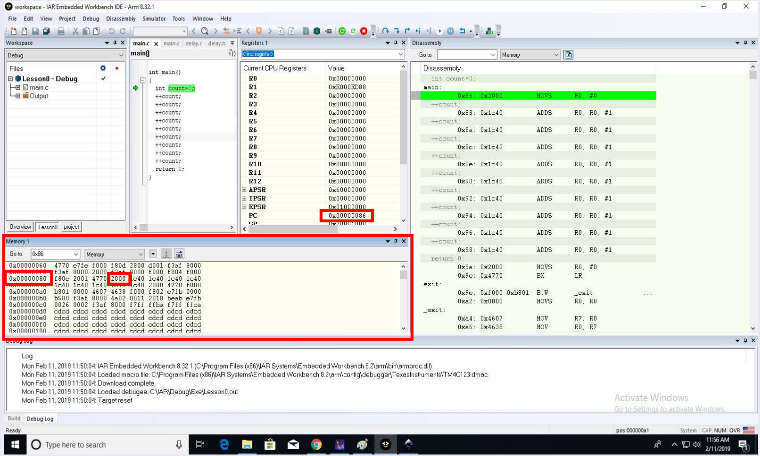
TM4C123GH6PM Micro-controllers Programming Concepts

Dr. Munesh Singh

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

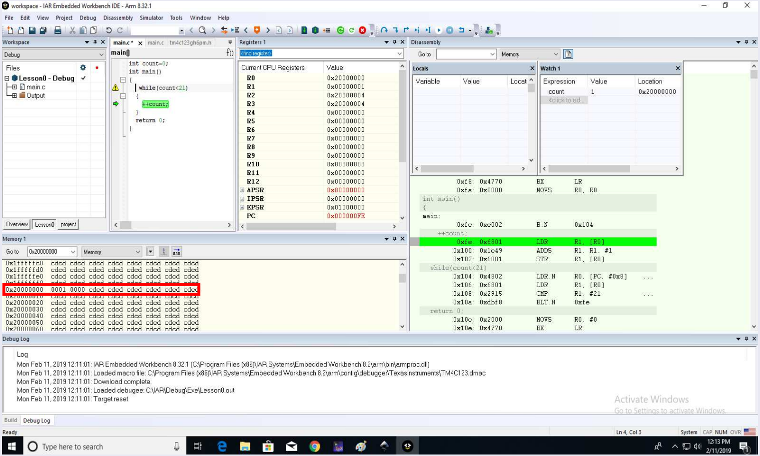
January 31, 2020

ARM Microcontroller

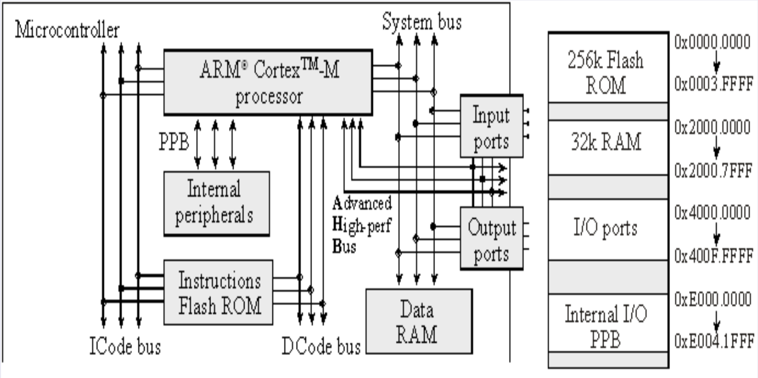


IAR Workbench Toolset

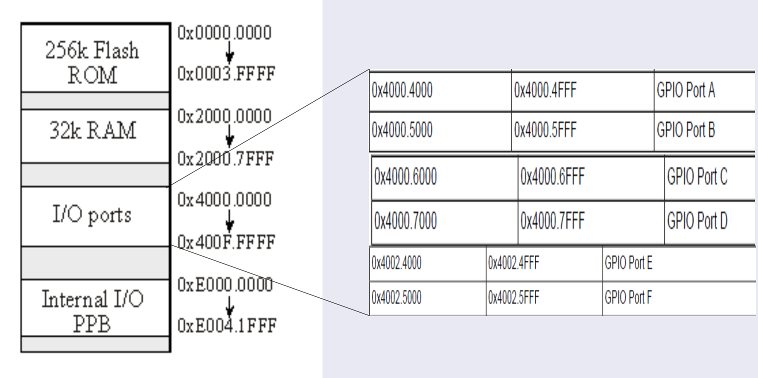
Local versus Global Variable



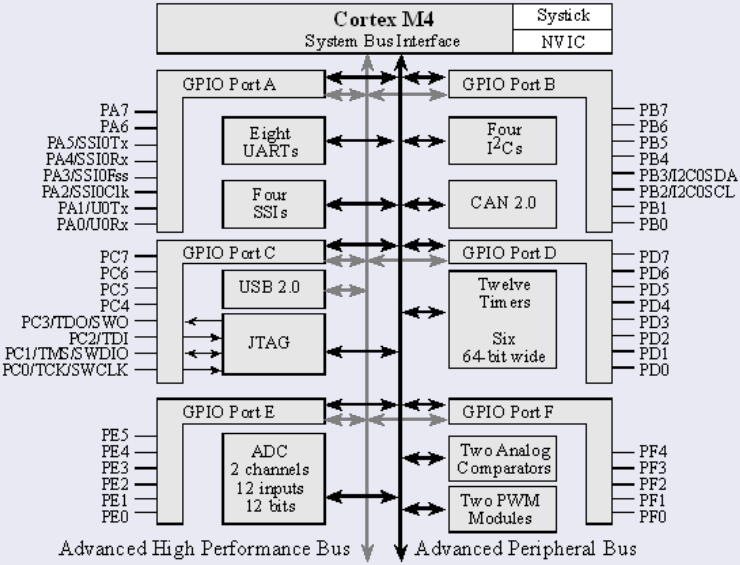
Memory Map Buses



Memory Map I/O Ports



Classification of I/O Ports



I/O Port Initialization

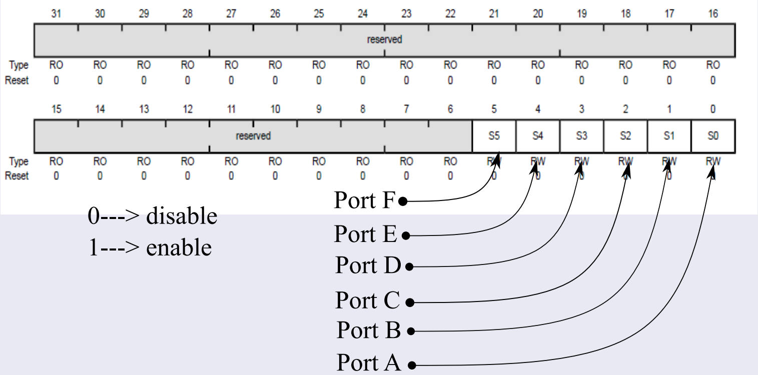
Seven Steps to Initialize a Port

- 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

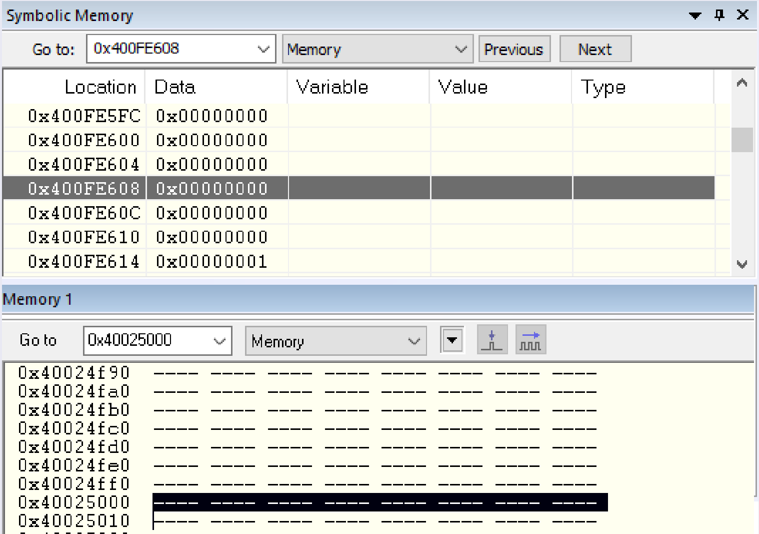


0---> disable

1---> enable

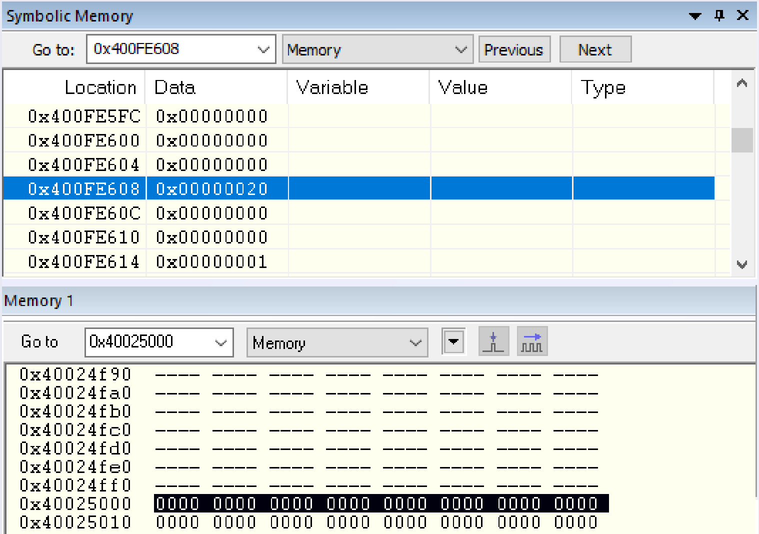
Debugger Mode Memory View SCGCGPIO Register and F Port

F Port Before Setting bit 5 of SCGCPIO register



Dibugger Mode Memory View SCGCGPIO Register and F Port

F Port after Setting bit 5 of SCGCPIO register



GPIODIR Direction Register

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	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								DIR							
Type	RO	RO	RO	RO	RO	RO	RO	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

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

	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								DIR							
Type	RO	RO	RO	RO	RO	RO	RO	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

0---> Disable Digital function

1---> Enable Digital function

0:7 Bits

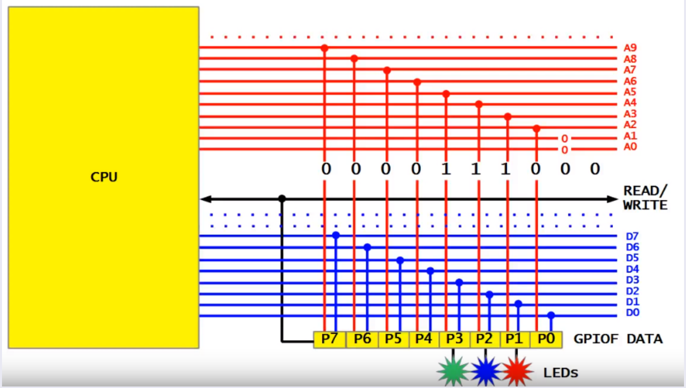
GPIO Digital Enable (GPIODEN)

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

	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								DIR							
Type	RO	RO	RO	RO	RO	RO	RO	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

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

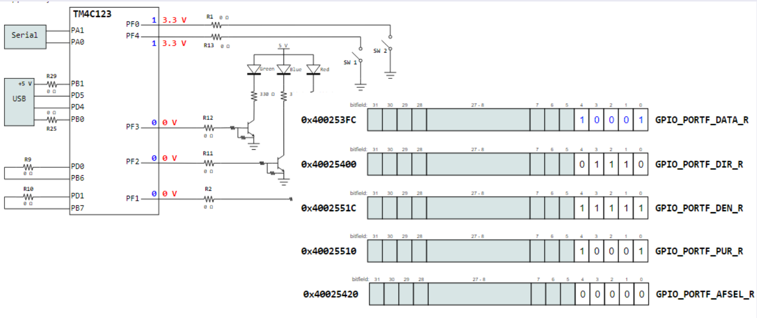
3FC

0x400253FC

GPIO Data (GPIODATA)

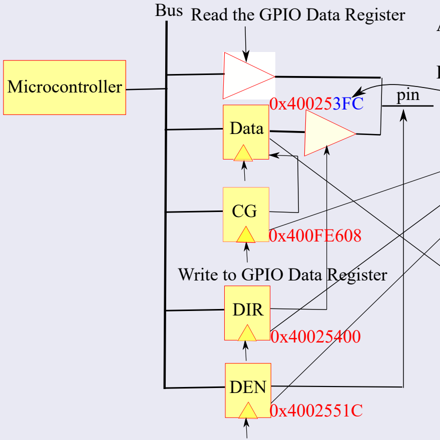
GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB)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



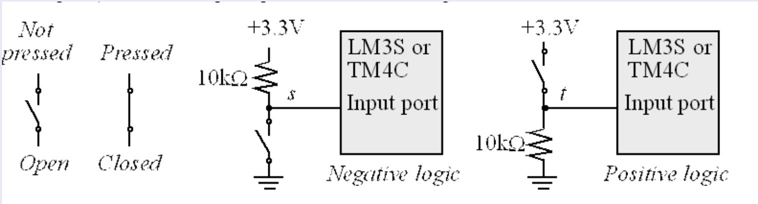
Address 9 8 7 6 5 4 3 2 1 0
2 1 8 4 2 1 8 4 2 1
DataF 7 6 5 4 3 2 1 0
1 1 1 1 1 1 1 1

```
int main()
{
*((unsigned int *)0x400FE608)=0x20U;
*((unsigned int *)0x40025400)=0x0EU;
*((unsigned int *)0x4002551C)=0x0EU;
while(1)
{
*((unsigned int *)0x400253FC)=0x02U;
*((unsigned int *)0x400253FC)=0x00U;
}
}
```

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

	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								DIR							
Type	RO	RO	RO	RO	RO	RO	RO	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

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.

	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								DIR							
Type	RO	RO	RO	RO	RO	RO	RO	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

GPIO Commit (GPIOCR)

0:7 Bits

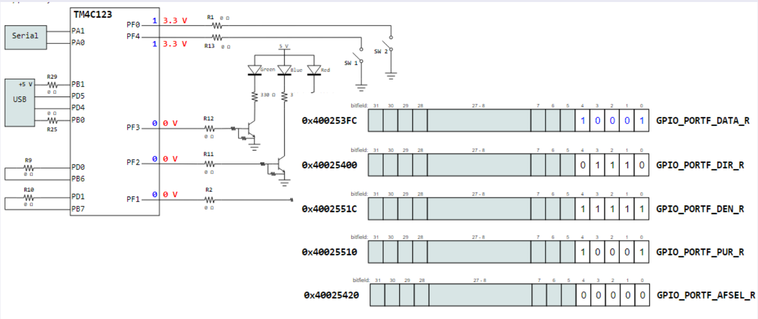
Default value is 0xFF

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 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

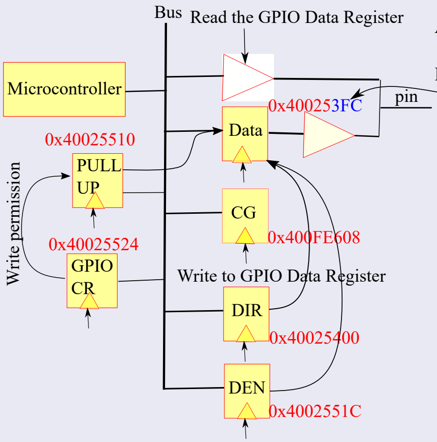


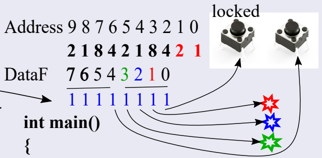
Blue LED Control using Switch at Pin 4

TM4C123GH6P Micro-controllers Programming Concepts

Dr. Munesh Singh

Simple Program






```
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[2]C0	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

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	LOCK															
Type	RW	RW	RW	RW	RW	RW	RW	RW	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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	LOCK															
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	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

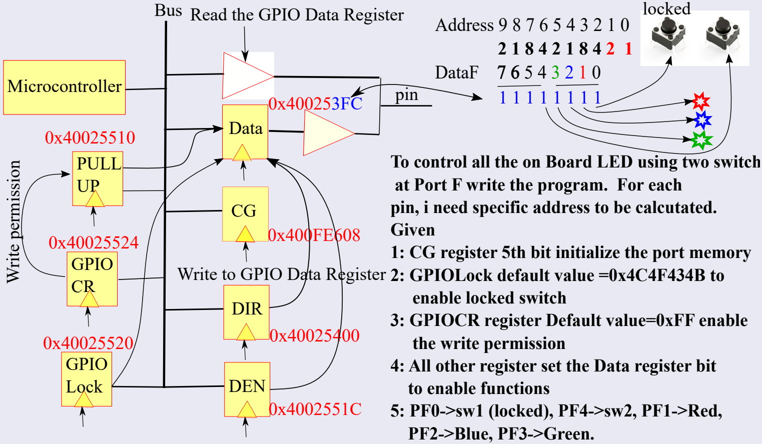
Default value=0x4C4F434B

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

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

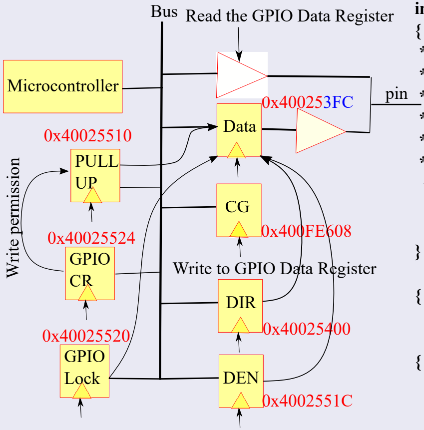
3: GPIOCR register Default value=0xFF enable the write permission

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$\wedge$ 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

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

```
#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;
```

```
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]
```

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]
```