Digital to Analog World

TM4C123GH Tiva C Board

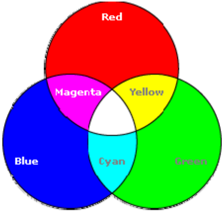
Dr. Munesh Singh

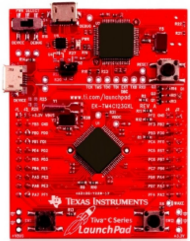
Indian Institute of Information Technology Design and Manufacturing
Kancheepuram, Chennai, Tamil Nadu

February 3, 2020

Next LAB Execise

PF4	PF3	PF2	PF1	PF0	←	Pin Numbers		
SW1	G	B	R	SW2		Code Data		Function
0	0	0	1	0	→	02	0x02	RED
0	0	1	0	0	→	04	0x04	BLUE
0	1	0	0	0	→	08	0x08	GREEN
0	0	1	1	0	→	06	0x06	MAGENTA
0	1	1	0	0	→	0C	0x0C	CYAN
0	1	0	1	0	→	0A	0x0A	YELLOW
0	1	1	1	0	→	0E	0x0E	WHITE
				1	→			SWITCH2
1					→			SWITCH1
0	0	0	0	0	→	00	0x00	OFF
PA7 – PA0					→	8 Pins		
PB7 – PB0					→	8 Pins		
PC7 – PC0					→	8 Pins		
PD7 – PD0					→	8 Pins		
PE5 – PE0					→	6 Pins		
PF4 – PF0					→	5 Pins		
Total GPIO Pins					→	43 Pins		

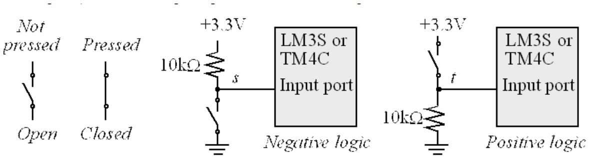




TOTAL MCU PINS --> 64 Pins

Steps need to be configure

- All the steps followed in previous program remain same.
- In-order to use on-board user switch two registers need to set:
  1. Pull Up Register or Pull Down Register
     - Pull-up and Pull-down resistors are used to correctly bias the inputs of digital gates.
     - It stop them from floating about randomly when there is no input condition
     - By default all GPIO pins has internal Pull-Up or Pull-Down register
  2. The GPIOCR register- it commit the operational setting of Pull-Up or Pull-Down.



PULLUP 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
												SW2				SW1
	reserved								8	4	2	1 DIR	8	4	2	1
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 PULLUP	0	0	0	0 PULLUP

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

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

Default value is 0xFF

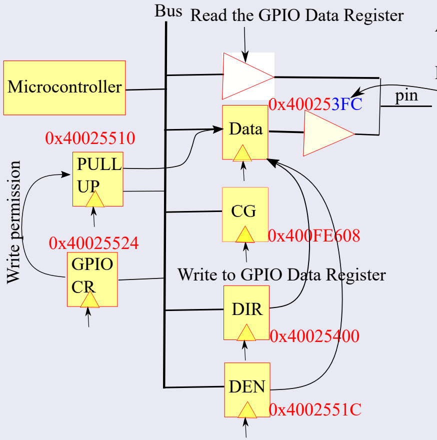
0x40025524=0xFF

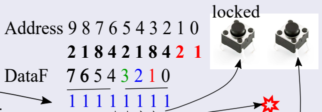
0	The corresponding **GPIOAFSEL**, **GPIOPUR**, **GPIOPDR**, or **GPIODEN** bits cannot be written.
1	The corresponding **GPIOAFSEL**, **GPIOPUR**, **GPIOPDR**, or **GPIODEN** bits can be written.

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 -

Simple Program






```c
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^{21}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

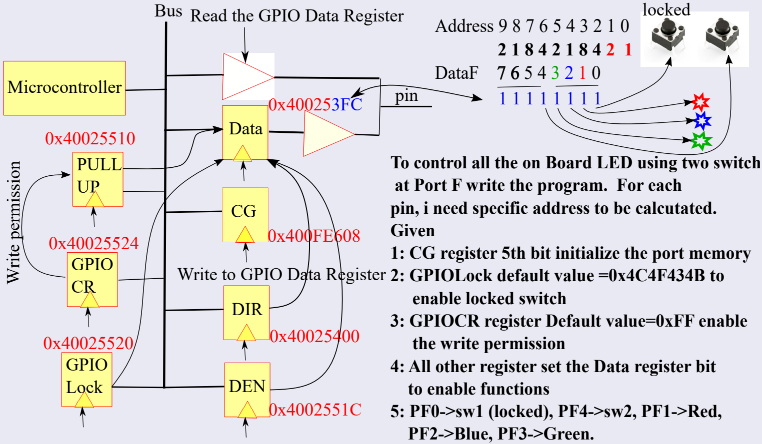
Default value=0x4C4F434B

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 User 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

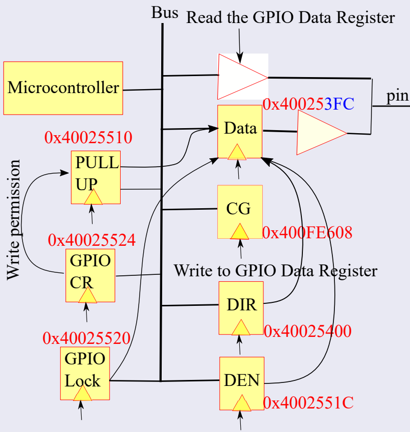
3: GPIOCR register Default value=0xFF enable the write permission

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 RGB LED Using On-Board User Switches

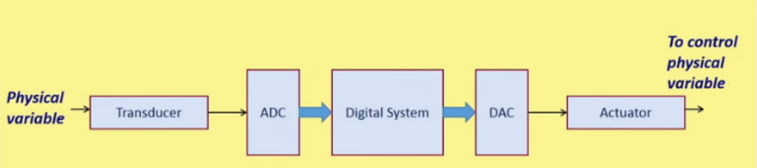
Write a Program




```
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;}
  }
  return 0;
}
```

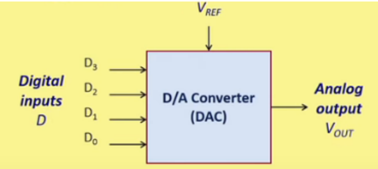
Interfacing with the Analog World

1. Transducer/Sensor: convert physical variable to electrical variable
2. Analog-to-digital converter (ADC)
3. Digital System (micro-controller)
4. Digital-to-analog converter (DAC)
5. Actuator



Digital-to-Analog Converter (DAC)

- Converts a given digital word D to a proportional analog voltage V_{out}
  $V_{OUT} \propto D$

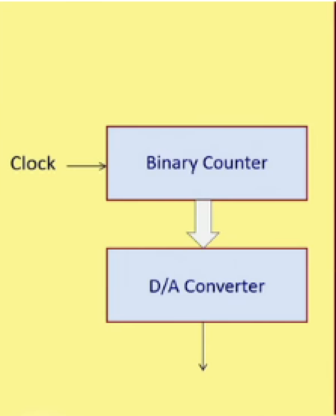


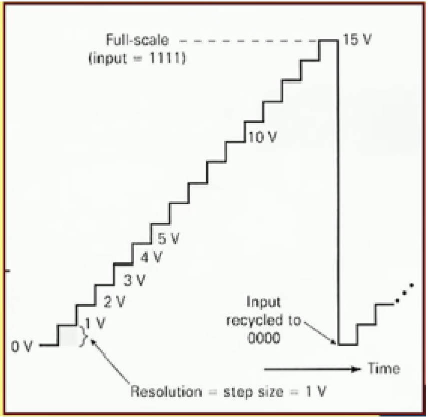
$V_{REF} = 15\ V$

D_3	D_2	D_1	D_0	V_{OUT}
0	0	0	0	0 V
0	0	0	1	1 V
0	0	1	0	2 V
0	0	1	1	3 V
0	1	0	0	4 V
0	1	0	1	5V
0	1	1	0	6 V
0	1	1	1	7 V
1	0	0	0	8 V
1	0	0	1	9 V
1	0	1	0	10 V
1	0	1	1	11 V
1	1	0	0	12 V
1	1	0	1	13 V
1	1	1	0	14 V
1	1	1	1	15 V

Resolution or Step Size

- Smallest change that can occur in V_{OUT} as a result of a change in input D.
  - Equal to the weight of the LSB, also called step size.
  - Same as the constant of proportionality in $V_{OUT} \propto D$
- Can also be defined as a percentage of the full-scale voltage:
  Step-size or resolution $\Delta = \frac{V_{REF}}{(2^N - 1)}$ for an N-bit DAC
  Percentage resolution $= \frac{\Delta}{(full-scale \;\; votage)) \times 100\%}$
  $= 1/(2^N - 1) \times 100$


Clock
Binary Counter
D/A Converter

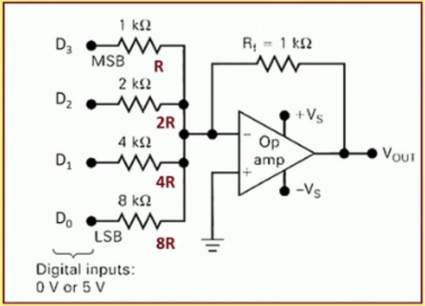

Full-scale
(input = 1111)
15 V
10 V
5 V
4 V
3 V
2 V
1 V
0 V
Input
recycled to
0000
Time
Resolution = step size = 1 V

Types of D/A Converter

- We shall discuss two different designs of digital-to-analog converters.
  - Weighted resistor type DAC
  - Resistive ladder type DAC
- The first type is easier to analyze, while the second type is more practical from the point of view of implementation.

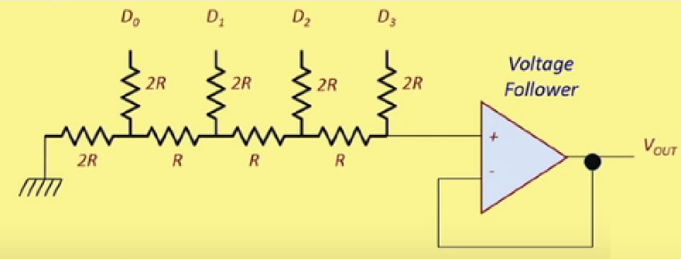
Weighted Resistor Type DAC

- For an n-bit DAC, it consists of n different resistance values of magnitudes R, 2R, 4R, ..., $2^{n-1}R$ respectively.
  - The resistances help in generating currents inversely proportional to their magnitudes.
  - The total current is added up by an operational amplifier, and is converted to the voltage outptu V_{OUT}.
- Main drawback:
  - A different-valued precision resistor must be used for each bit position of the digital input.



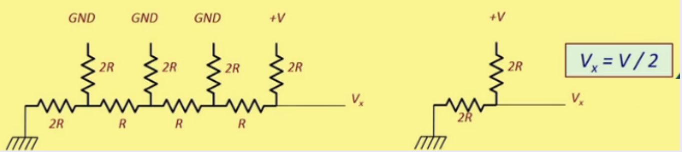
$$V_{OUT} = -[D_3 + D_2/2 + D_1/4 + D_0/8]\,R_f/R$$
$$= -[8D_3 + 4D_2 + 2D_1 + D_0]\,R_f/8R$$
$$\alpha\, D$$

- Most widely used, and requires only two different values of precision resistance (R and 2R)



Calculation

- Let us calculate the voltages at the op-amp input when exactly one of the D_i inputs is at 1 (say, $+$V volts).
- Case 1: input is 1000.

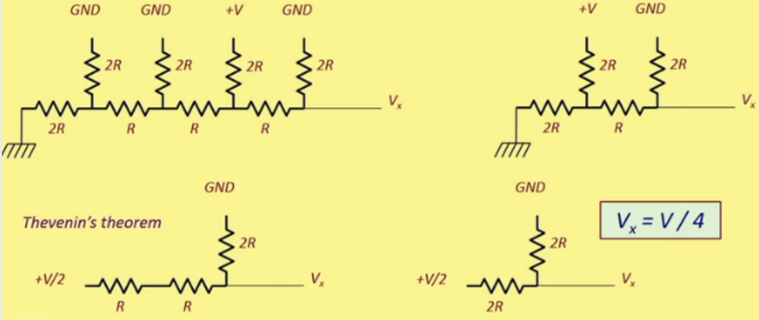


$$V_x = \frac{V * 2R}{2R+2R}$$

Input is 0100

Input is 0100

- *Case 2: Input is 0100.*

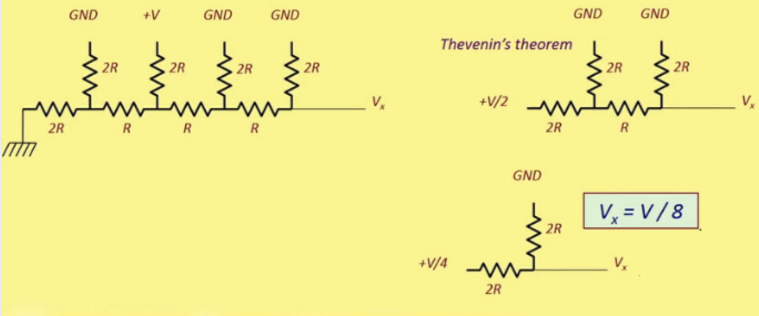


$$V_x = V/4$$

Input is 0010

Input is 0010

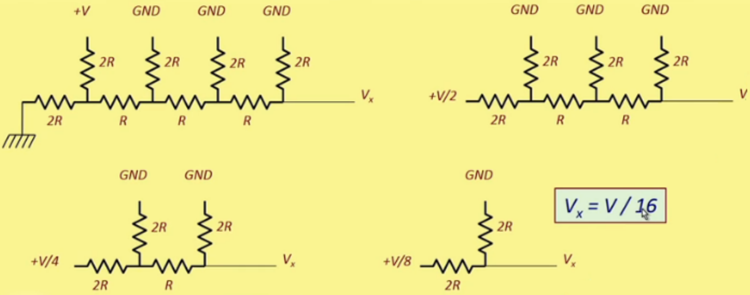
- *Case 3: Input is 0010.*



Input is 0001

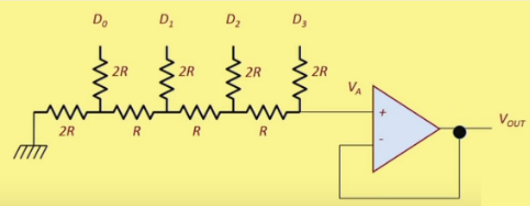
Input is 0001

- *Case 4: Input is 0001.*



$$V_x = V / 16$$

- When all the four inputs D=$D_3 D_2 D_1 D_0$ are applied (where $D_i = GND$ or +V volts), we can apply the principle of superposition to compute the final output voltage V_A.
$V_A = [D_3.(V/2) + D_2(V/4) + D_1.(V/8) + D_0(V/16)]$
$= [8D_3 + 4D_2 + 2D_1 + D_0](V/16)$
$= D.(V/16)$
Thus $V_A \propto D$
- For an N-bit DAC, the contribution of the k-th input D_k on the output voltage will be $V/2^{N-k}$

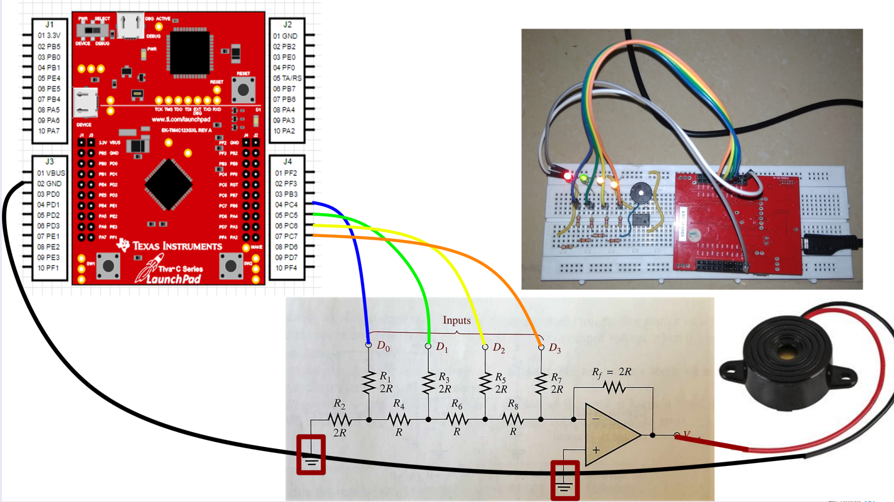


D/A Converter in TM4C123GH

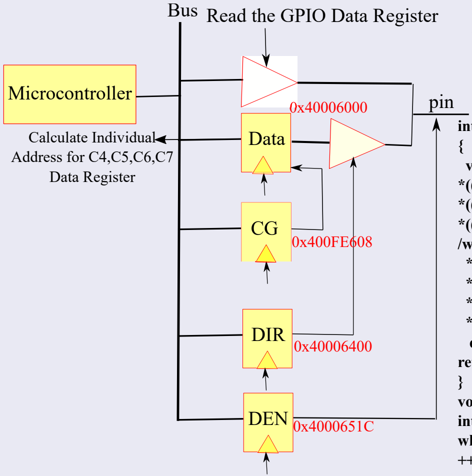
- This development board does not support any DAC channels.
  - We can use the PWM feature for analog control of external devices.
  - Or else we can connect a DAC externally (using resistances or chip).

DAC Using External R-2R Ladder

Connection Diagram



Programming



Address 9 8 7 6 5 4 3 2 1 0

2 1 8 4 2 1 8 4 2 1

PortC 7 6 5 4 3 2 1 0

1 1 1 1 1 1 1 1 C7 C6 C5 C4

```
int main()
{
 void delay();
*((volatile unsigned int *)0x400FE608)=(?);
*((volatile unsigned int *)0x40006400)=(?);
*((volatile unsigned int *)0x4000651C)=(?);
/while(1) {
  *((volatile unsigned int *)C4(?))=(?);
  *((volatile unsigned int *)C5(?))=(?);
  *((volatile unsigned int *)C6(?))=(?);
  *((volatile unsigned int *)C7(?))=(?);
   delay(); }
return 0;
}
void delay() {
int volatile counter=0;
while(counter<100000){
++counter; } }
```

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 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

Thank You