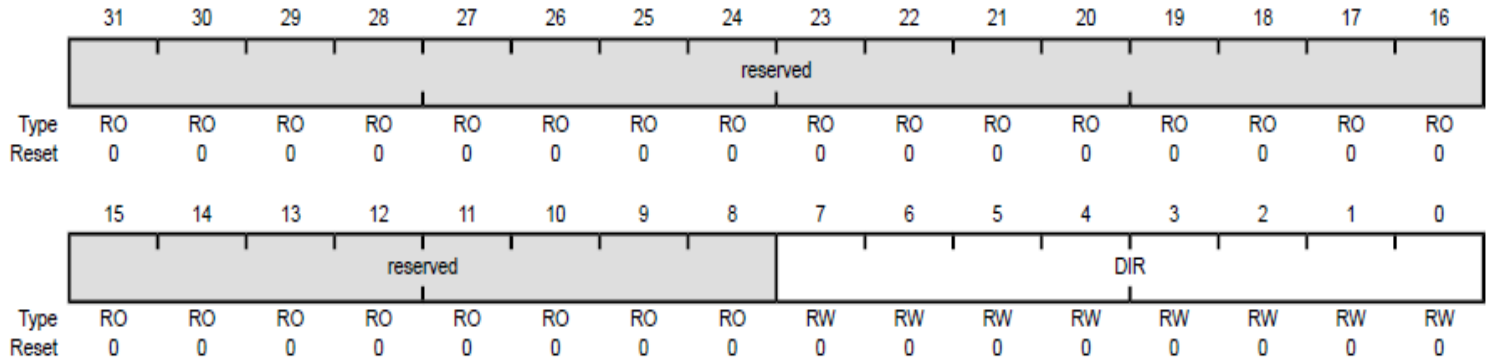
):7 Bits

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

Register Information



0---> Disable Digital function

1---> Enable Digital function

0:7 Bits

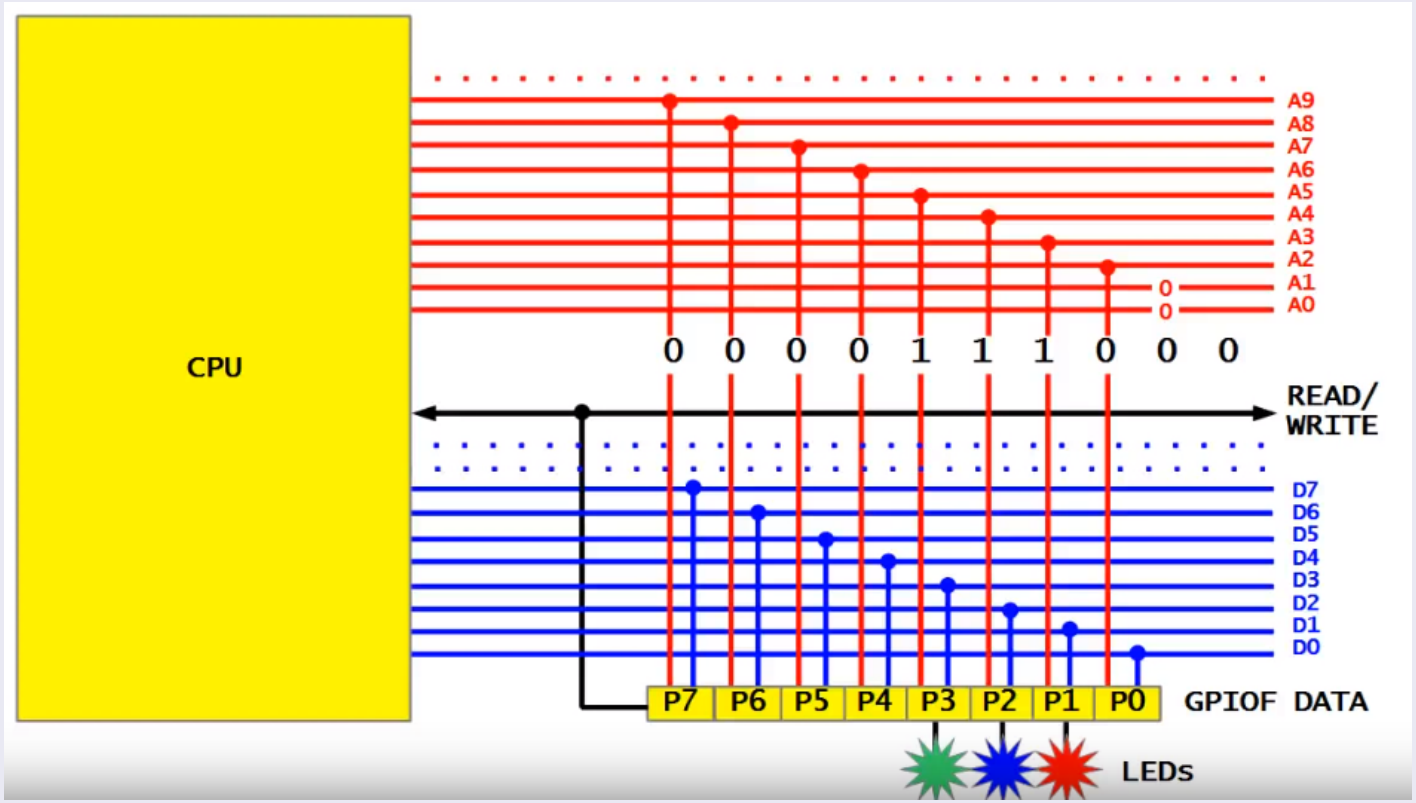
GPIO Digital Enable (GPIODEN)

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 -

GPIODATA Register

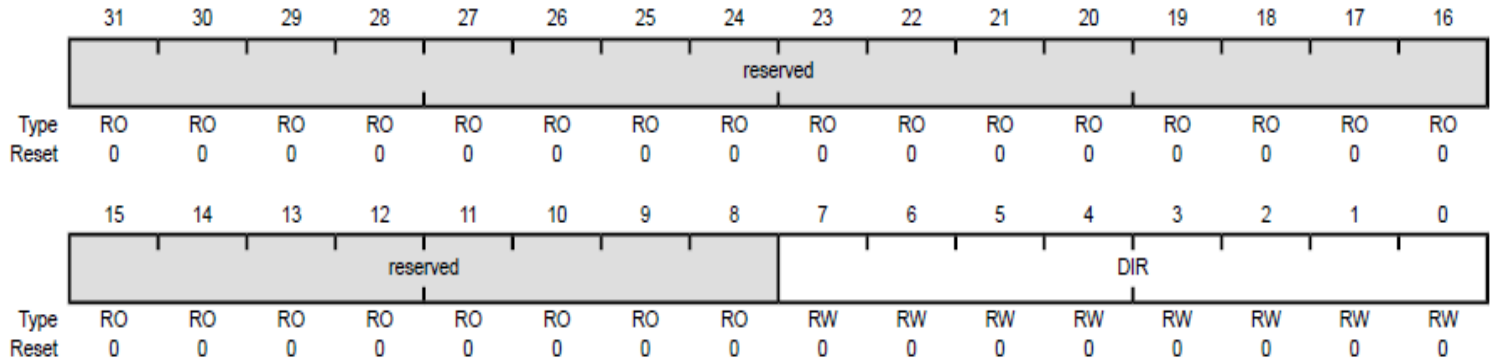
Register Information

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



GPIODATA Register

Register Information



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

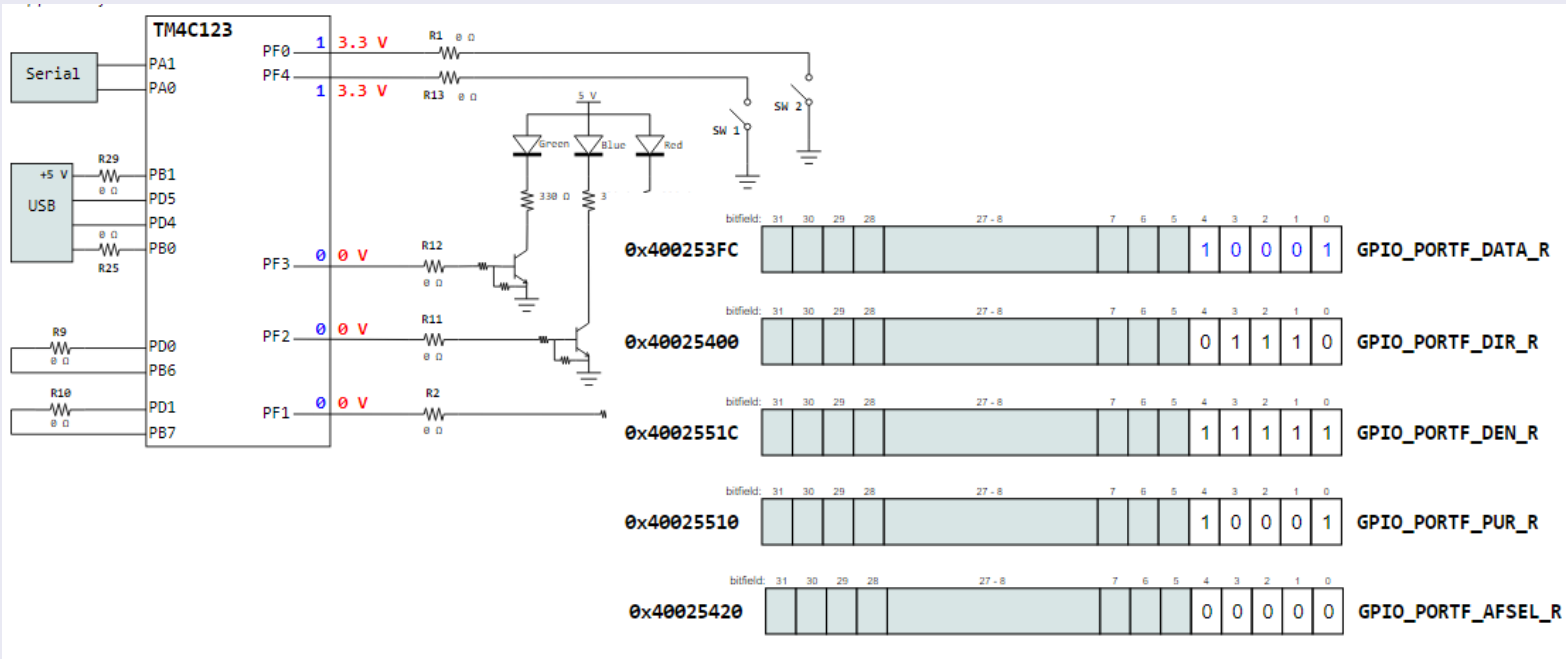
3FC

0x400253FC

GPIO Data (GPIODATA)

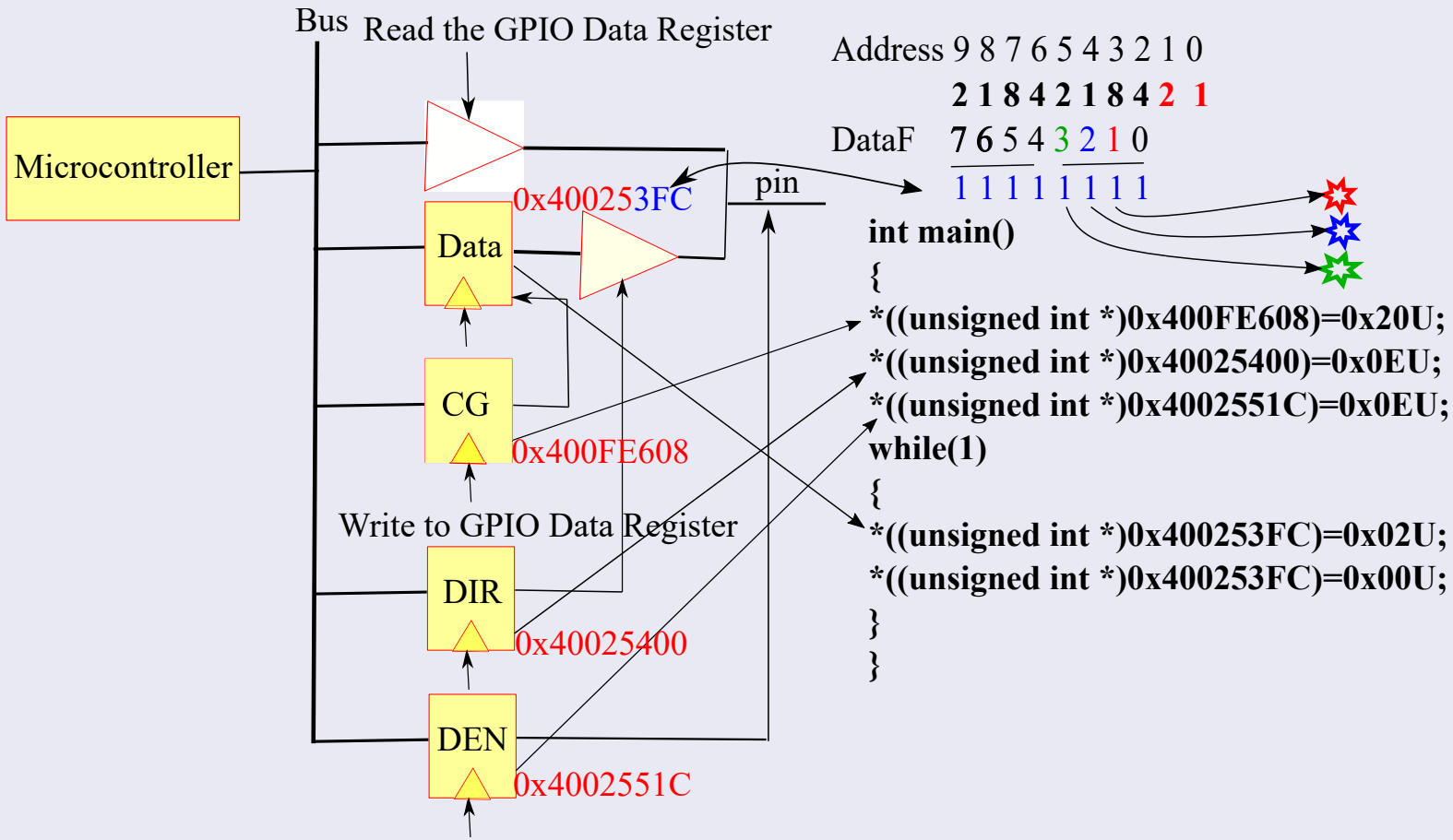
GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) 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



Red LED Glow Program

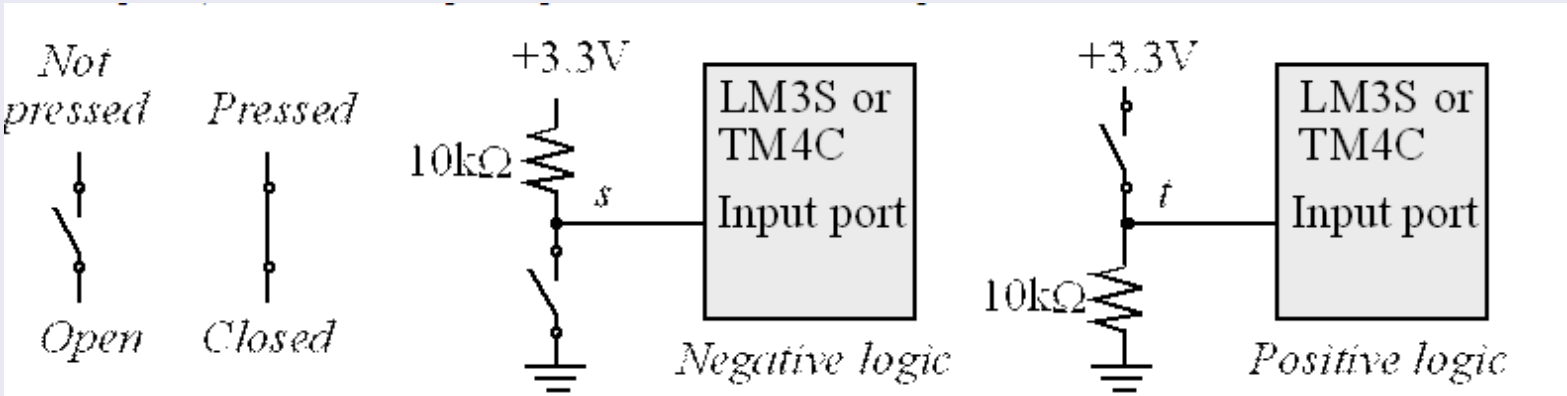
Simple Program



PULLUP and PULL DOWN Resistance

Switch Interfacing

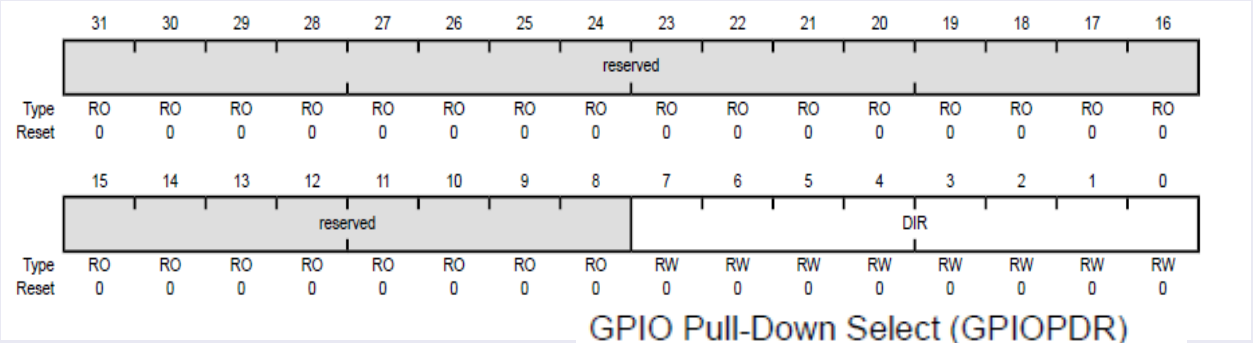
- Pull-up and Pull-down resistors are used to correctly bias the inputs of digital gates to stop them from floating about randomly when there is no input condition



PULLUP Register

Register Information

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



0:7 Bits

PortF PULL UP Register Set
0x40025514=0x10

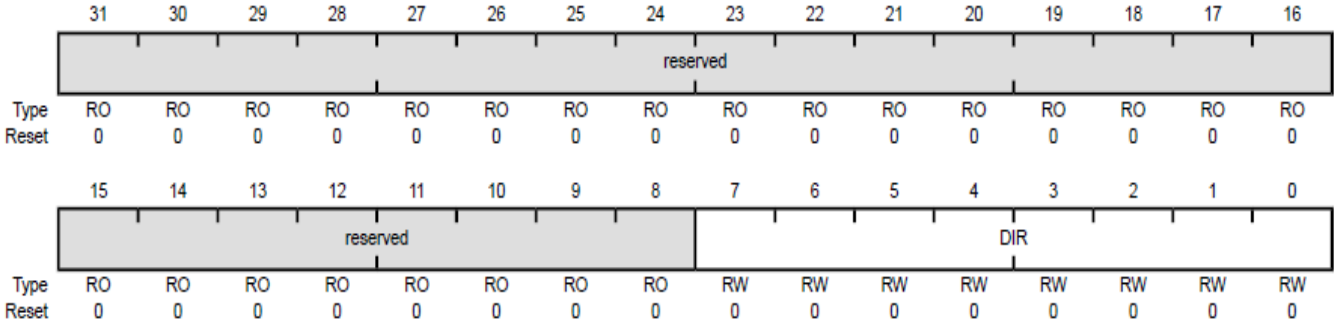
GPIO Pull-Down Select (GPIOPDR)

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.



0:7 Bits

Default value is 0xFF

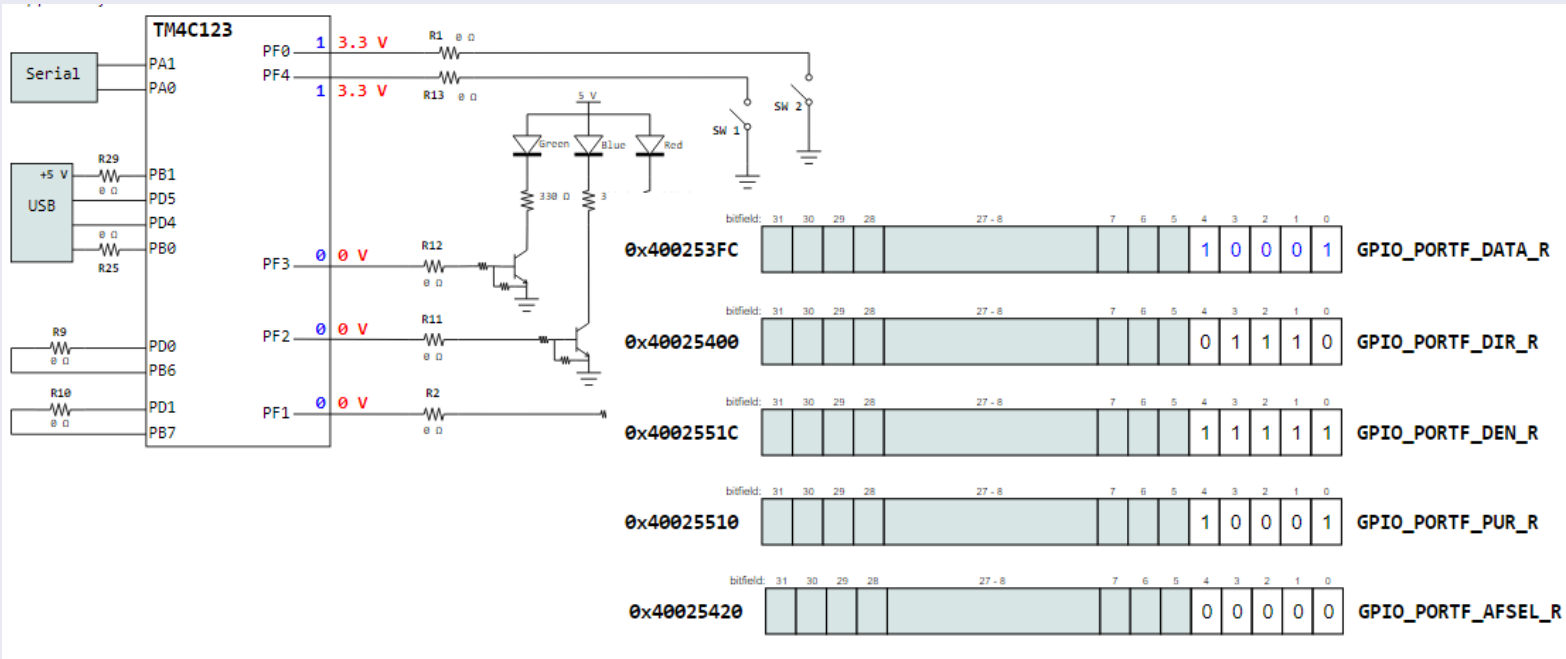
0x40025524=0xFF

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

GPIO Commit (GPIOCR)

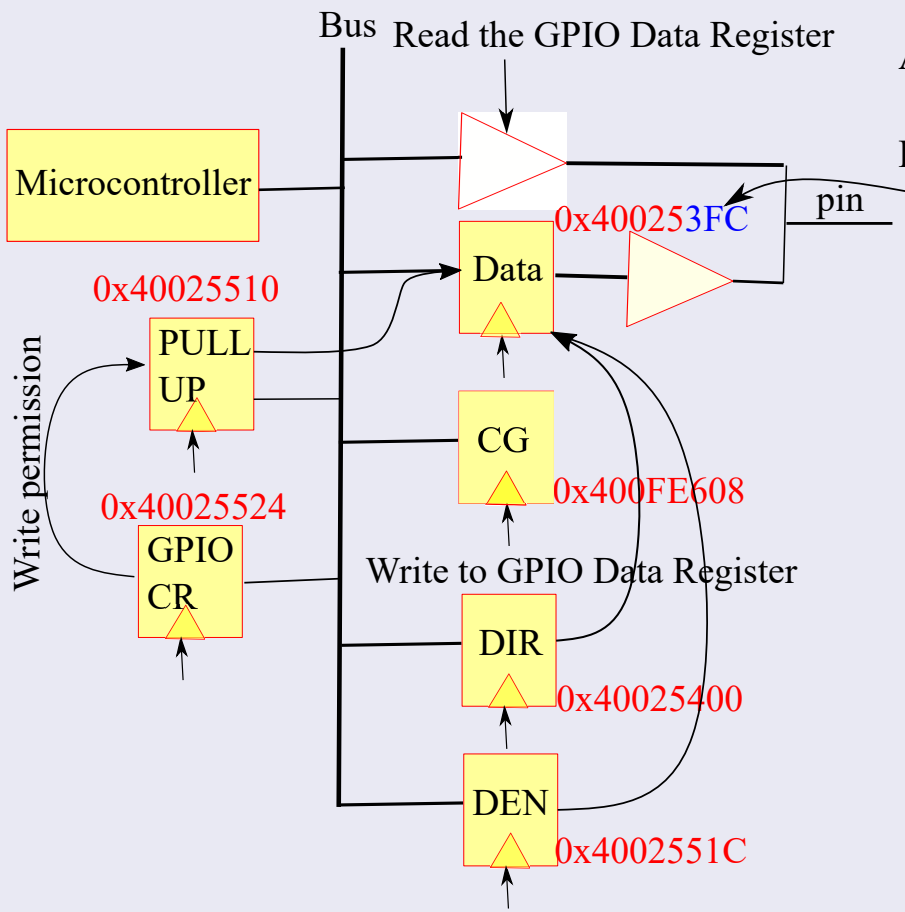
GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) 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 -

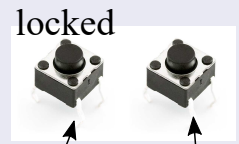
Pin Diagram of TM4C123



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
DataF 7 6 5 4 3 2 1 0
1 1 1 1 1 1 1 1

```
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

Register Setting

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

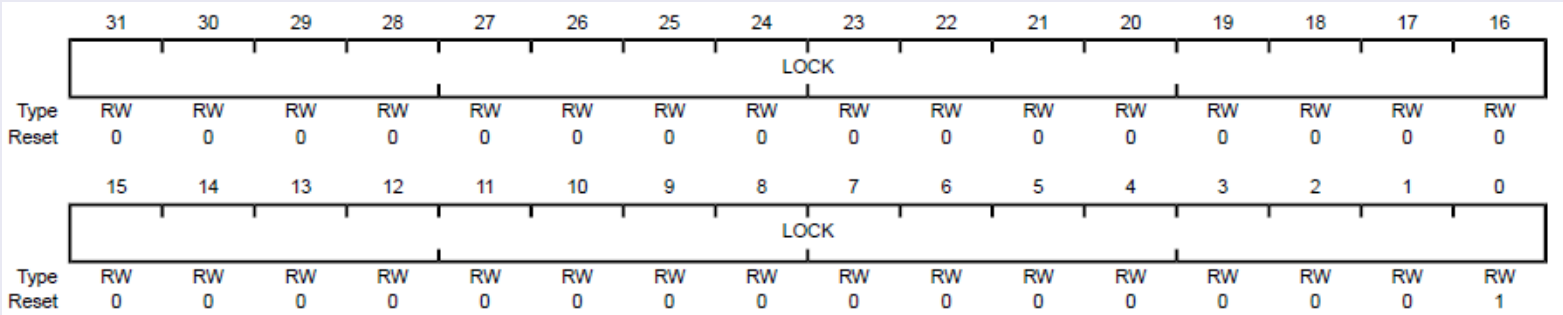
Table 10-1. GPIO Pins With Special Considerations

GPIO Pins	Default Reset State	GPIOAFSEL	GPIODEN	GPIOPDR	GPIOPUR	GPIOPCTL	GPIOCR
PA[1:0]	UART0	0	0	0	0	0x1	1
PA[5:2]	SSI0	0	0	0	0	0x2	1
PB[3:2]	I^2C0	0	0	0	0	0x3	1
PC[3:0]	JTAG/SWD	1	1	0	1	0x1	0
PD[7]	GPIO[a]	0	0	0	0	0x0	0
PF[0]	GPIO[a]	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

GPIOLOCK



0:31 Bits

To Unlock the Special Port
0x40025520=0x4C4F434B

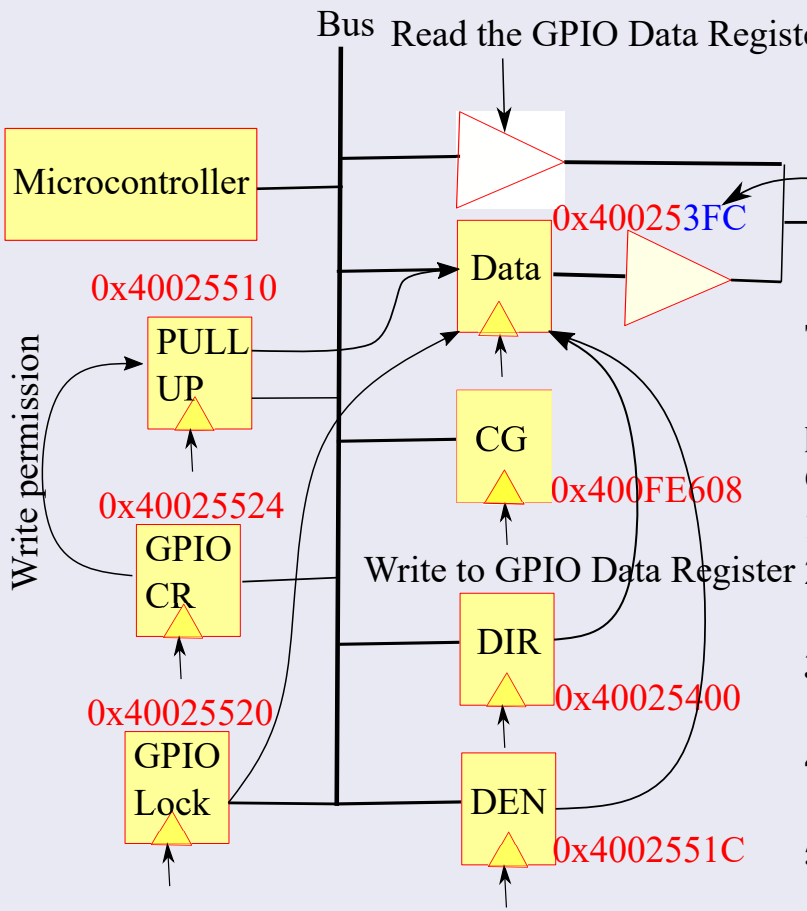
Default value=0x4C4F434B

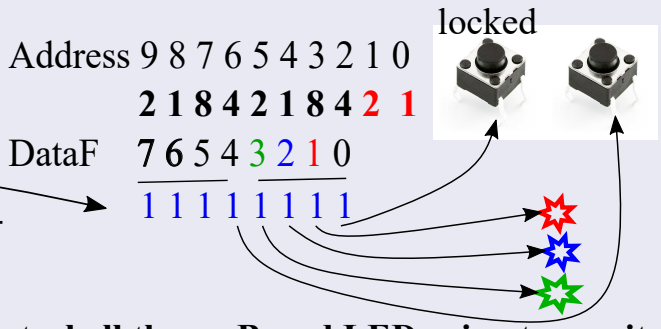
GPIO Lock (GPIOLOCK)

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

Write a Program



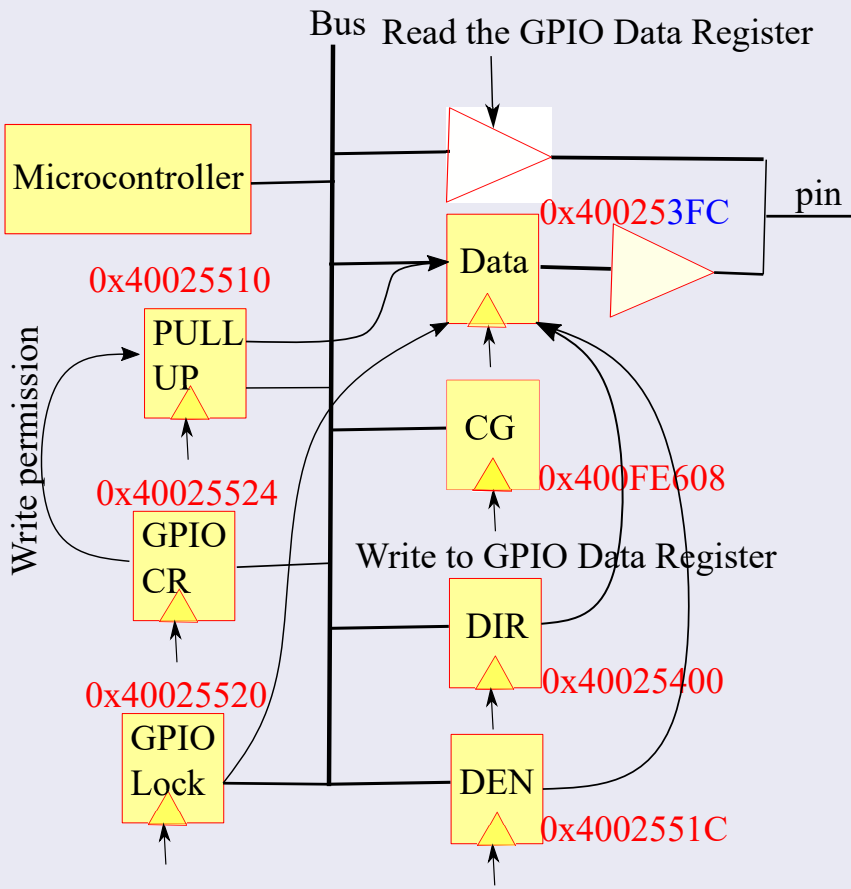


To control all the on Board LED using two switch at Port F write the program. For each pin, i need specific address to be calcutated.
Given

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∧ b; Exclusive OR
- c=b>>1; right shift
- c=b<<1; left shift
- c= ¬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

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=0x0E; // DIR|=(RED|BLUE|GREEN) = DIR|=((1U<<1)|(1U<<2)|(1U<<3))
 DEN=0x0E; // 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	
X X X X X X T X	reg
0 0 0 0 0 0 1 0	mask
X X X X X X 1 X	reg set

AND to Clear	
X X X X X X T X	reg
1 1 1 1 1 1 0 1	mask
X X X X X X 0 X	reg clear

Read/Write/Modify Issue During the Interrupt

Bitwise Operaotrs

- 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 GPIODATA register bits may be modified.
- After the completion of ISR, the processor resume its current execution.
- Resume instruction further changes the GPIODATA bits, which lost the ISR bit change
- Through the pointer arithmetic this problem can be minimized to a certain extend

Array and Pointer

Bitwise Operaotrs

- Arm instruction read/write/modify instruction is replace with an atomic write operation
- Each bit of GPIODATA register is addressable (8 bit GPIODATA register is addressed by 256 register)
- 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 Operaotrs

```
#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;
```

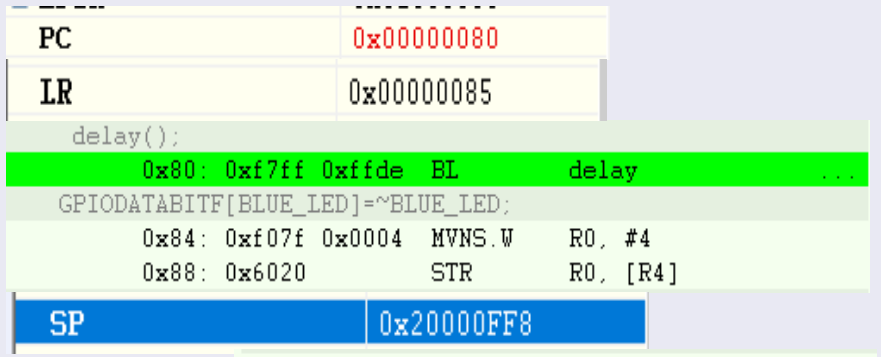
```
0x52: 0x480d        LDR.N    R0, [PC, #0x34]   ...
0x54: 0x6801        LDR      R1, [R0]
0x56: 0xf051 0x0102 ORRS.W   R1, R1, #2
0x5a: 0x6001        STR      R1, [R0]
```

```
*(GPIODATABITF+RED_LED)=RED_LED; // GPIODATABITF[RED_L...
0x54: 0x2102        MOVS     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;
}
```



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

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

16/32-Bit General-Purpose Timer Run Mode Clock Gating Control (RCGCTIMER)

Base 0x400F.E000
Offset 0x604
Type RW, reset 0x0000.0000

1->Enable
0-> Disable

	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	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Timer5 Timer3 Timer1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved										R5	R4	R3	R2	R1	R0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Timer4 Timer2 Timer0

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;
 SYSCTL_RCGCTIMER_R|=(1U<<0); //clock gating on timer circuit
 TIMER0_CTL_R &=~(1<<0);     // diable 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

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	
.		.	.	generted by user program or pheripheral
.		.	.	
.		.	.	
		0x004C		
18	2	0x0048	IRQ2	
17	1	0x0044	IRQ1	
16	0	0x0040	IRQ0	
15	-1	0x003C	Systick	set timer generated
14	-2	0x0038	PendSV	software inteerupt (ICSR)
13			Reserved	
12			Reserved for Debug	debug when no halting
11	-5	0x002C	SVCall	supervised form os to kernal
10			Reserved	
9				
8				
7				
6	-10	0x0018	Usage fault	user generated fault
5	-11	0x0014	Bus fault	memory related fault
4	-12	0x0010	Memory management fault	memory protection
3	-13	0x000C	Hard fault	error during exception processing
2	-14	0x0008	NMI	software inteerupt (ICSR)
1		0x0004	Reset	Reset the processor
		0x0000	Initial SP value	

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

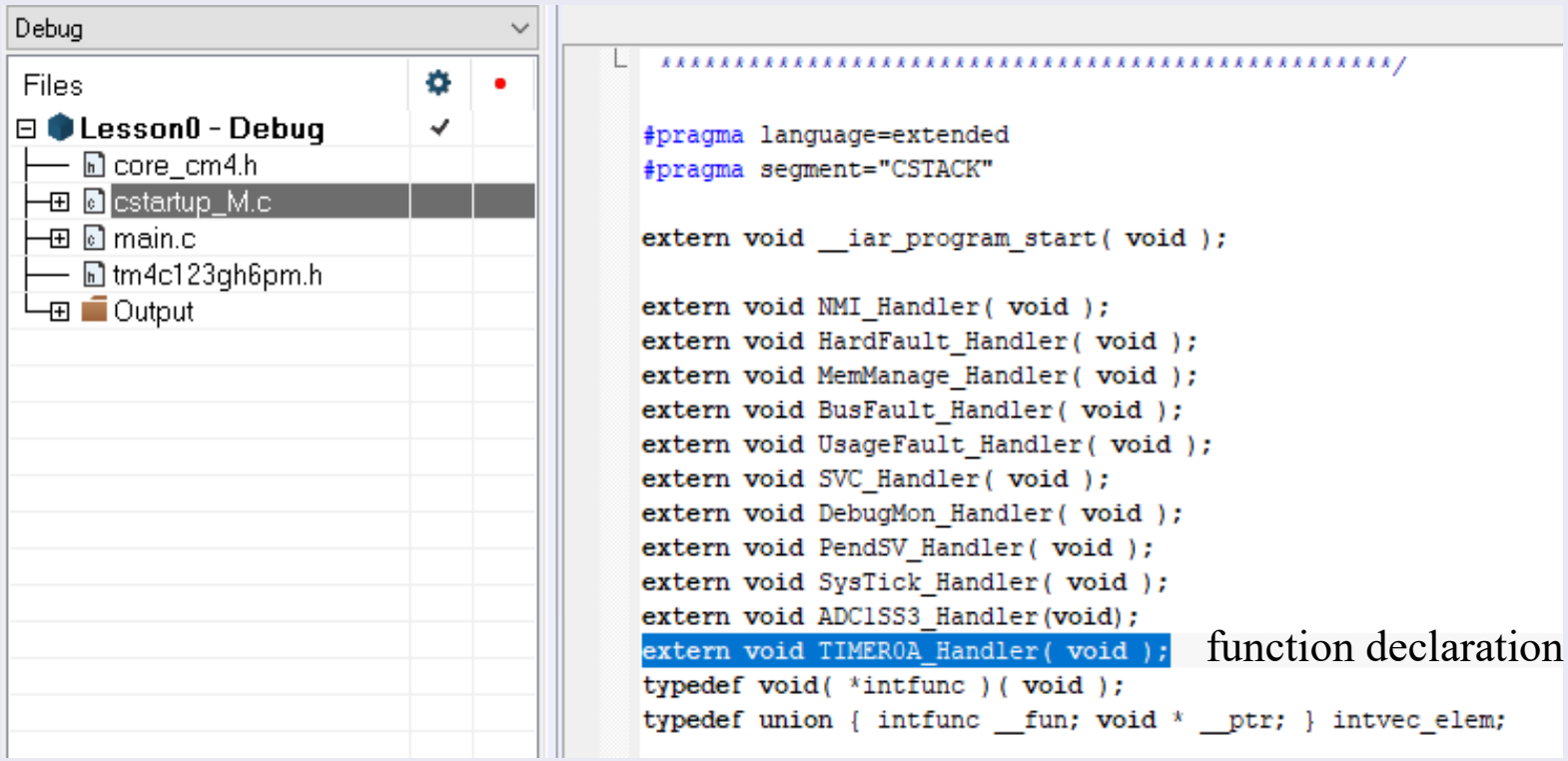
Vector Number	Interrupt Number (Bit in Interrupt Registers)	Vector Address or Offset	Description
0-15	-	0x0000.0000 - 0x0000.003C	Processor exceptions
16	0	0x0000.0040	GPIO Port A
17	1	0x0000.0044	GPIO Port B
18	2	0x0000.0048	GPIO Port C
19	3	0x0000.004C	GPIO Port D
20	4	0x0000.0050	GPIO Port E
21	5	0x0000.0054	UART0
22	6	0x0000.0058	UART1
23	7	0x0000.005C	SSI0
24	8	0x0000.0060	I^2C0
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	QEI0
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 is 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 get in the interrupt vector table
  - Lastly, function definition.

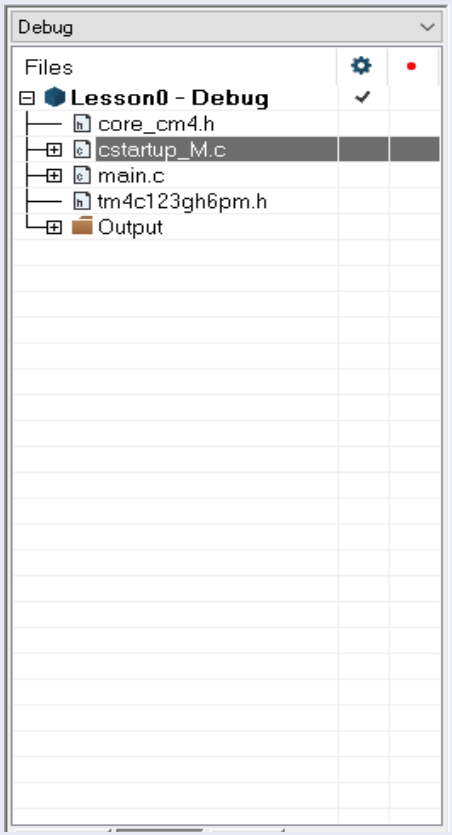
Function Addition in cstartup_M.c vector file

Function declaration



Function Addition in cstartup_M.c vector file

Function Entry in Vector Table

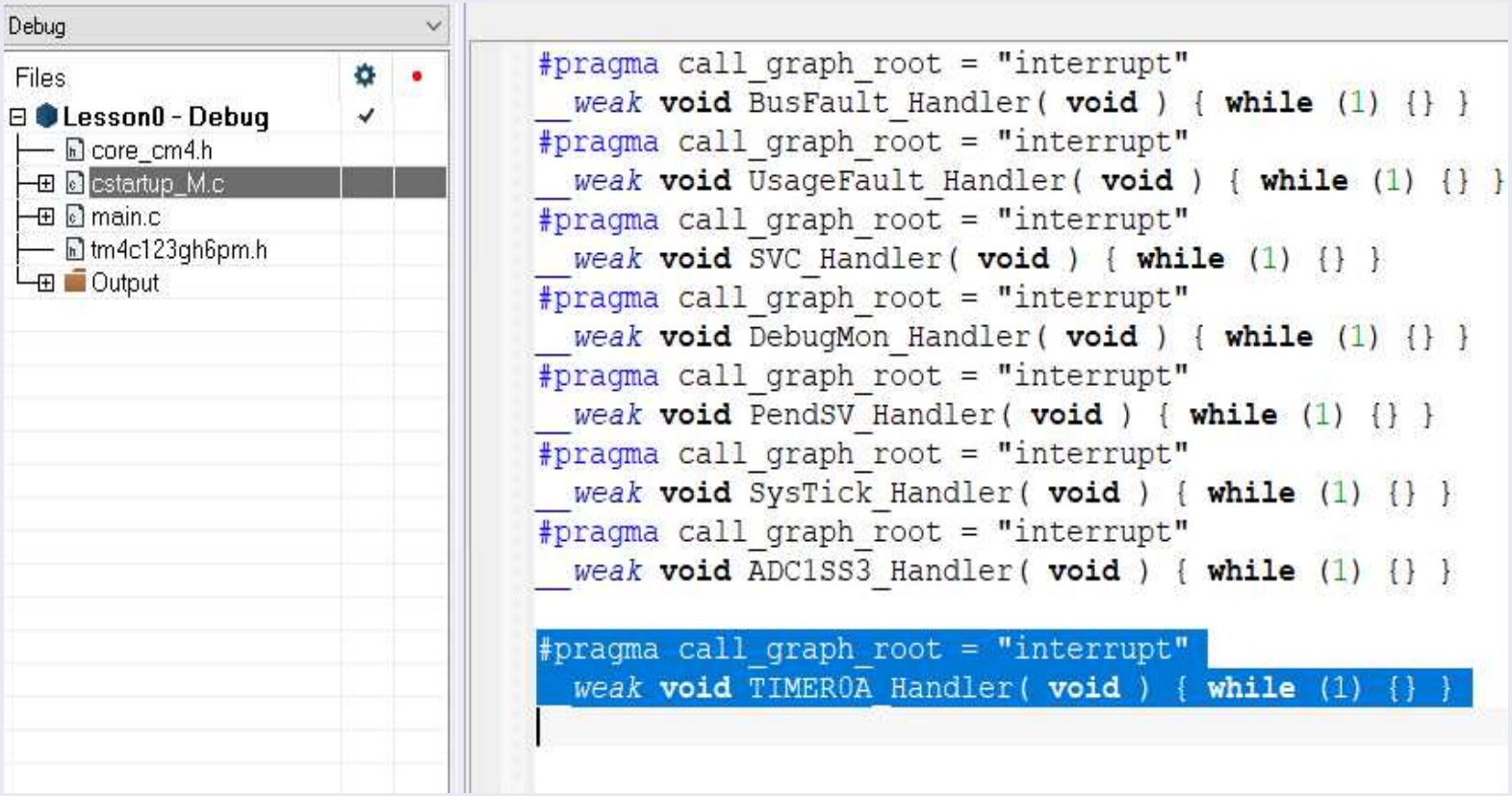



```
__vector_table
    SVC_Handler,
    DebugMon_Handler,
    0,
    PendSV_Handler,
    SysTick_Handler,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    TIMER0A_Handler, //19 bit of NVIC register
    ADC1SS3_Handler
```

function entry

Function Addition in cstartup_M.c vector file

Function Definition



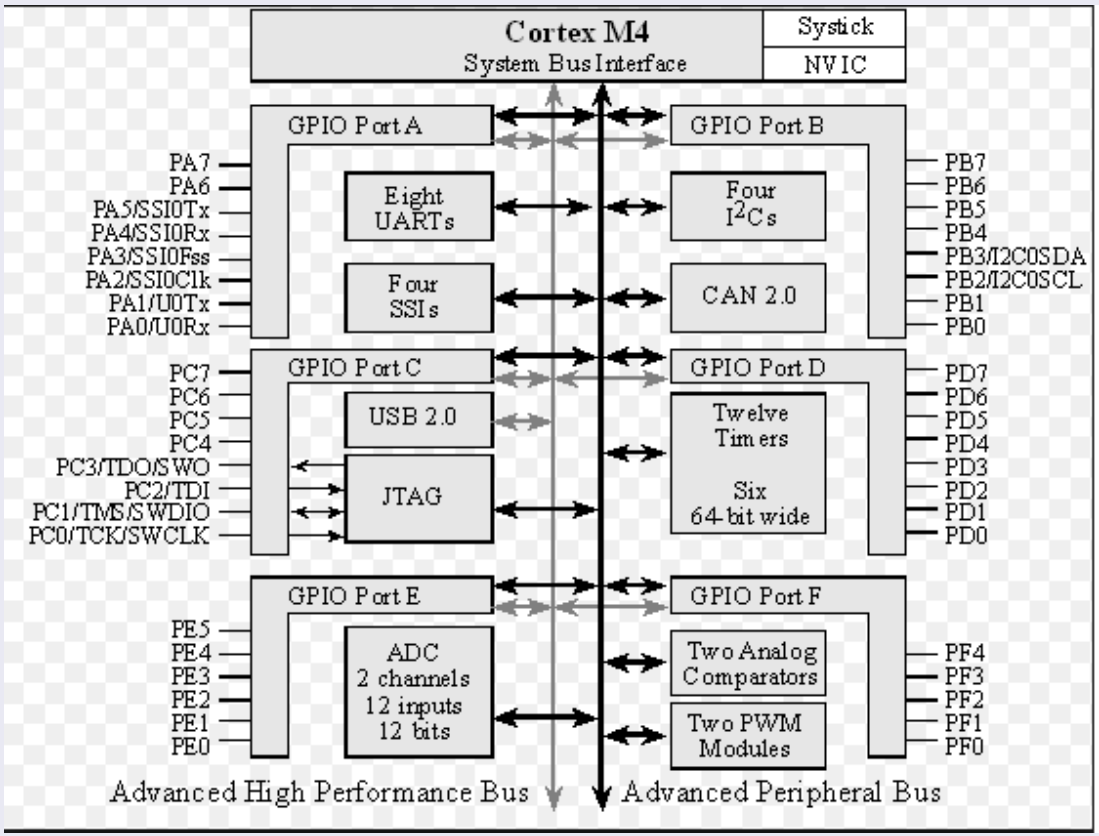
Timer Interrupt Based LED Blink

Complete Program

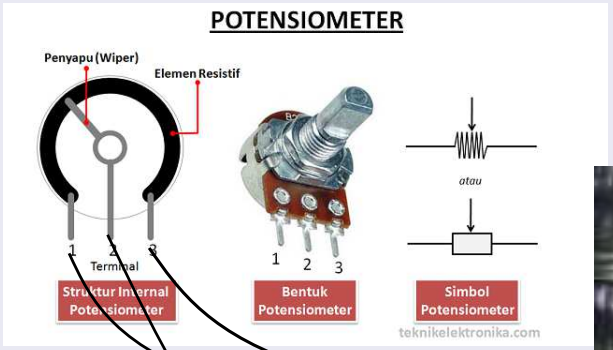
```
#include"tm4c123gh6pm.h"
#define CG *((volatile unsigned int *)0x400FE608) // clock gating
#define DIR *((volatile  unsigned int *)0x40025400)// direction register (b01110)->portF
#define DEN *((volatile unsigned int *)0x4002551C) //digital enable (b11111)->portF
#define DATAF *((unsigned int *)0x400253FC)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



Analog To Digital Convertors Connections



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

- An analog-to-digital converter (ADC) is a peripheral that converts a continuous analog voltage to a discrete digital number.
- Important Terms
  - Amplitute
  - Amplitute 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

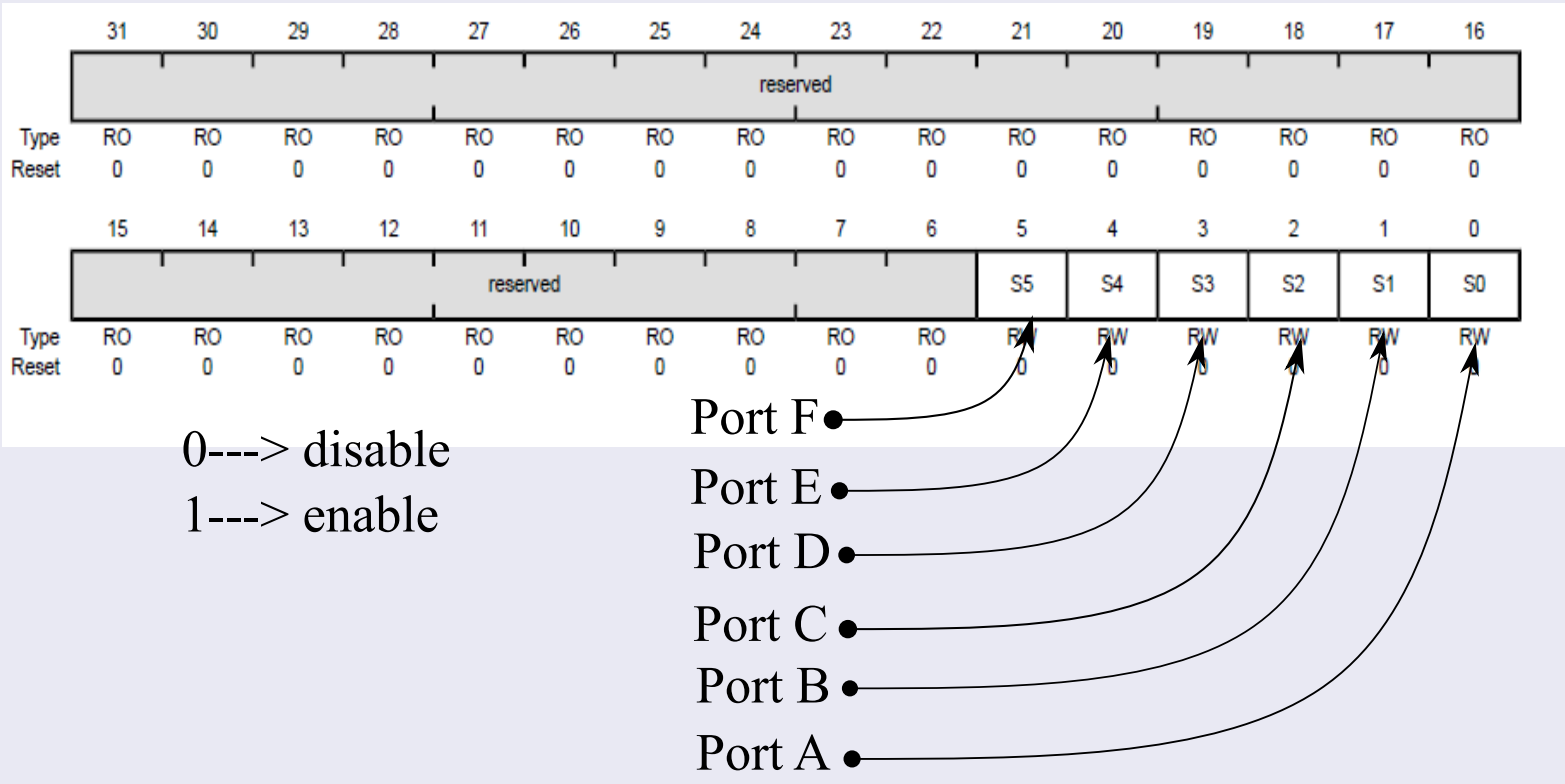
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved														R1	R0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Value	Description
0	ADC module 0 is disabled.
1	Enable and provide a clock to ADC module 0 in Run mode.

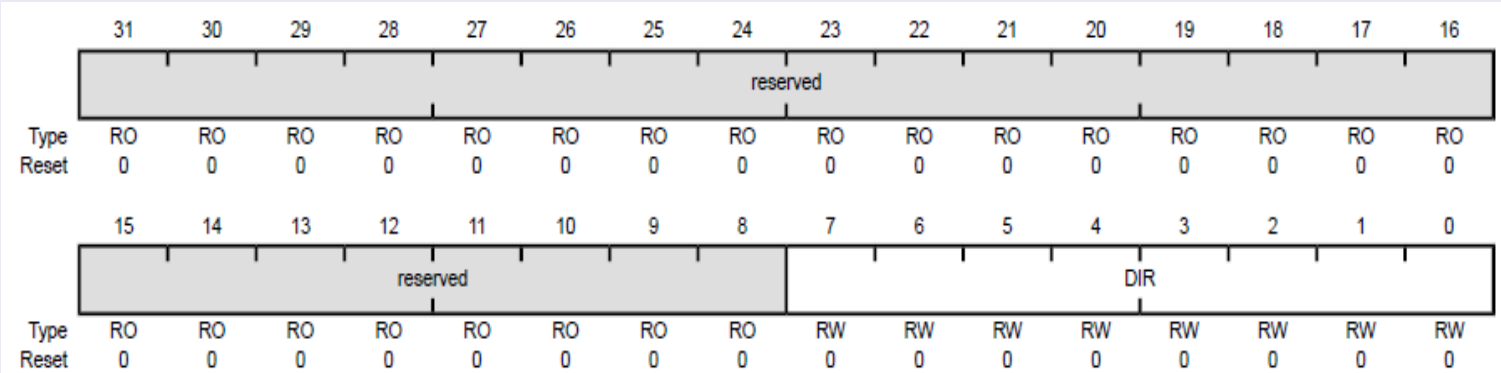
Initialization of GPIO PORT

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



GPIODIR Direction Register

Register Information



0---> Cooresponding pin input

1---> Cooresponding pin output

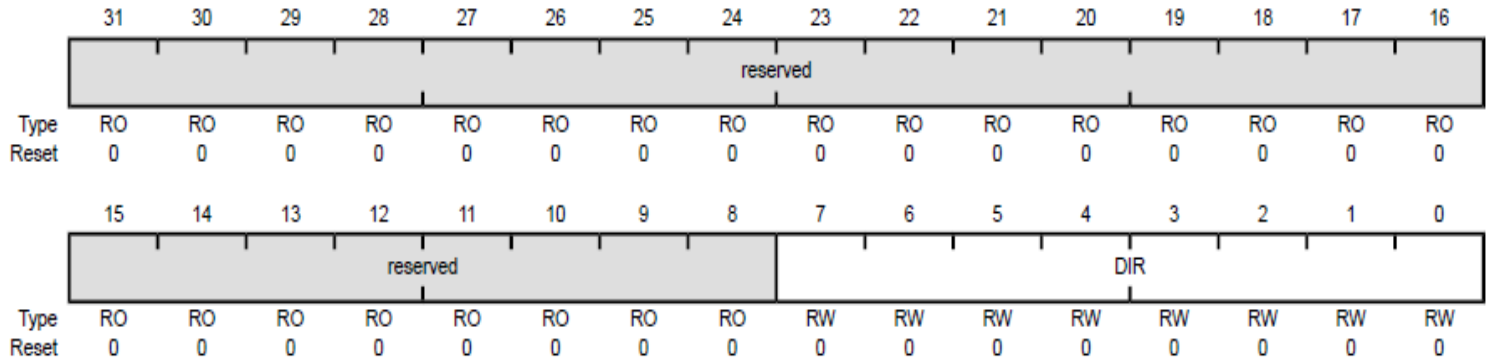
):7 Bits

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

Register Information



0---> Disable Digital function

1---> Enable Digital function

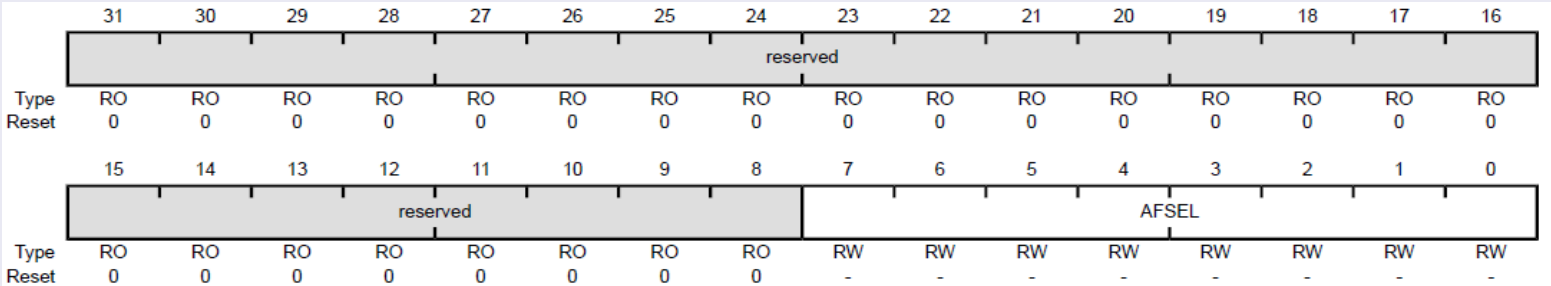
0:7 Bits

GPIO Digital Enable (GPIODEN)

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)

Register Information



GPIO Alternate Function Select (GPIOAFSEL)

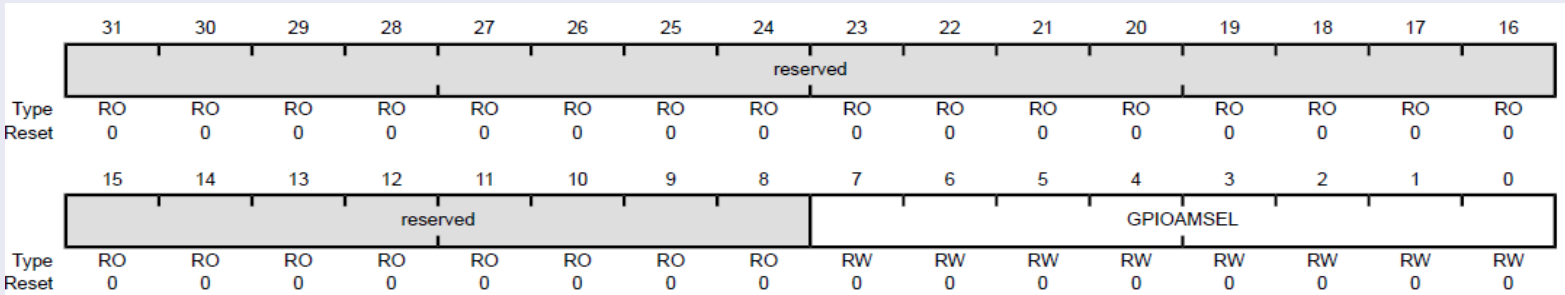
GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
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 -

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



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

Register Information

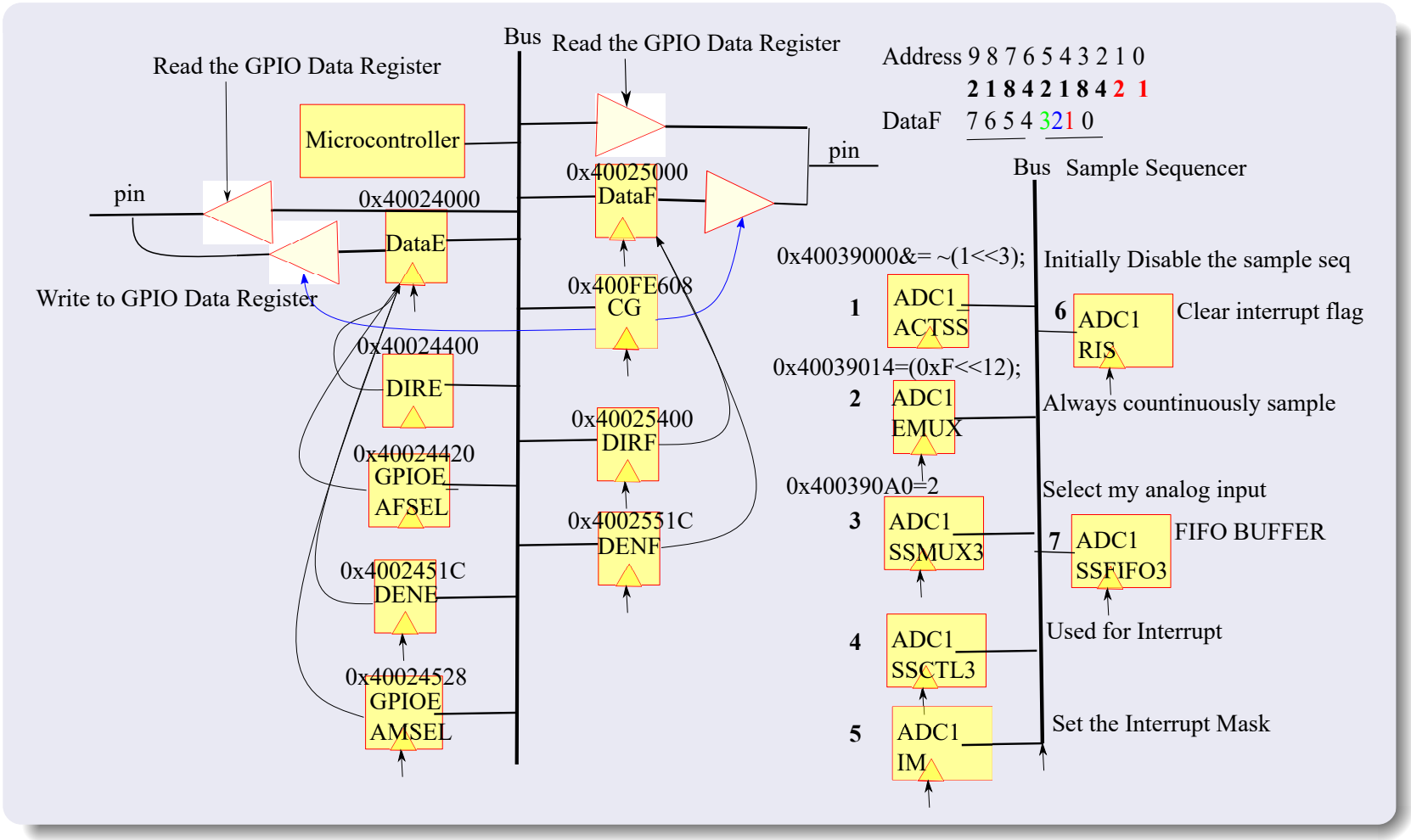
Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type[a]	Description
AIN0	6	PE3	I	Analog	Analog-to-digital converter input 0.
AIN1	7	PE2	I	Analog	Analog-to-digital converter input 1.
AIN2	8	PE1	I	Analog	Analog-to-digital converter input 2.
AIN3	9	PE0	I	Analog	Analog-to-digital converter input 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

Types of Smaple 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

```
#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->RCGCGPIO = (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 sequenser 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

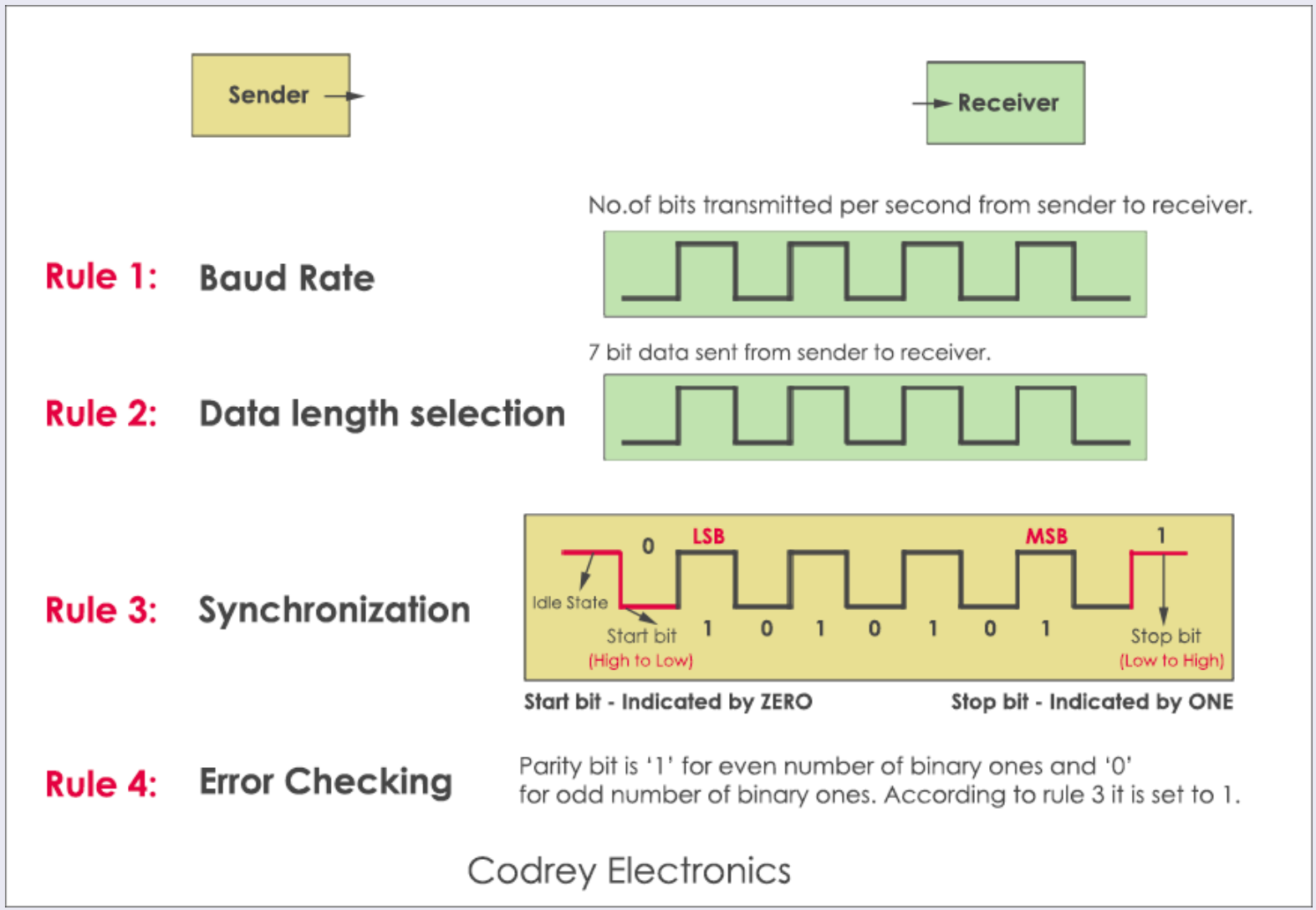
cstartup_M.c vector file

```
__vector_table
    0,
    TIMER0A_Handler,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0, //50
    ADC1SS3_Handler
};
```

```
extern void TIMER0A_Handler(void);
extern void ADC1SS3_Handler(void);
```

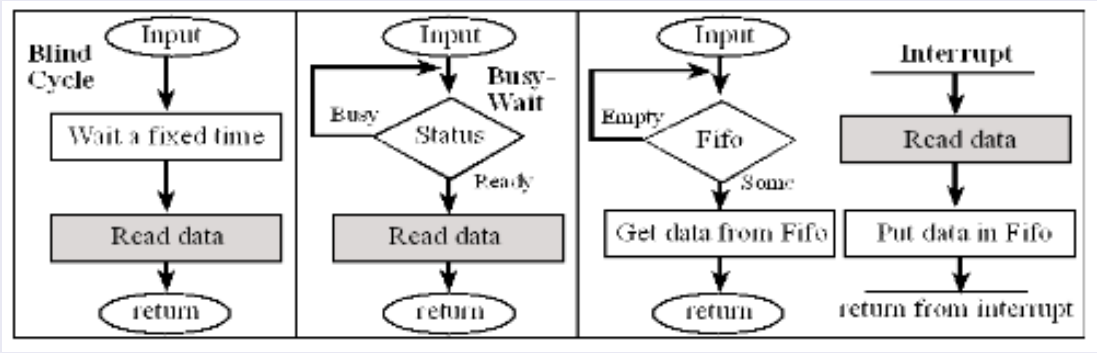
```
#pragma call_graph_root = "interrupt"
__weak void ADC1SS3_Handler( void ) { while (1) {} }
```

UART Communication To Compunicate with Computer



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.



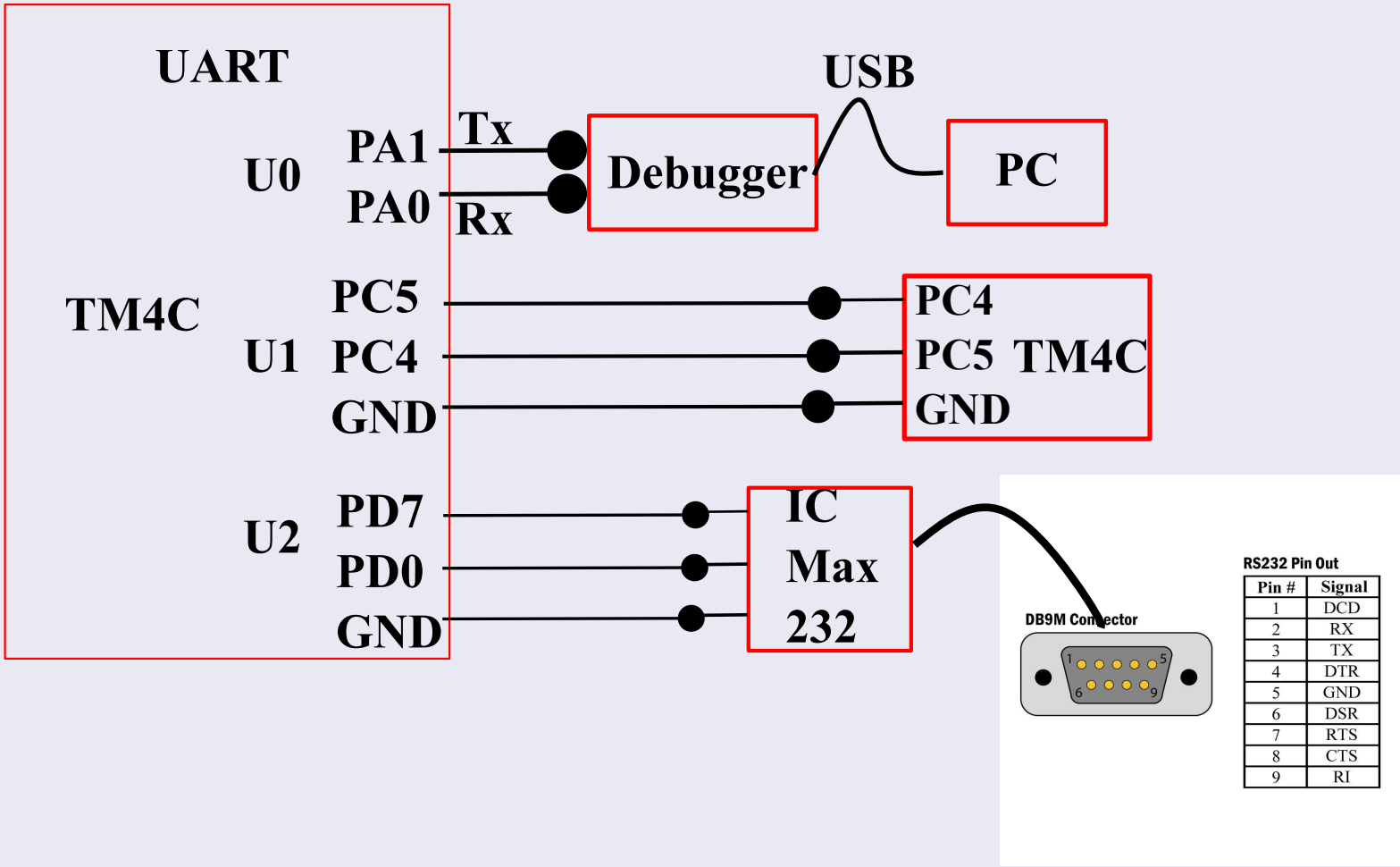
I/O Synchronization

- **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

- **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

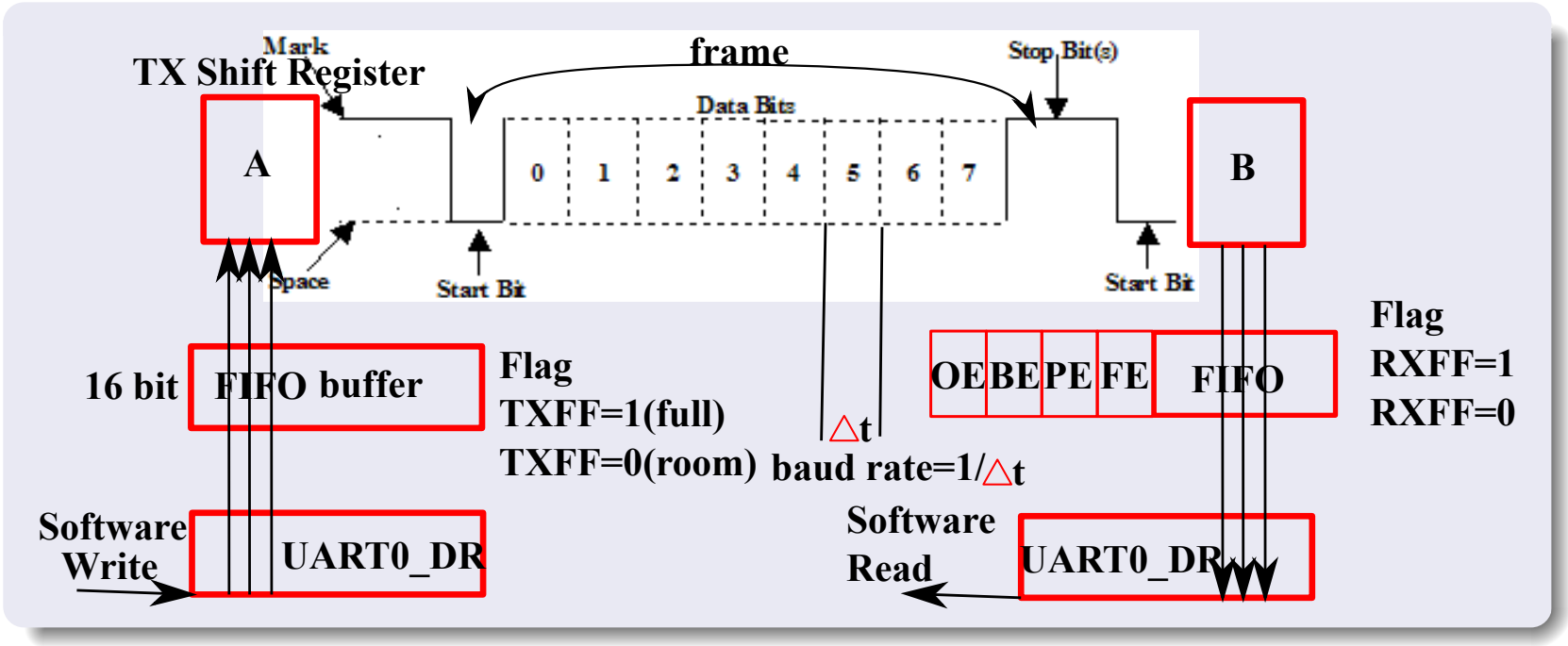
UART Communication Port On TM4C



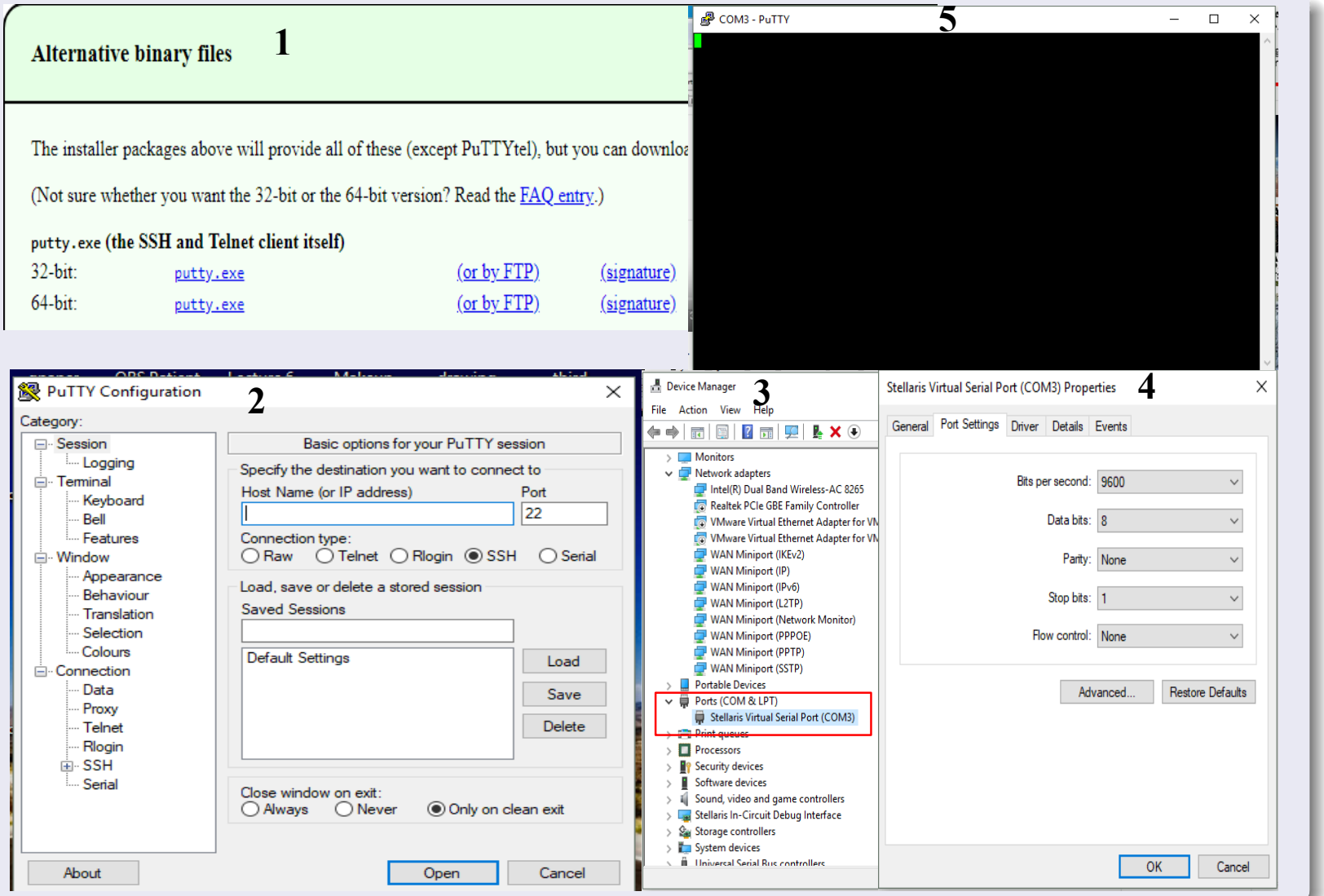
UART Communication

- Software write the data into the USRT0_DR register
- Data later be transfered to FIFO buffer before it transfer to the TX shift register
- Later the each bit is transfered
- 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

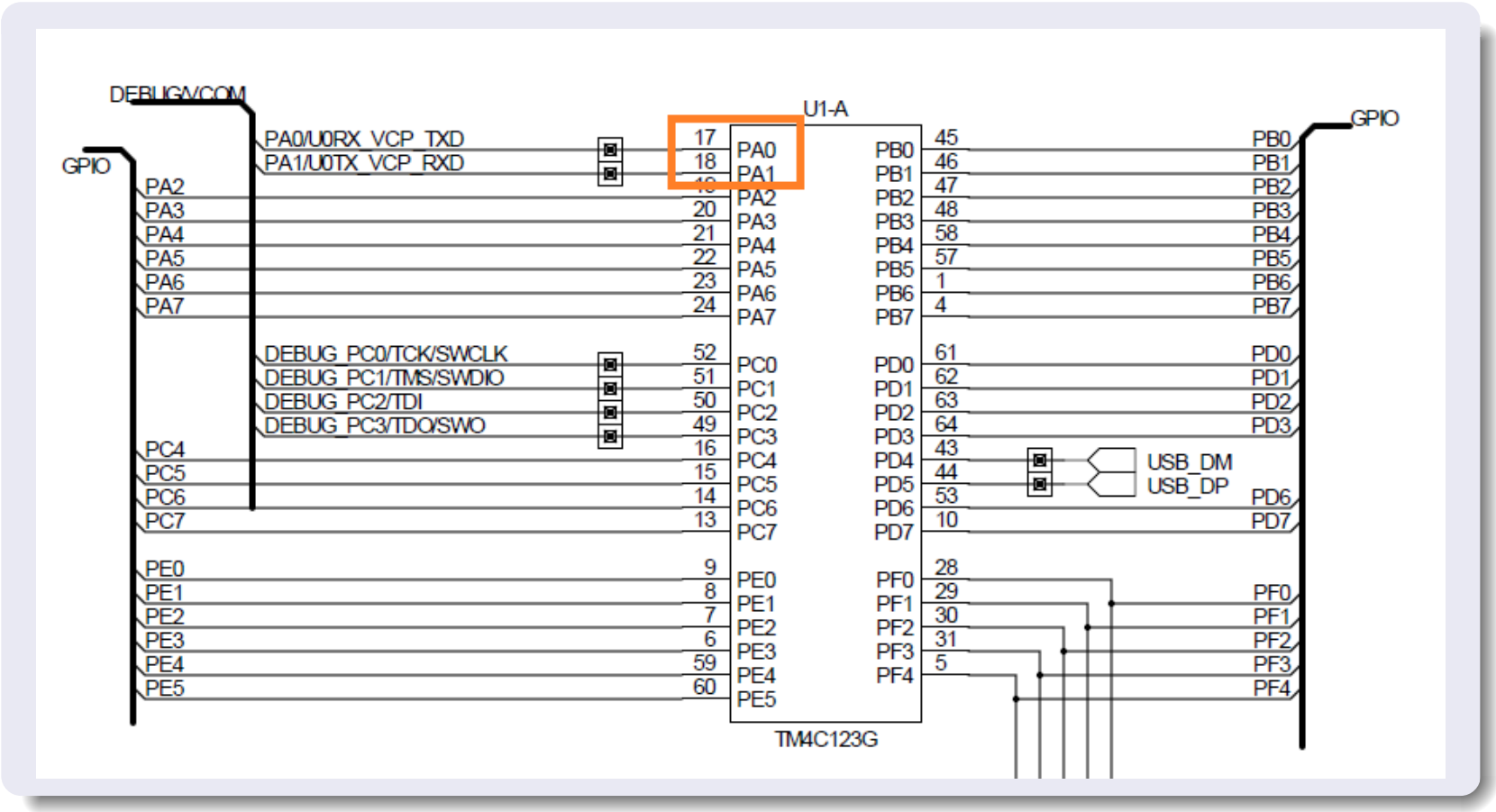
UART Communication



UART Communication Port Configuration

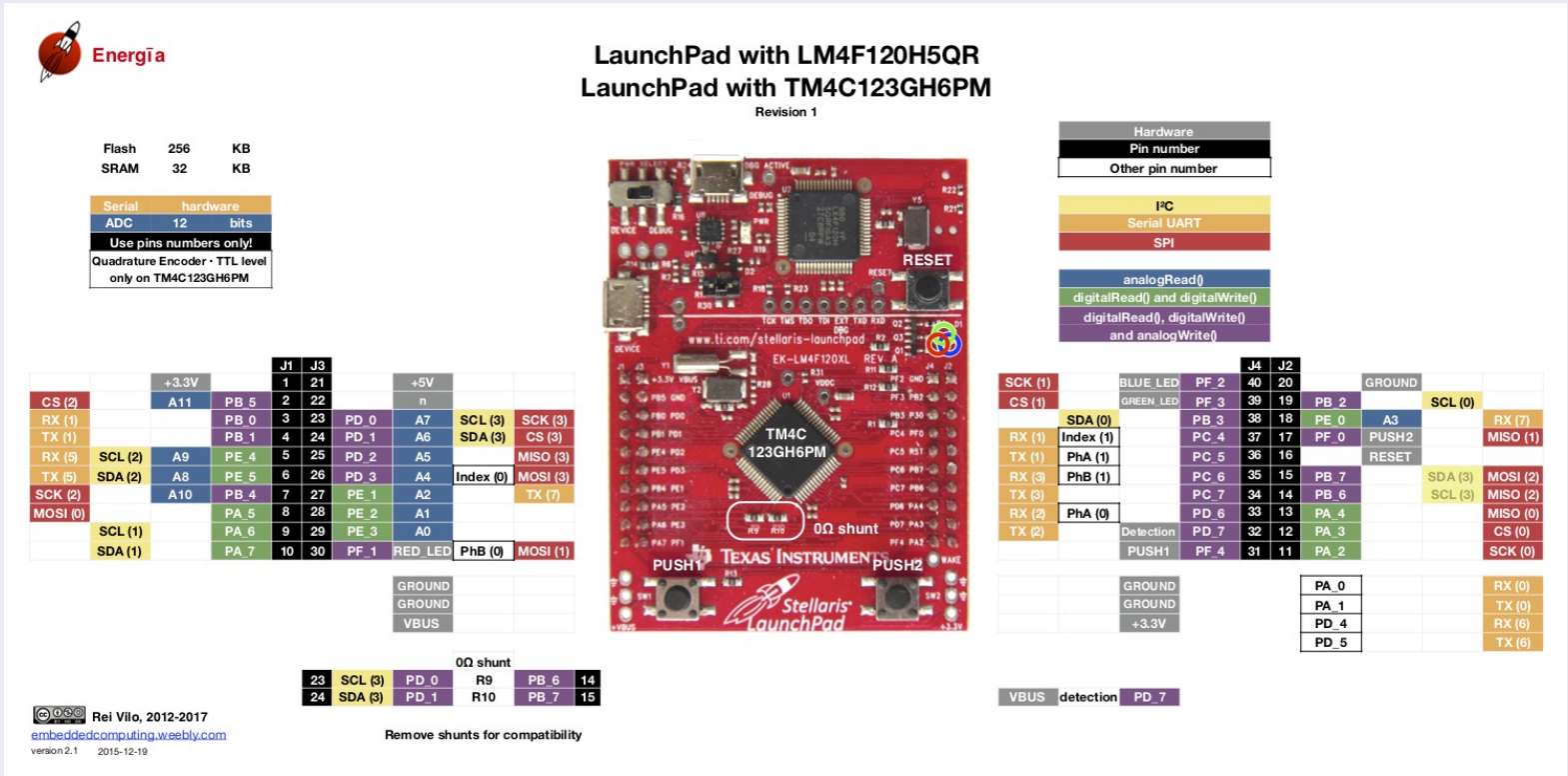


UART Communication Port On TM4C



UART Communication Port On TM4C

TOP VIEW

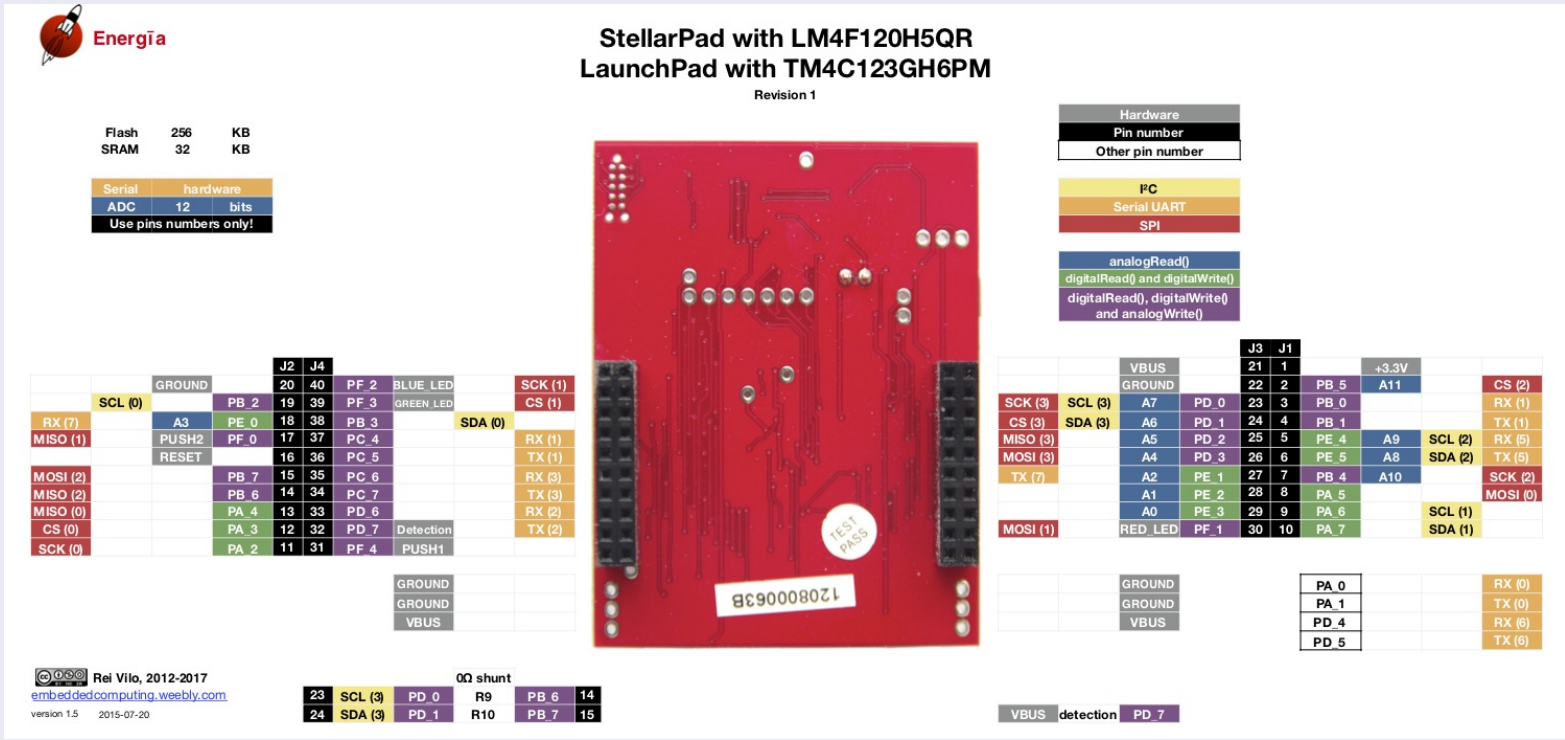


UART Communication Port On TM4C

TM4C123GH6P Micro-controllers Programming Concepts

Dr. Munesh Singh

BOTTOM VIEW



UART Enable Steps

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

UART Enable Step

Table 23-5. GPIO Pins and Alternate Functions

IO	Pin	Analog Function	Digital Function (GPIOPCTL PMCx Bit Field Encoding)[a]										
			1	2	3	4	5	6	7	8	9	14	15
PA0	17	-	U0Rx	-	-	-	-	-	-	CAN1Rx	-	-	-
PA1	18	-	U0Tx	-	-	-	-	-	-	CAN1Tx	-	-	-
PA2	19	-	-	SSI0Clk	-	-	-	-	-	-	-	-	-
PA3	20	-	-	SSI0Fss	-	-	-	-	-	-	-	-	-
PA4	21	-	-	SSI0Rx	-	-	-	-	-	-	-	-	-
PA5	22	-	-	SSI0Tx	-	-	-	-	-	-	-	-	-

GPIO Port Control (GPIOPCTL)

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 0x52C
Type RW, reset -

PA7 PA6 PA5 PA4

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	PMC7				PMC6				PMC5				PMC4			
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PMC3				PMC2				PMC1				PMC0			
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

PA3 PA2 PA1 PA0

UART Enable Steps

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

UART Enable Steps

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

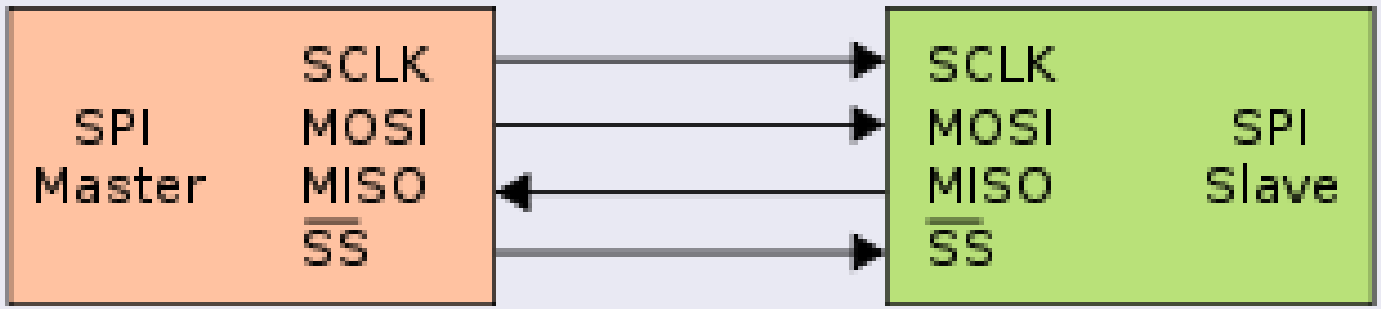
UART Communication To Light The LED

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

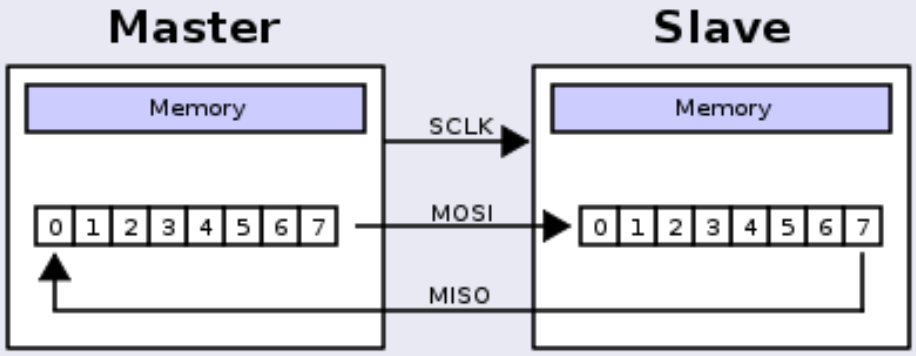
1. The Serial Peripheral Interface (SPI) is a synchronous serial communication interface
2. The interface was developed by Motorola in the mid 1980s
3. SPI devices communicate in full duplex mode using a master-slave architecture with a single master.
4. Multiple slave devices are supported through selection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



Synchronous Serial Interface

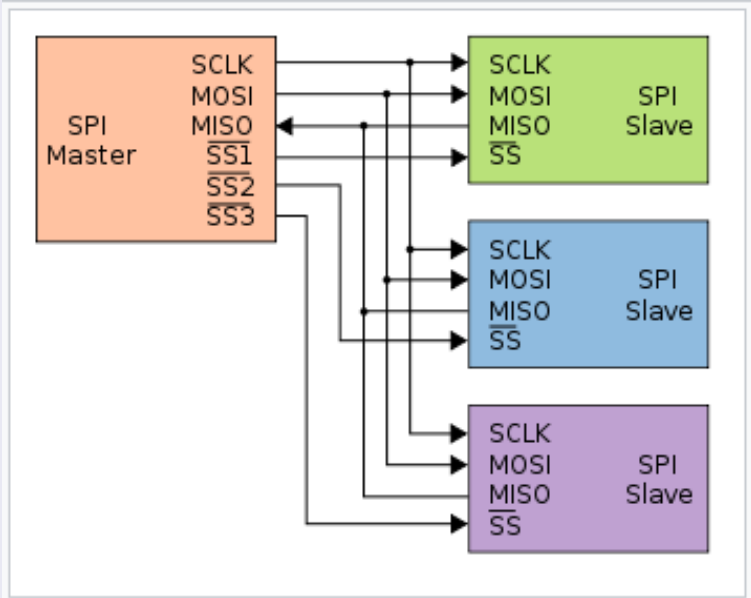
Data Transmission

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

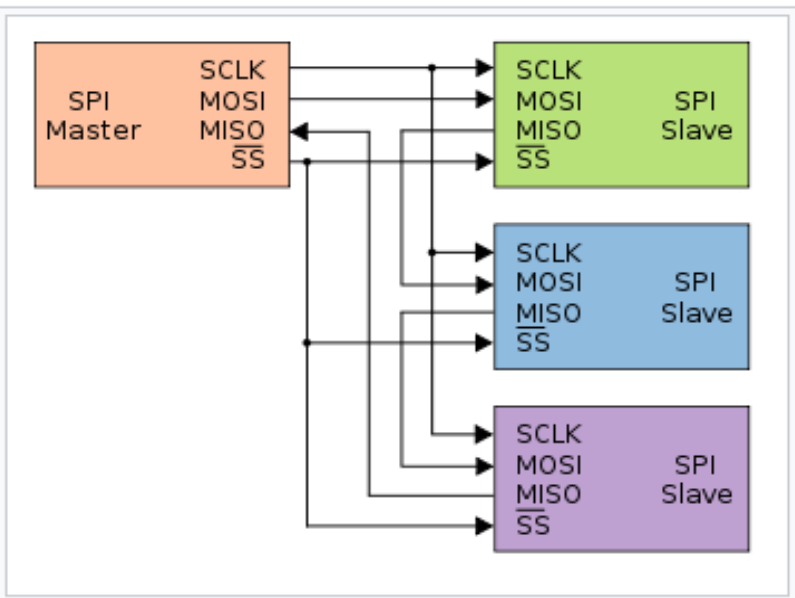


Synchronous Serial Interface

Connecting with Multiple SPI device



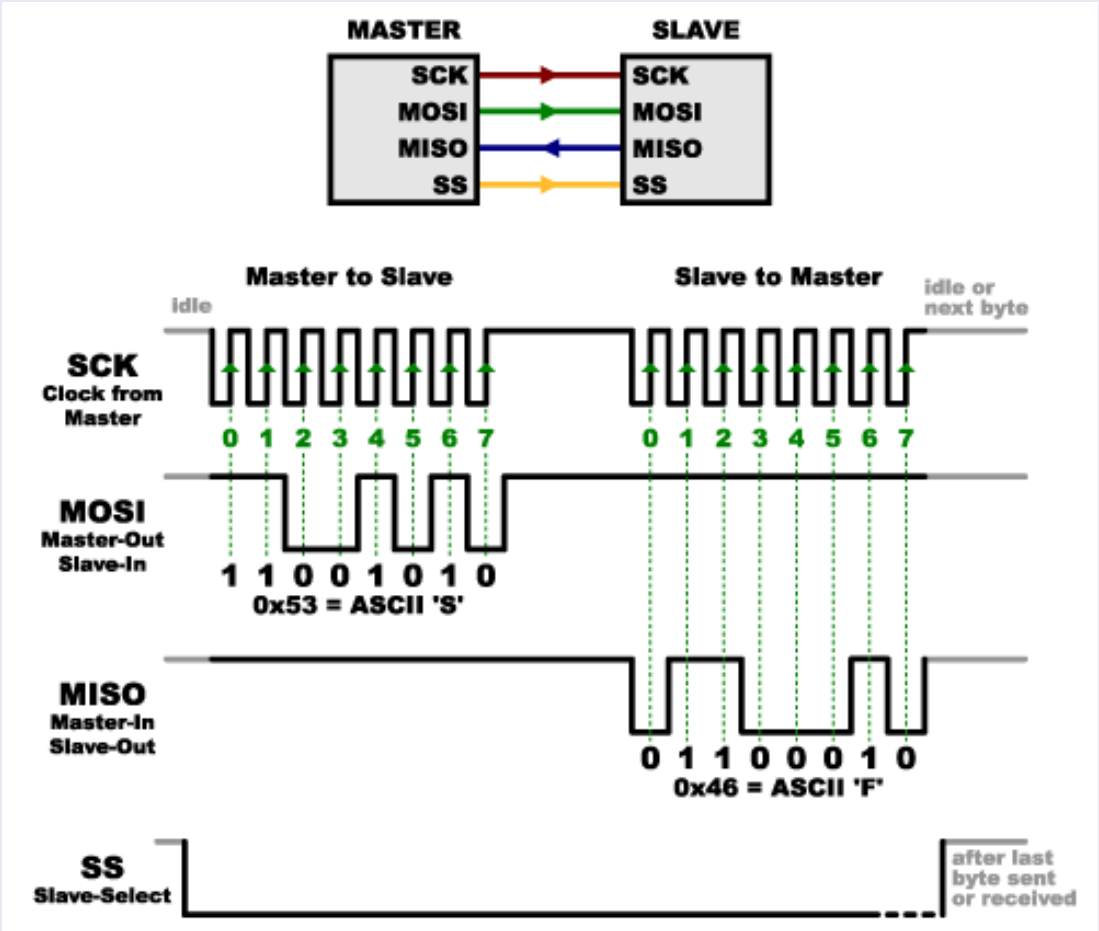
Typical SPI bus: master and three independent slaves



Daisy-chained SPI bus: master and cooperative slaves

Synchronous Serial Interface

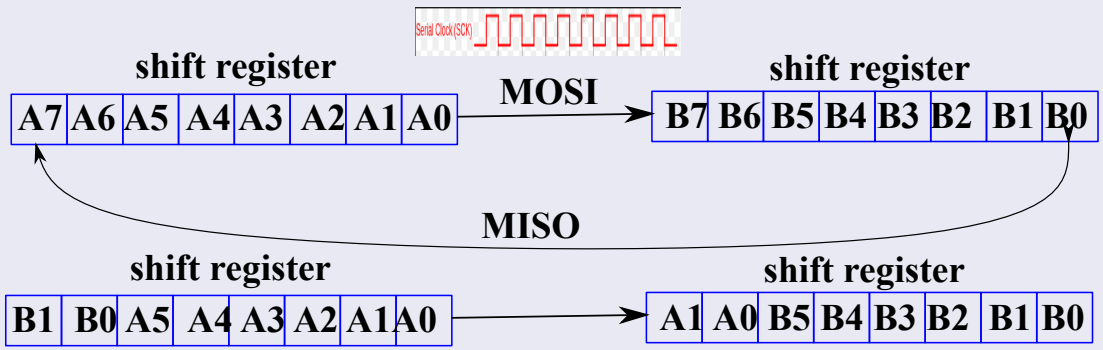
Operation



Synchronous Serial Interface

Configuration

- SPI communication uses simple shift register

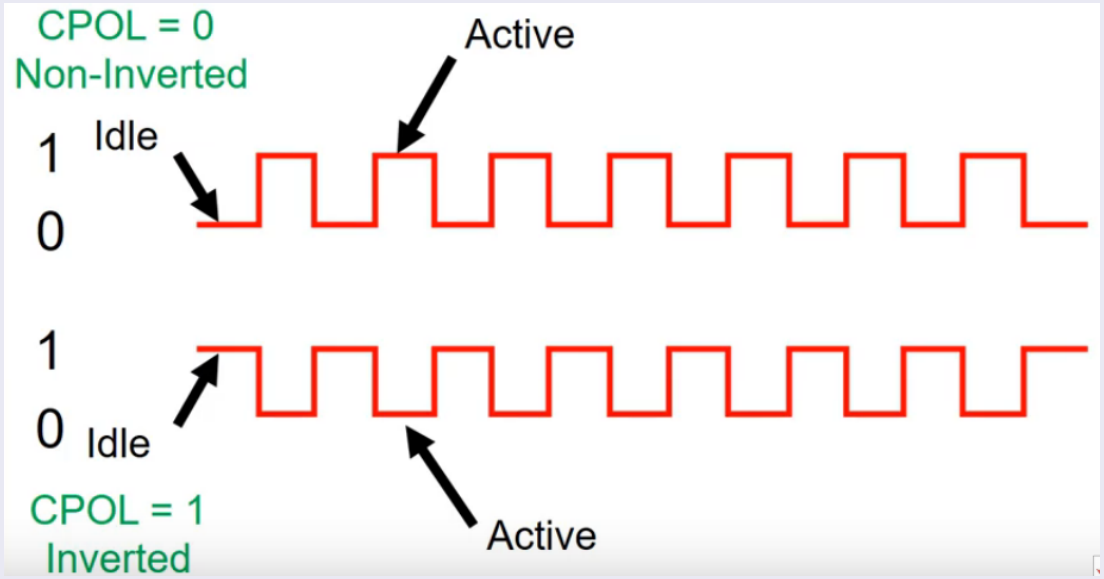


After the Clock 8 data transmission get over

Synchronous Serial Interface

Configuration

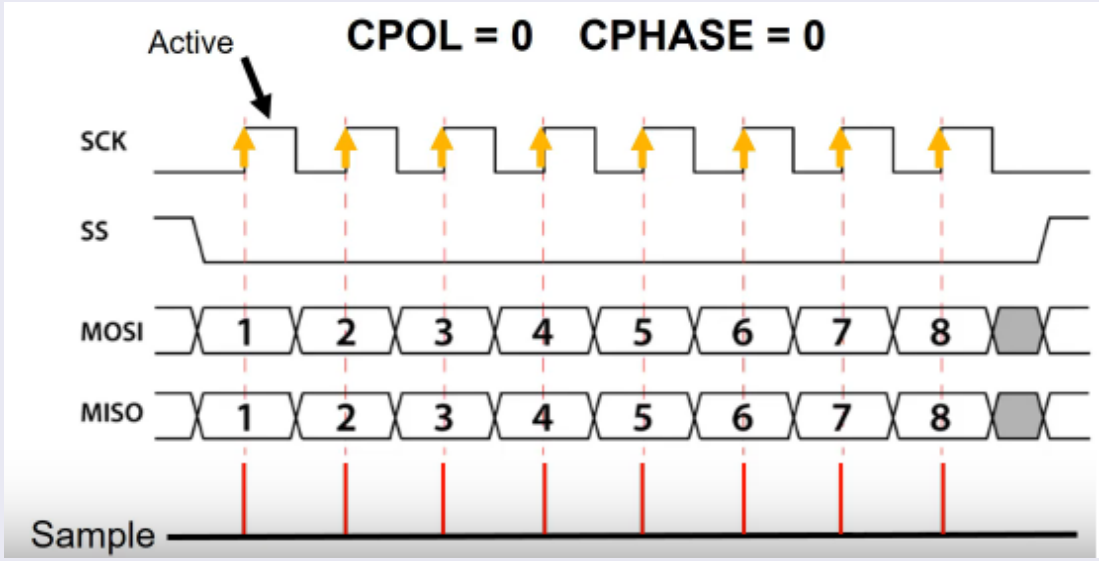
- SPI Mode Configuration required to set two register
  - **Clock Polarity (CPOL)**
  - **Clock Phase(CPHASE)**
    - When the data has to be toggled
    - When the data has to be sampled



Synchronous Serial Interface

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

Modes

Mode	CPOL	CPHASE	Comments
Mode 0	0	0	Active state of of clock is 1 Data sampling at leading edge
Mode 1	0	1	Active state of of clock is 1 Data sampling at trailing edge
Mode 2	1	0	Active state of of clock is 0 Data sampling at leading edge
Mode 3	1	1	Active state of of clock is 0 Data sampling at trailing edge

I2C Communication Port

What is I2C

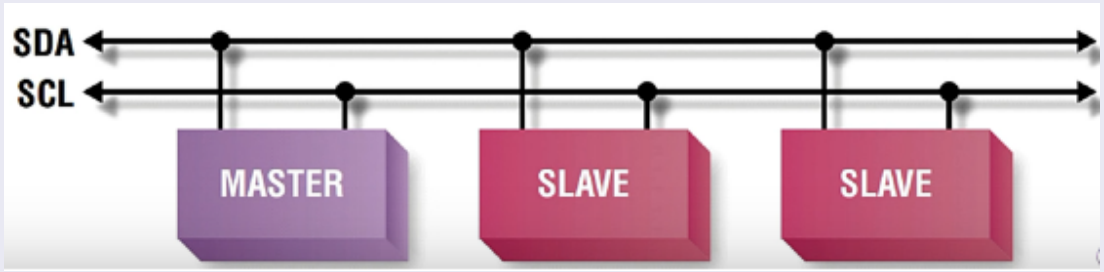
- Inter-integrated Circuit
- Bidirectional Data Transfer
- Half Duplex (have only one data-line)
- Synchronous bus so data is clocked with clock signal
- Clock is controlled when data line is changed

Speed of I2C

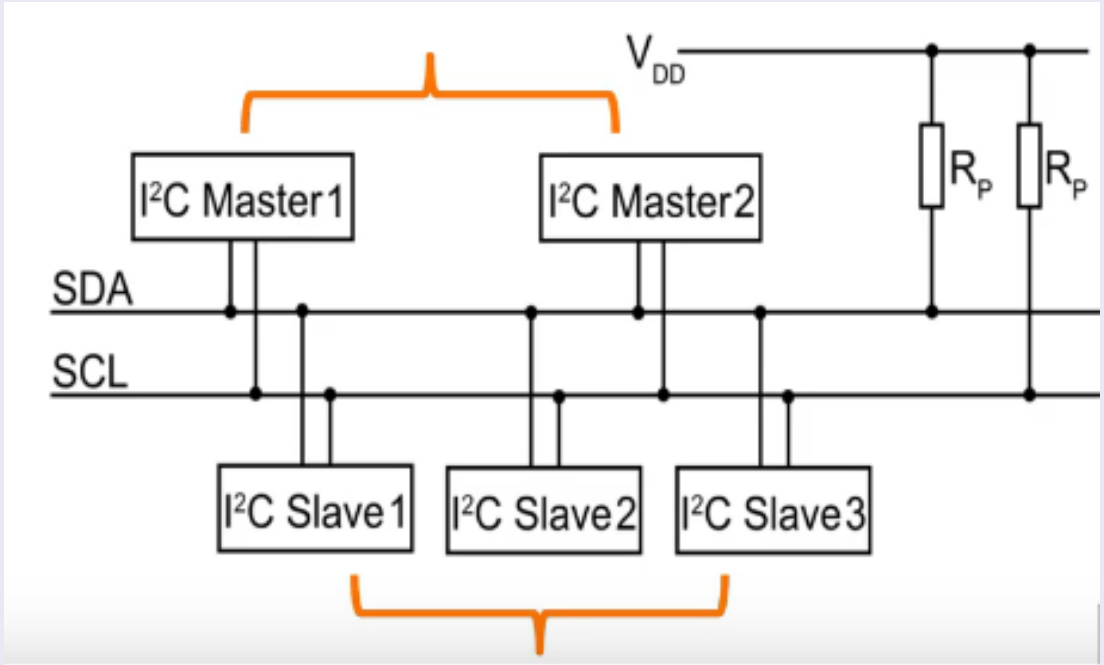
- Low (under 100 kbps)
- Fast (400 Kbps)
- High speed (3.4 mbps) (I2C V2.0)
- 2 Wire communication:
- SDA and SCL

I2C Communication Port

What is I2C

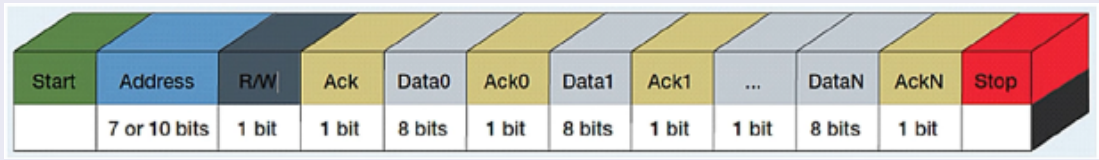


Basic protocol is master slave protocol



I2C Communication Port

Packet Formate



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

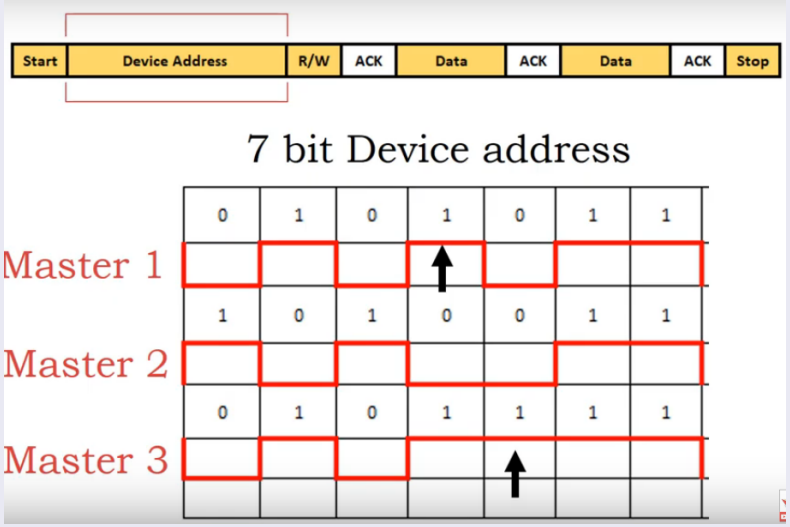
I2C Concepts

Arbitration

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

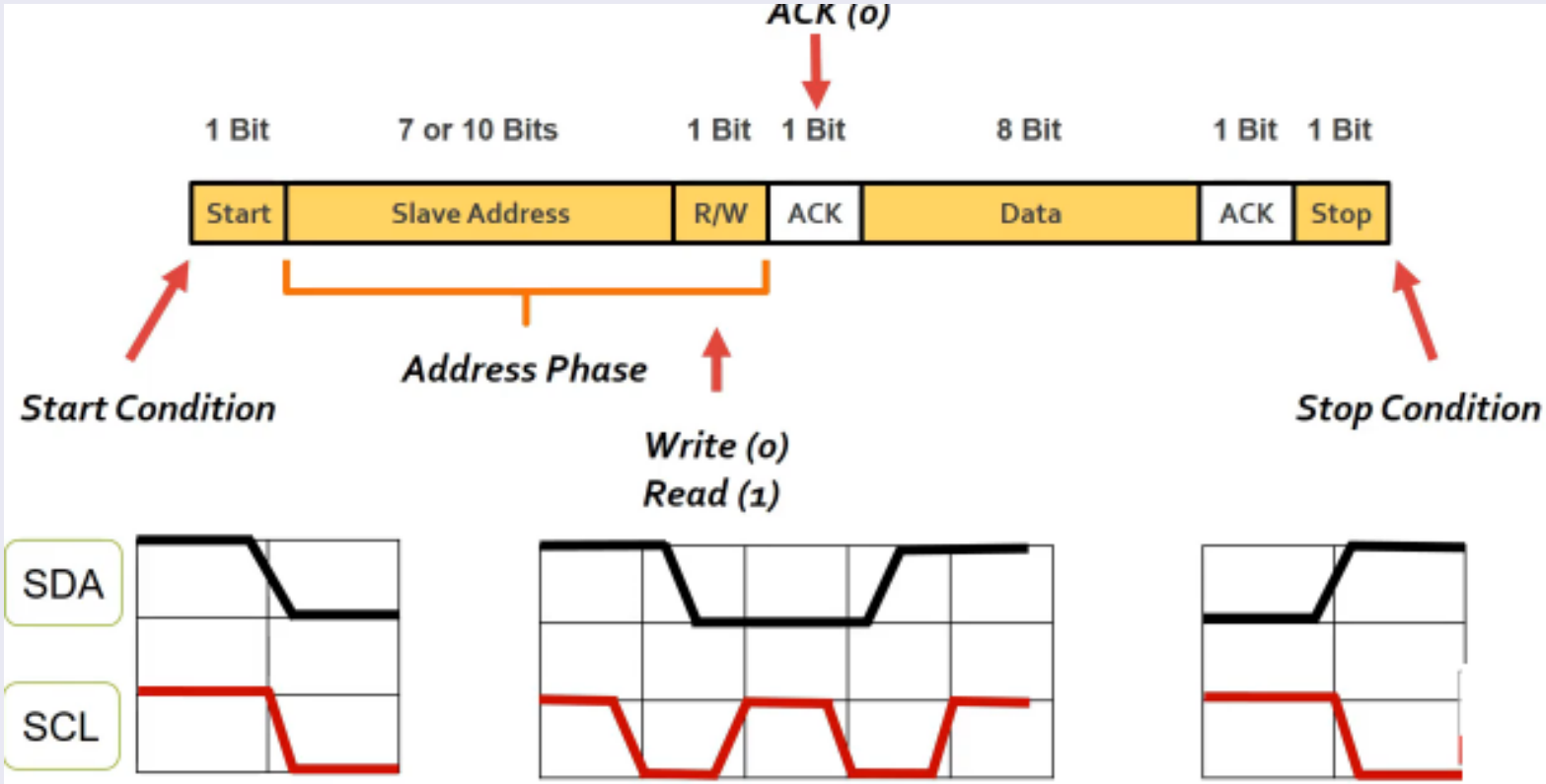
Clock Streaching

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



I2C Driver Development

Case 1 "Master Want to Write 1 Byte"



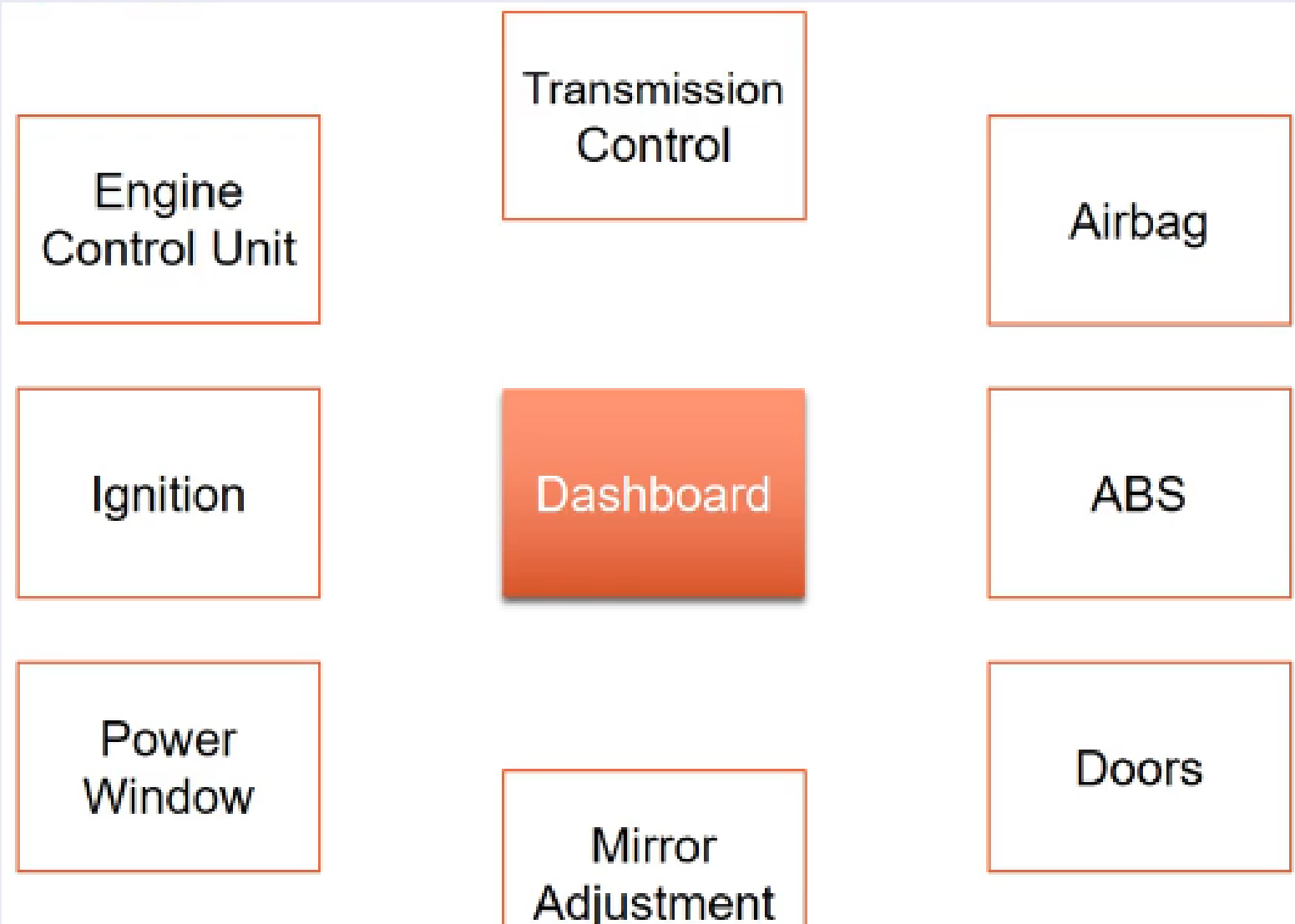
Control Area Networks

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

CAN Protocol



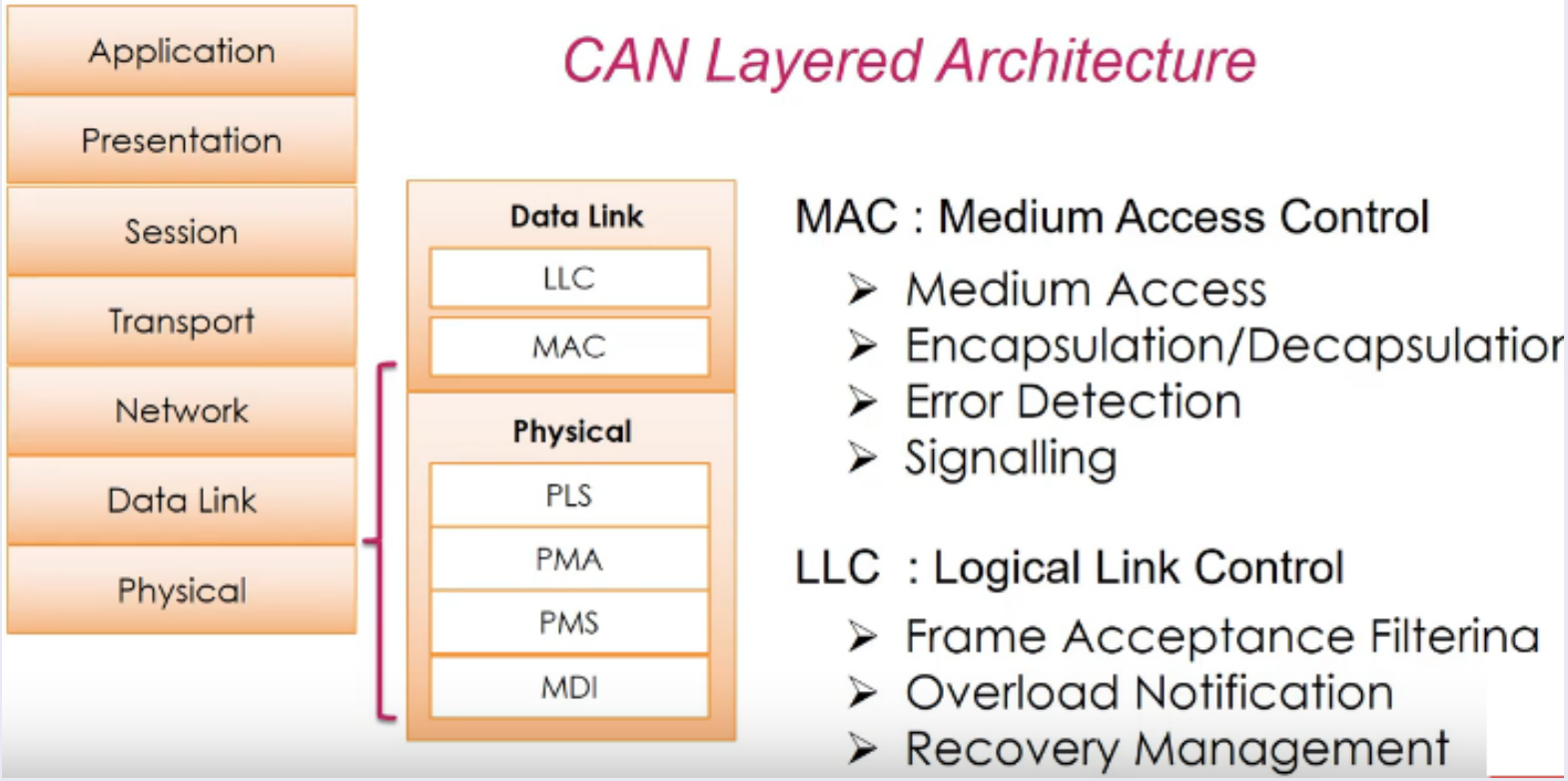
Application of CAN Protocol

CAN Protocol

- Automotive(passenger vechicles, 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

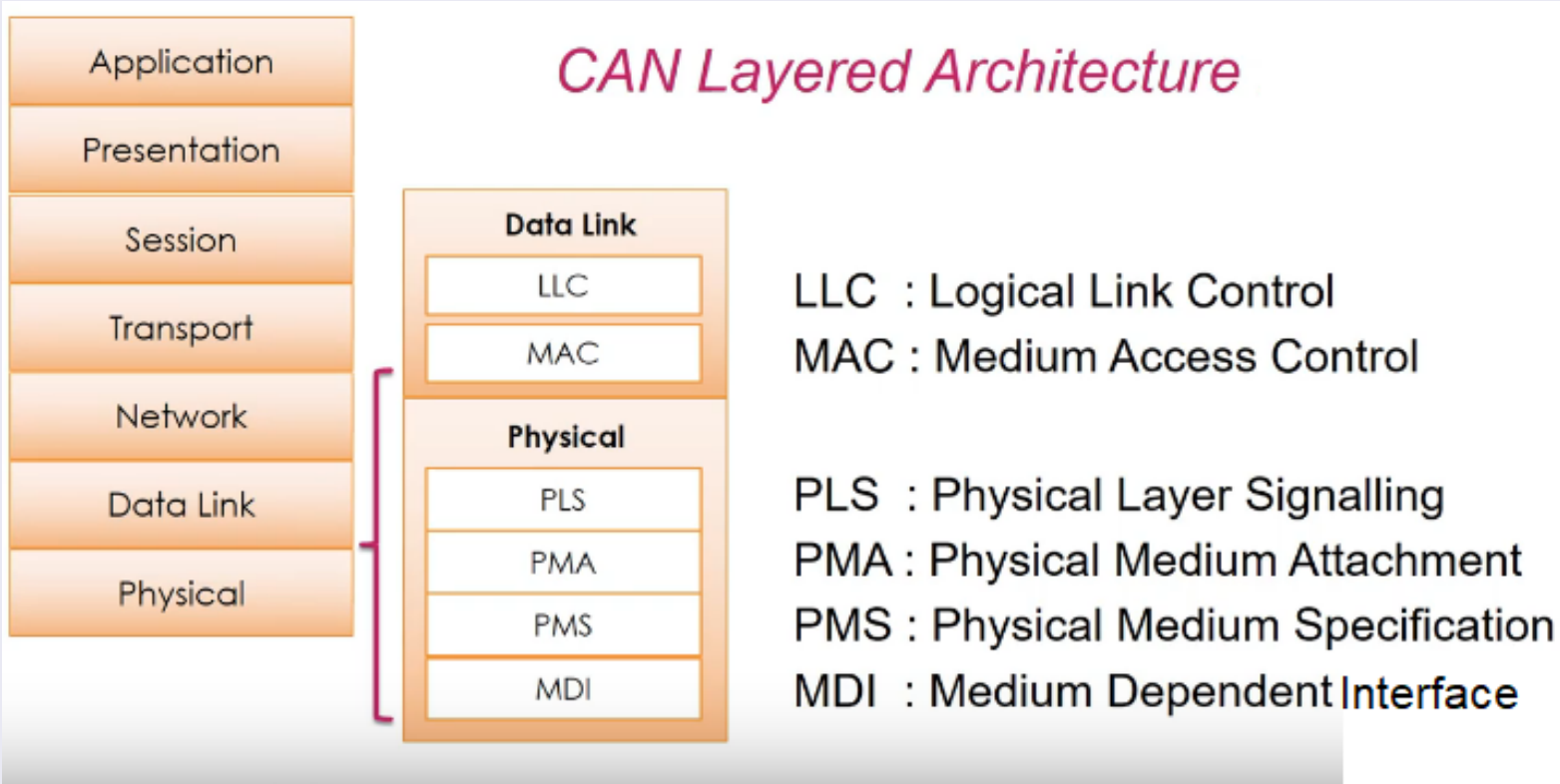
CAN Architecture

CAN



CAN Architecture

CAN



CAN Architecture

CAN

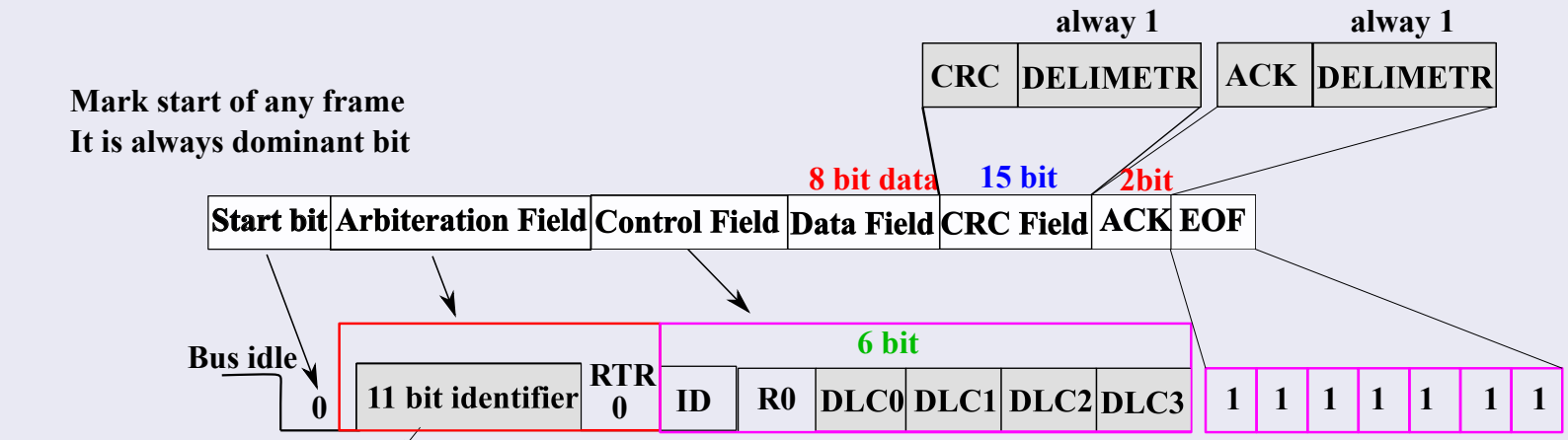
- Datalink Layer (CAN Protocol)
- Physical Layer (CAN Physical Layer)
- CAN Datalink Layer has standard (ISO 11898-1)
- CAN Physical Layer has tree standard
  - ISO 11898-2 (signalling) (datarate-1 Mbps)
  - ISO 11898-3 (medium) (datarate-125kb)
  - CIA (CAN in Automation) DS-102 (Types of connectors)
- CAN controller and CAN tranciver are implemented in sotware 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

- 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

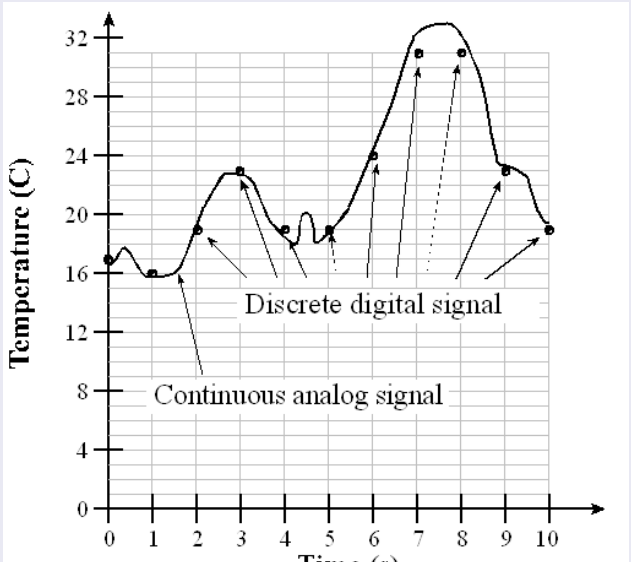
CAN Frame Formate

CAN



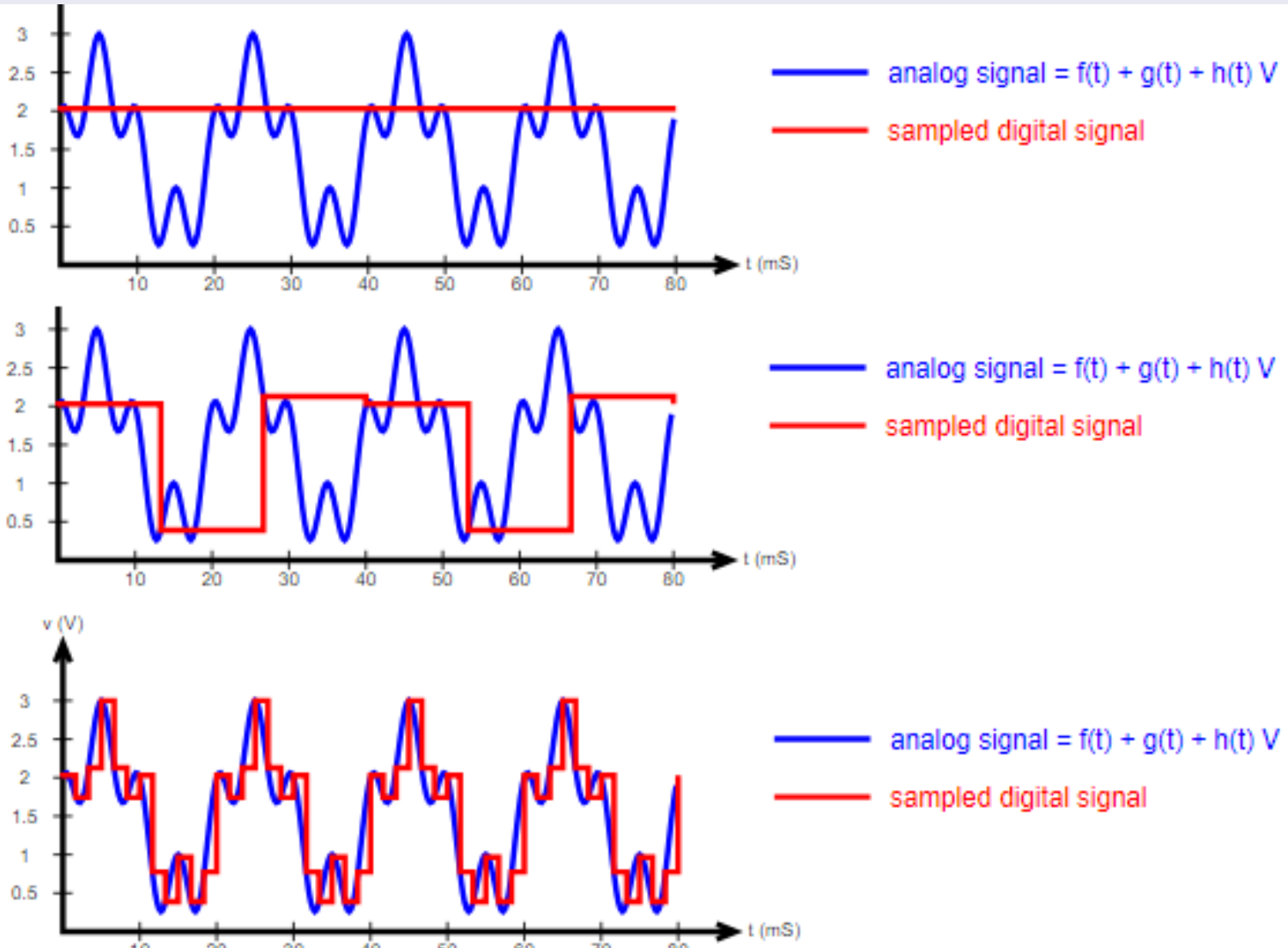
Digital to Analog Conversion

- 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 fs, then to correctly digitalizes the signal we need to sample the signal twice of sampling frequency



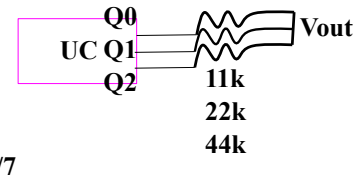
Nyquist Theorem

Sampling frequency



Digital to Analog Converter

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

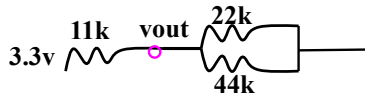


3 bit DAC

Resolution= 3.3/7

N Levels	Volt Out
0	0
1	3.3*1/7
2	3.3*2/7
3	3.3*3/7
4	3.3*4/7
5	3.3*5/7
6	3.3*6/7
7	3.3*7/7

N Levels	Q2	Q1	Q0
0	0	0	0
1	0	0	3.3
2	0	3.3	0
3	0	3.3	3.3



Vout=(44/3/11+44/3)*3.3