

Introduction to Embedded Systems

CSE 211

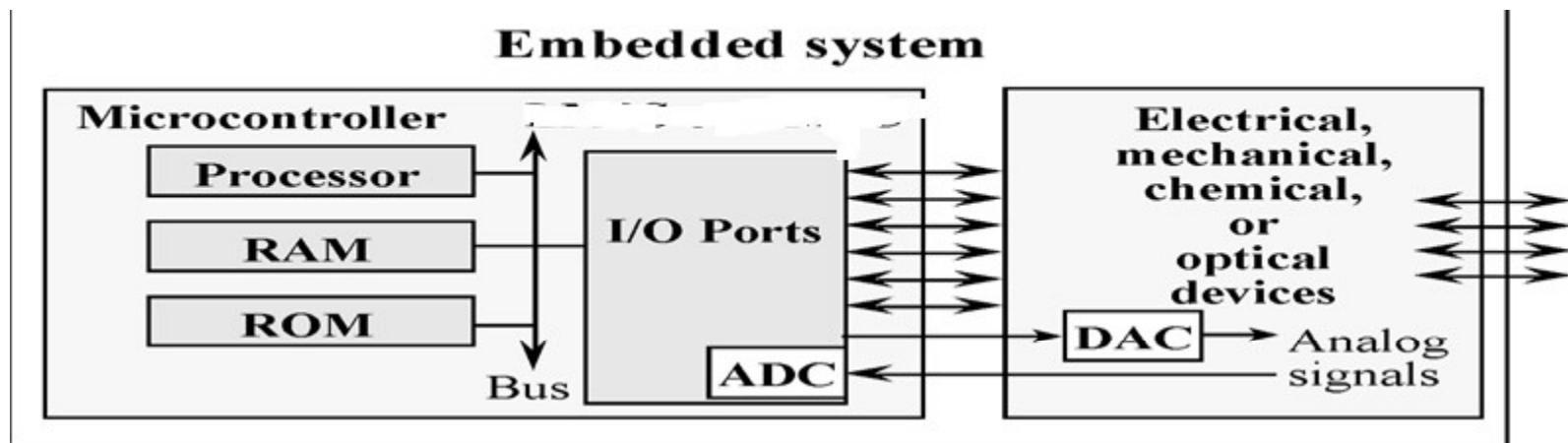
Textbooks – Hardware - Compiler

- Introduction to ARM Cortex- M Microcontroller,
Jonathan Valvano
- Computers as Components, Wayne Wolf
- Hardware
 - Tiva LanuchPad TM4C123
- Compiler
 - Keil – ARM Compiler
- Instructor : Prof. Dr. Ashraf Salem
 - ashraf.salem@eng.asu.edu.eg

Course Contents

- CSE 211
 1. ARM Cortex-M architecture
 2. ARM Cortex-M assembly Language
 3. TM4C123 Microcontroller
 4. Input and output ports
 5. SysTick Timer
 6. Serial and Parallel Interfaces
 7. Interrupt Programming
 8. Analog I/I Interface
 9. Real Time Operating System

Embedded Systems



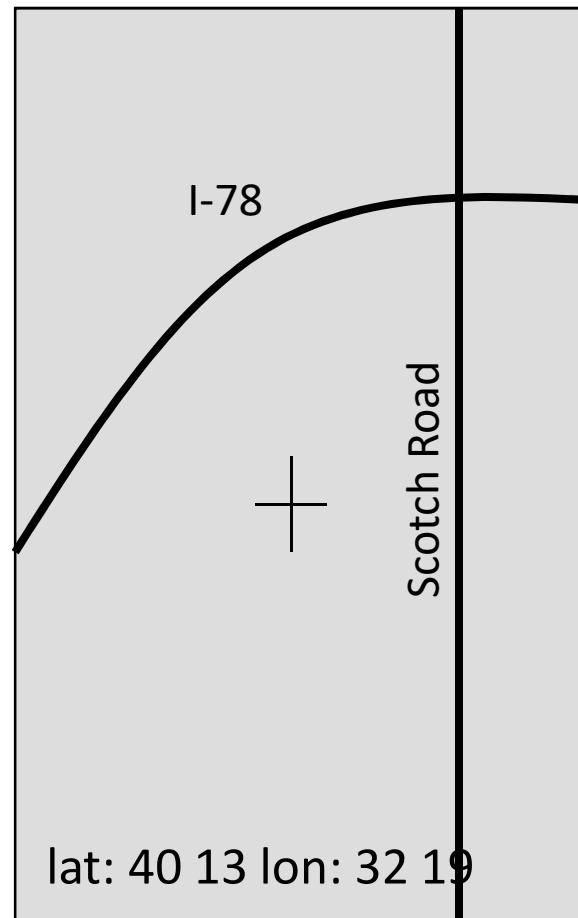
Microcontroller

- ❑ Processor – Instruction Set + memory + accelerators
- ❑ Memory
 - ❑ Non-Volatile
 - o ROM
 - o EPROM, EEPROM, Flash
 - ❑ Volatile
 - o RAM (DRAM, SRAM)
- ❑ Interfaces
 - ❑ H/W: Ports
 - ❑ S/W: Device Driver
 - ❑ Parallel, Serial, Analog, Time

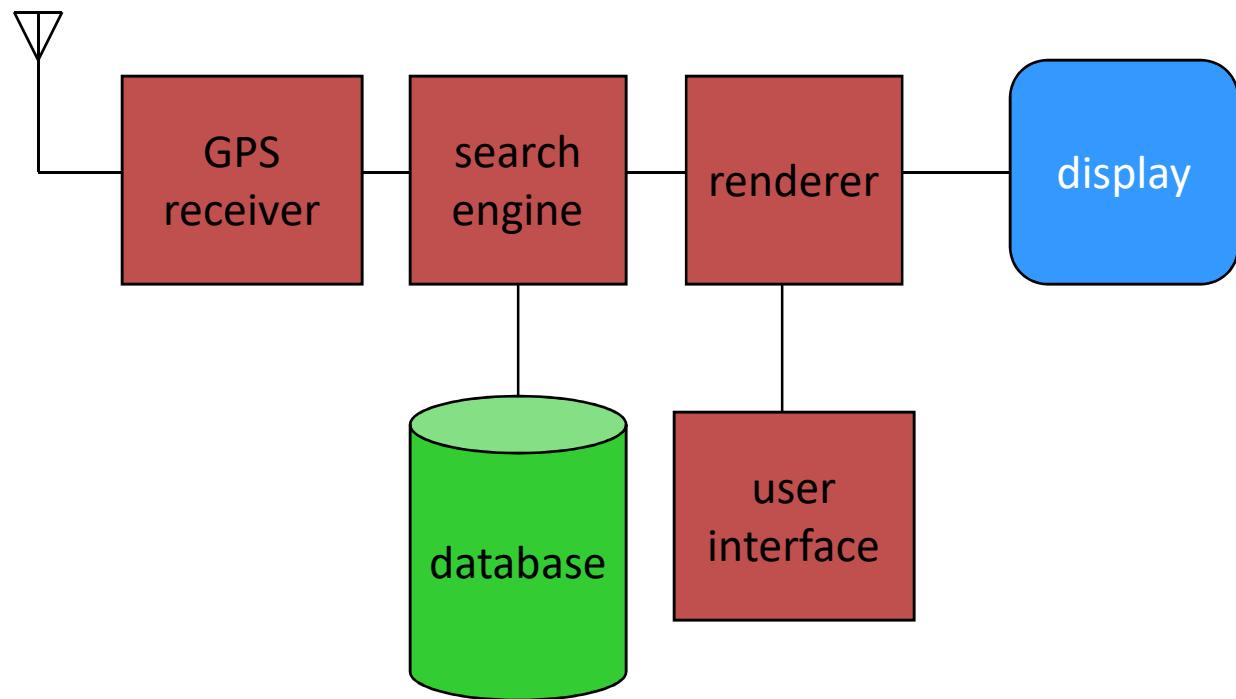
Embedded System Example

GPS

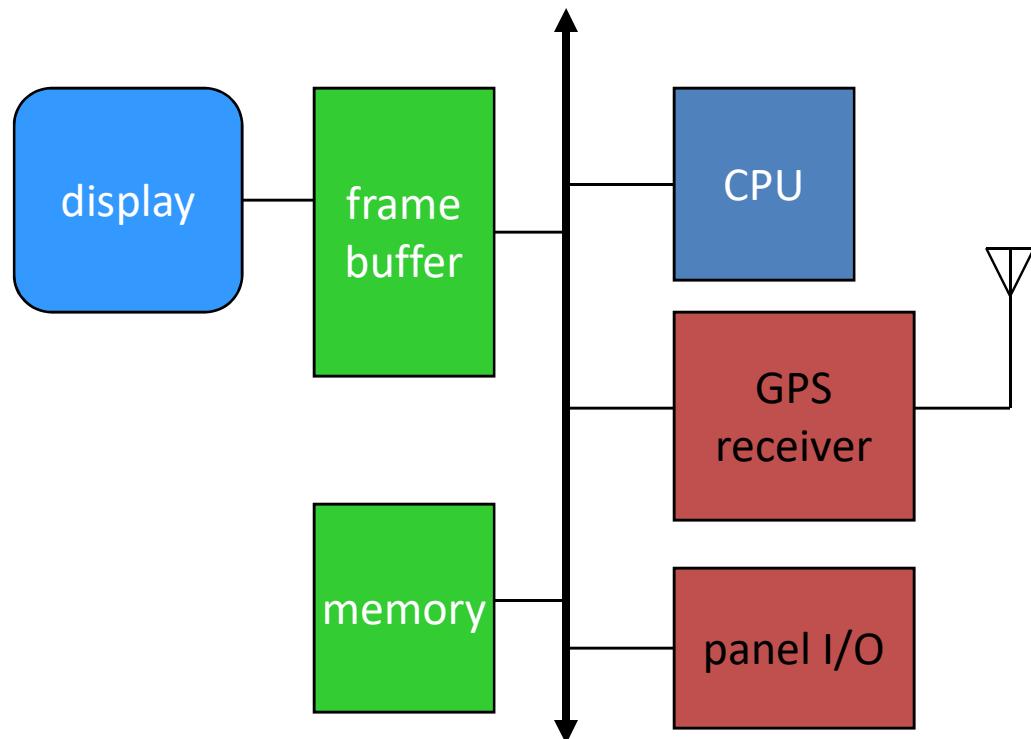
- Moving map obtains position from GPS, paints map from local database.



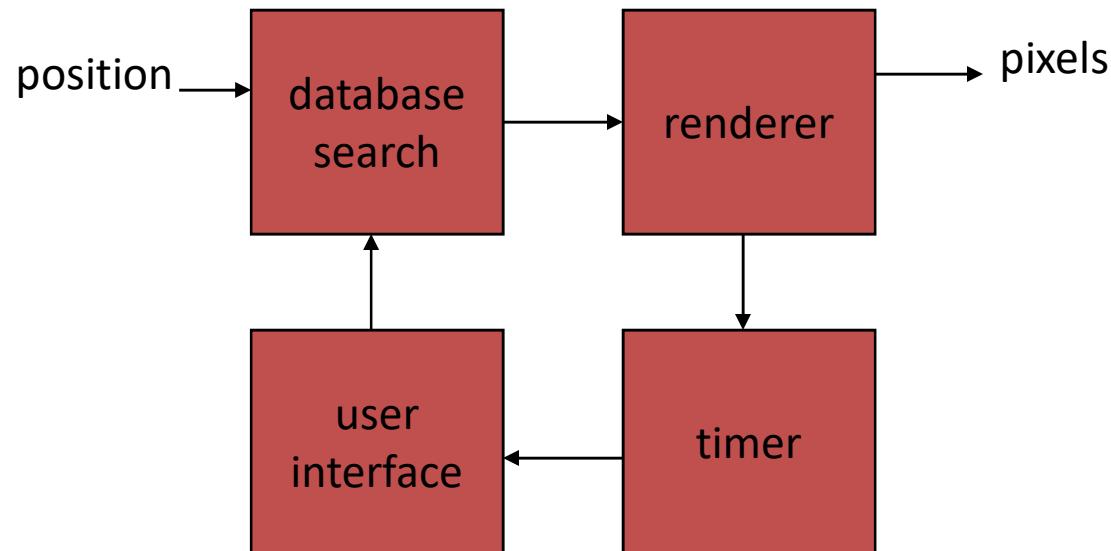
GPS moving map block diagram



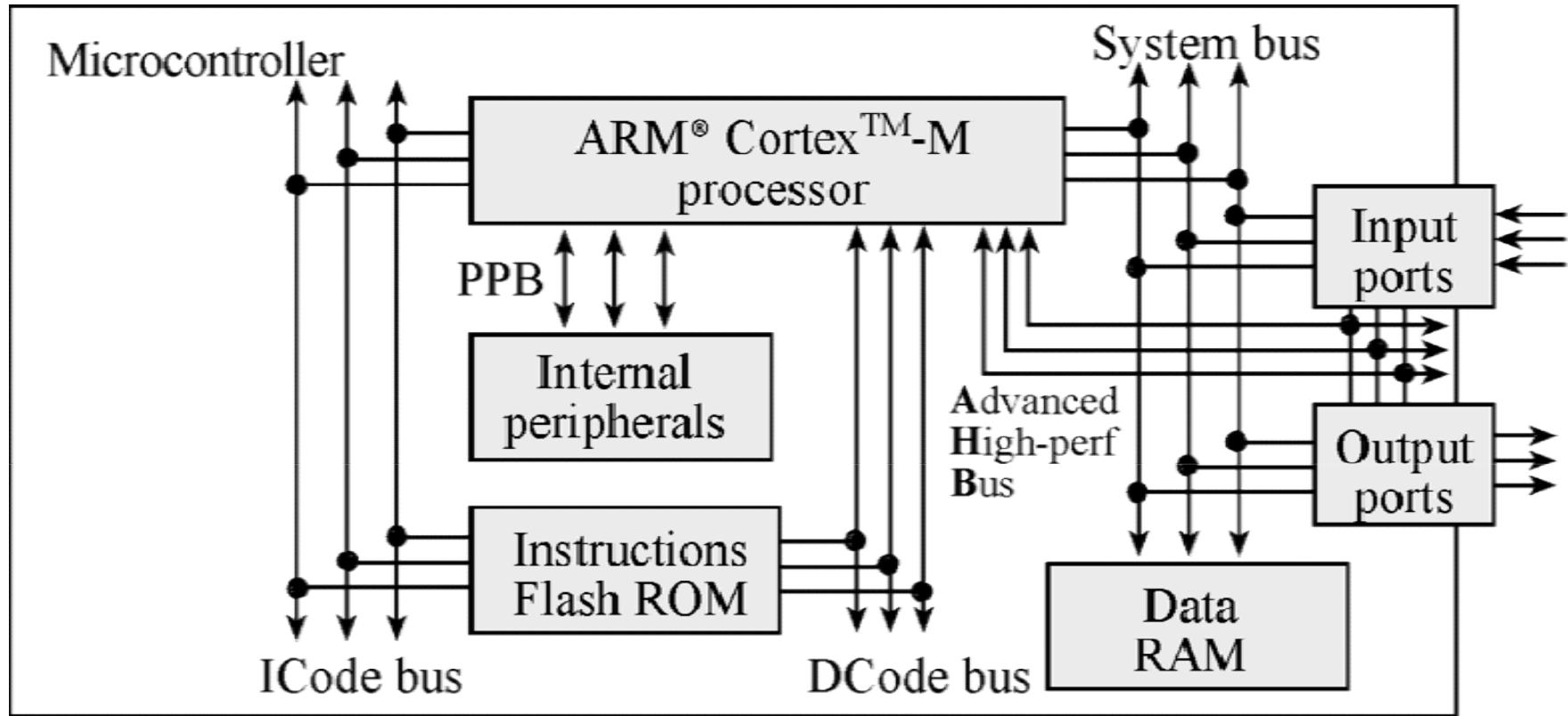
GPS moving map hardware architecture



GPS moving map software architecture

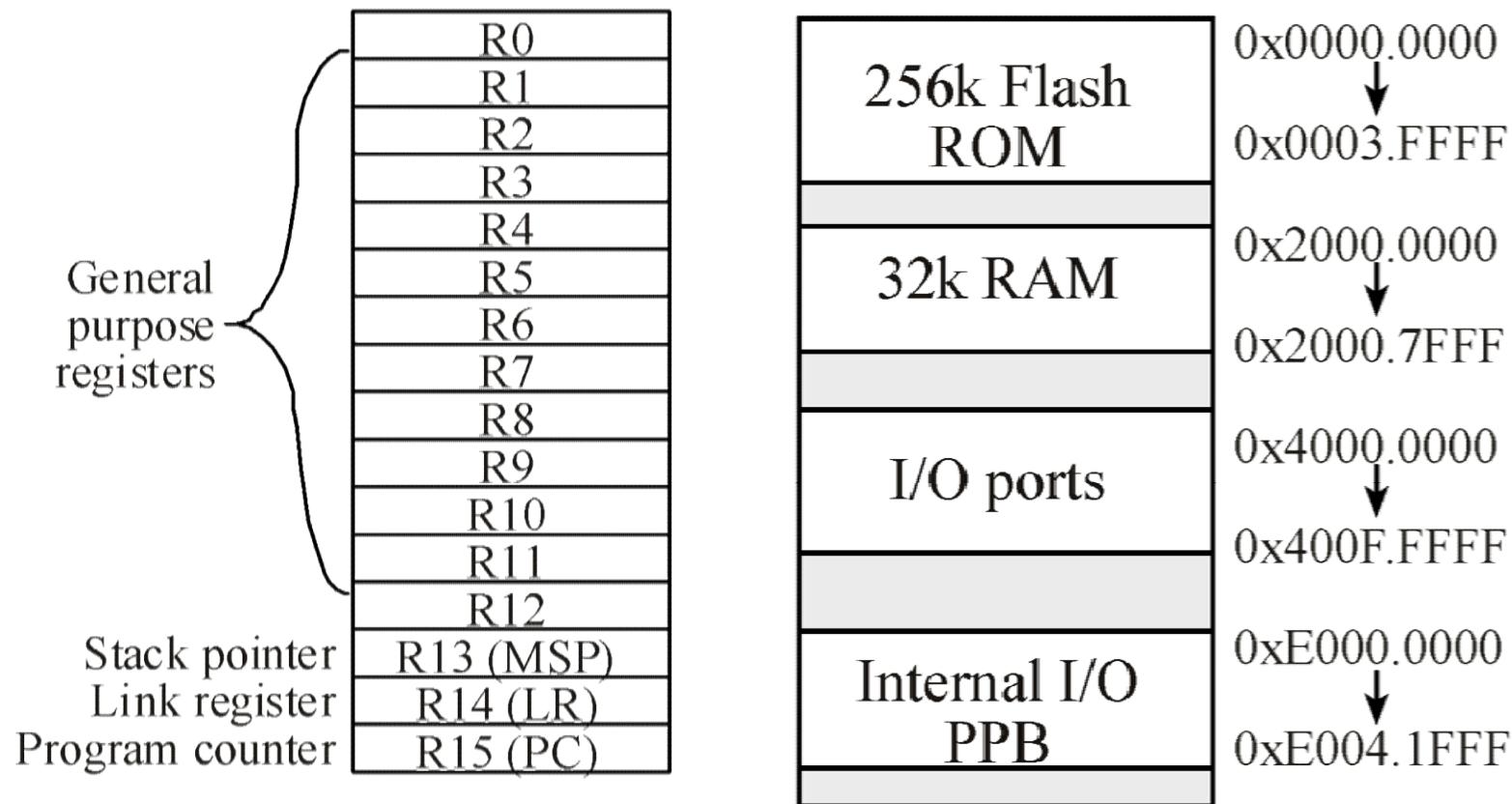


ARM Cortex M4-based System



- ❑ ARM Cortex-M4 processor
- ❑ *Harvard* architecture
 - ❖ Different busses for instructions and data

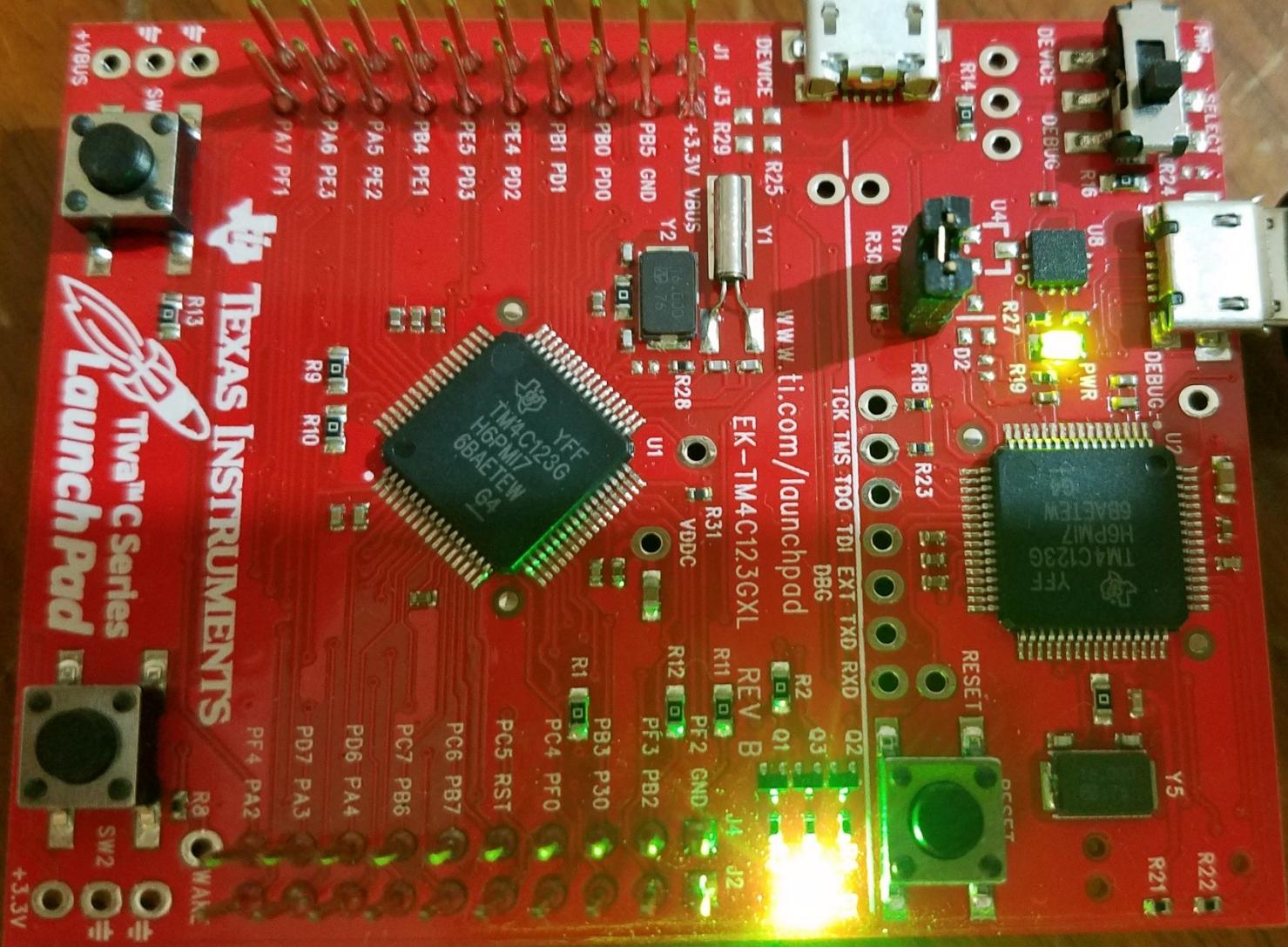
ARM ISA: Registers, Memory-map



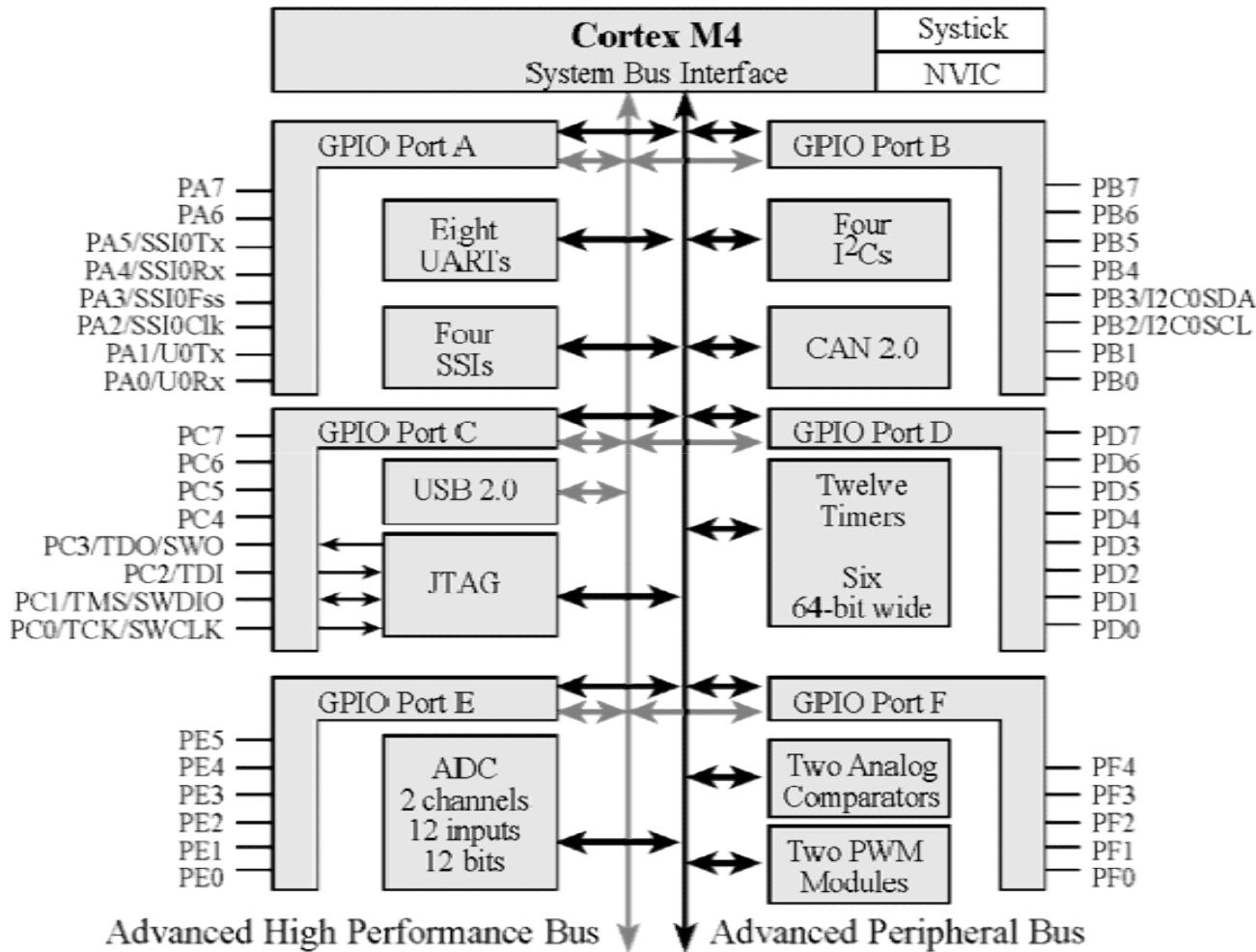
<u>Condition Code Bits</u>	<u>Indicates</u>
N	negative
Z	zero
V	overflow
C	carry

Result is negative
Result is zero
Signed overflow
Unsigned overflow

TI TM4C123
Microcontroller



Texas Instruments TM4C123



ARM Cortex-M4
+ 256K Flash
+ 32K RAM
+ JTAG
+ 43 Ports
+ SysTick
+ ADC
+ UART

ARM data instructions

- Basic format:

ADD R0,R1,R2

– Computes R1+R2, stores in R0.

- Immediate operand:

ADD R0,R1,#2

– Computes R1+R2, stores in R0.

ARM data instructions

- ADD, ADC : add (w. carry)
- SUB, SBC : subtract (w. carry)
- MUL: multiply
- AND, ORR, EOR
- BIC : bit clear
- LSL, LSR : logical shift left/right
- ROR : rotate right

ARM load/store instructions

- LDR, LDRH, LDRB : load (half-word, byte)
- STR, STRH, STRB : store (half-word, byte)
- Addressing modes:
 - register indirect : LDR R0, [R1]
 - with constant : LDR R0, [R1,#4]

ARM LDR instruction

- Load from memory into a register

```
LDR R8, [R10]
```

Example: C assignments

- C:

```
x = (a + b) - c;
```

- Assembler:

```
LDR R4,=A          ; get address for a
LDR R0,[R4]         ; get value of a
LDR R4,=B          ; get address for b, reusing r4
LDR R1,[R4]         ; get value of b
ADD R3,R0,R1        ; compute a+b
LDR R4,=C          ; get address for c
LDR R2,[R4]         ; get value of c
```

C assignment, cont'd.

```
SUB R3,R3,R2      ; complete computation of x  
LDR R4,=X        ; get address for x  
STR R3,[R4]       ; store value of x
```

Example: C assignment

- C:

```
y = a*(b+c);
```

- Assembler:

```
LDR R4,=B ; get address for b  
LDR R0,[R4] ; get value of b  
LDR R4,=C ; get address for c  
LDR R1,[R4] ; get value of c  
ADD R2,R0,R1 ; compute partial result  
LDR R4,=A ; get address for a  
LDR R0,[R4] ; get value of a
```

C assignment, cont'd.

```
MUL R2,R2,R0 ; compute final value for y  
LDR R4,=Y ; get address for y  
STR R2,[R4] ; store y
```

Example: C assignment

- C:

```
Z = (A << 2) | (B & 15);
```

- Assembler:

```
LDR R4,=A ; get address for a  
LDR R0,[R4] ; get value of a  
LSL R5,R0,#2 ; perform shift  
LDR R4,=B ; get address for b  
LDR R1,[R4] ; get value of b  
AND R1,R1,#15 ; perform AND  
ORR R1,R5,R1 ; perform OR
```

C assignment, cont'd.

```
LDR R4,=Z ; get address for z  
STR R1,[R4] ; store value for z
```

Example: if statement

- C:

```
if (a > b) { x = 5; y = c + d; } else x = c - d;
```

- Assembler:

; compute and test condition

```
LDR R4,=A ; get address for a
```

```
LDR R0,[R4] ; get value of a
```

```
LDR R4,=B ; get address for b
```

```
LDR R1,[R4] ; get value for b
```

```
CMP R0,R1 ;
```

```
BLE fblock ;
```

If statement, cont'd.

```
MOV R0,#5 ; generate value for x  
LDR R4,=X ; get address for x  
STR R0,[R4] ; store x  
LDR R4,=C ; get address for c  
LDR R0,[R4] ; get value of c  
LDR R4,=D ; get address for d  
LDR R1,[R4] ; get value of d  
ADD R0,R0,R1 ; compute y  
LDR R4,=Y ; get address for y  
STR R0,[R4] ; store y  
B after ; branch around false block
```

If statement, cont'd.

```
fblock LDR R4,=C ; get address for c  
LDR R0,[R4] ; get value of c  
LDR R4,=D ; get address for d  
LDR R1,[R4] ; get value for d  
SUB R0,R0,R1 ; compute c-d  
LDR R4,=X ; get address for x  
STR R0,[R4] ; store value of x  
after ...
```

Example

- C:

```
for (i=0, f=0; i<N; i++)  
    f = f + c[i]*x[i];
```

- Assembler

```
; loop initiation code  
MOV R0,#0 ; use r0 for I  
MOV R8,#0 ; use separate index for arrays  
LDR R2,=N ; get address for N  
LDR R1,[R2] ; get value of N  
MOV R2,#0 ; use r2 for f
```

Example, cont'd

```
LDR R3,=C ; load r3 with base of c
LDR R5,=X ; load r5 with base of x
; loop body
loop  LDR R4,[R3,R8] ; get c[i]
      LDR r6,[R5,R8] ; get x[i]
      MUL R4,R4,R6 ; compute c[i]*x[i]
      ADD R2,R2,R4 ; add into running sum
      ADD R8,R8,#4 ; add one word offset to array index
      ADD R0,R0,#1 ; add 1 to i
      CMP R0,R1 ; exit?
      BLT loop ; if i < N, continue
```

To set

The **or** operation to set bits 1 and 0 of a register.

The other six bits remain constant.

Friendly software modifies just the bits that need to be.

`x |= 0x03;`

Assembly:

```
LDR R0,=x  
LDR R1,[R0]          ; read previous value  
ORR R1,R1,#0x03     ; set bits 0 and 1  
STR R1,[R0]          ; update
```

c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0
0	0	0	0	0	0	<u>1</u>	<u>1</u>

value of R1
0x03 constant
result of the ORR

To toggle

The **exclusive or** operation can also be used to toggle bits.

```
x ^= 0x80;
```

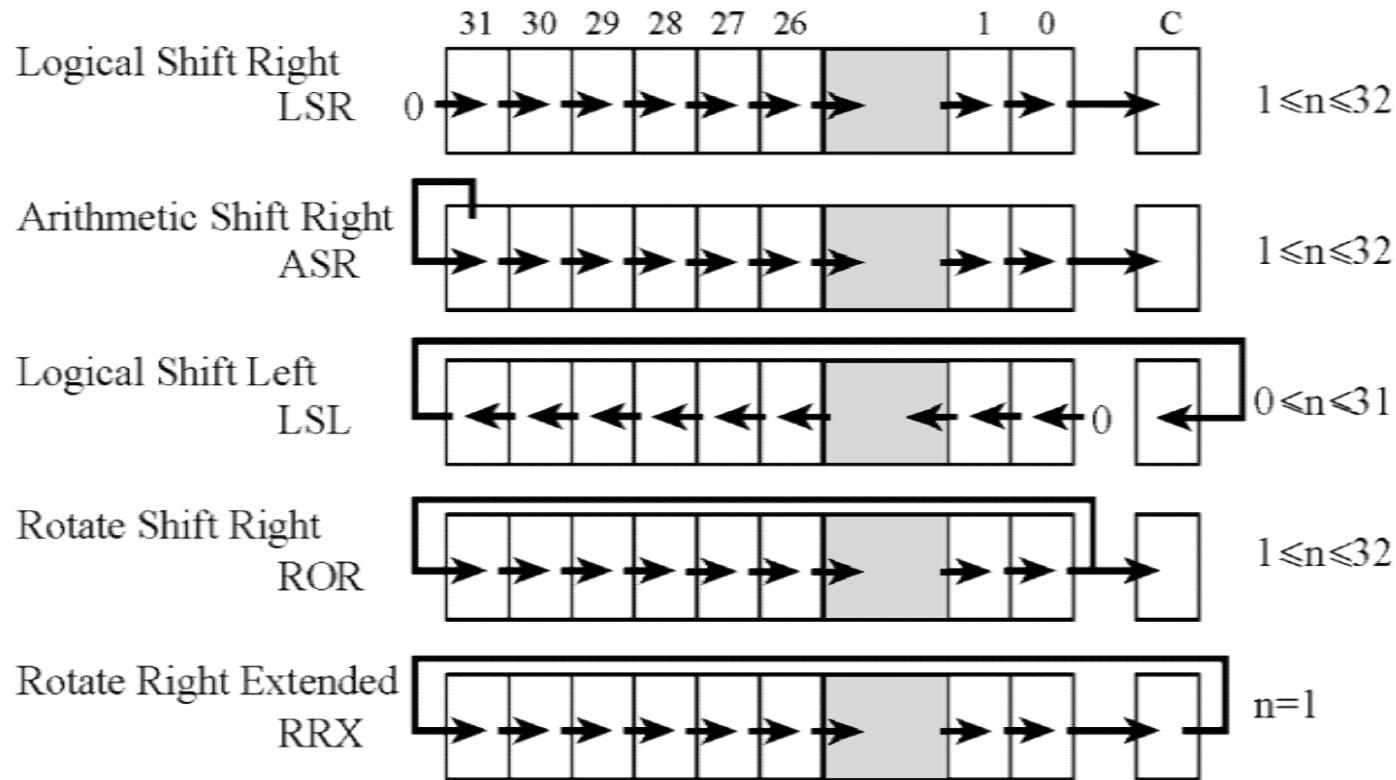
Assembly:

```
LDR R0,=x  
LDR R1,[R0]          ; read port D  
EOR R1,R1,#0x80      ; toggle bit 7  
STR R1,[R0]          ; update
```

$b_7 \ b_6 \ b_5 \ b_4 \ b_3 \ b_2 \ b_1 \ b_0$	
<u>1</u> 0 0 0 0 0 0 0	
$\sim b_7 \ b_6 \ b_5 \ b_4 \ b_3 \ b_2 \ b_1 \ b_0$	

value of R1
0x80 constant
result of the EOR

Shift Operations



Shift Example

High and **Low** are unsigned 4-bit components, which will be combined into a single unsigned 8-bit **Result**.

```
Result = (High<<4) | Low;
```

Assembly:

```
LDR R0,=High
LDR R1,[R0]           ; read value of High
LSL R1,R1,#4          ; shift into position
LDR R0,=Low
LDR R2,[R0]           ; read value of Low
ORR R1,R1,R2          ; combine the two parts
LDR R0,=Result
STR R1,[R0]           ; save the answer
```

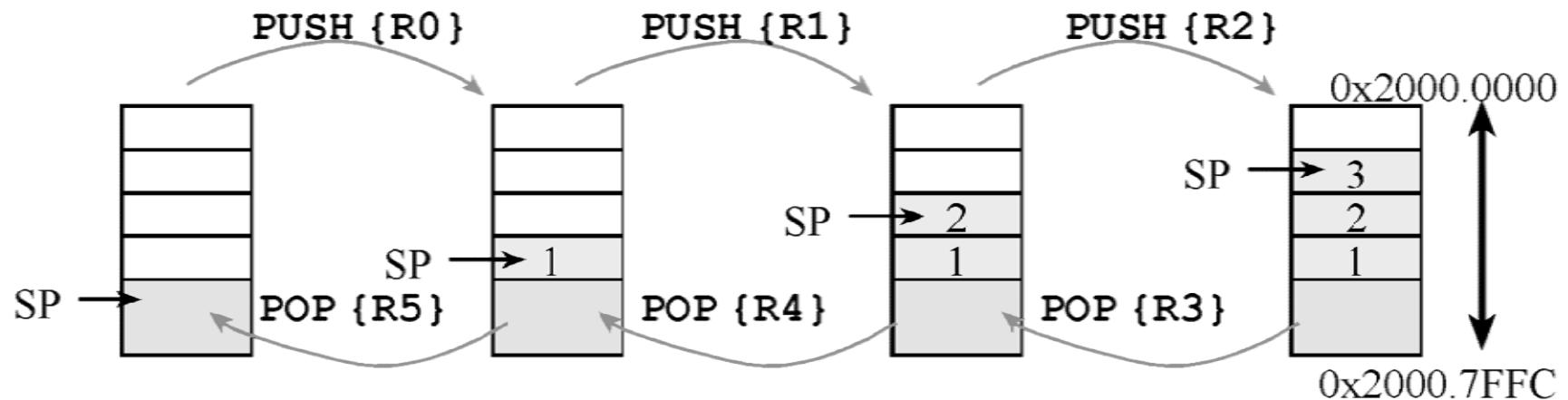
0 0 0 0 h ₃ h ₂ h ₁ h ₀	value of High in R1
h ₃ h ₂ h ₁ h ₀ 0 0 0 0	after last LSL
0 0 0 0 l ₃ l ₂ l ₁ l ₀	value of Low in R2
<u>h₃ h₂ h₁ h₀ l₃ l₂ l₁ l₀</u>	result of the ORR instruction

The Stack

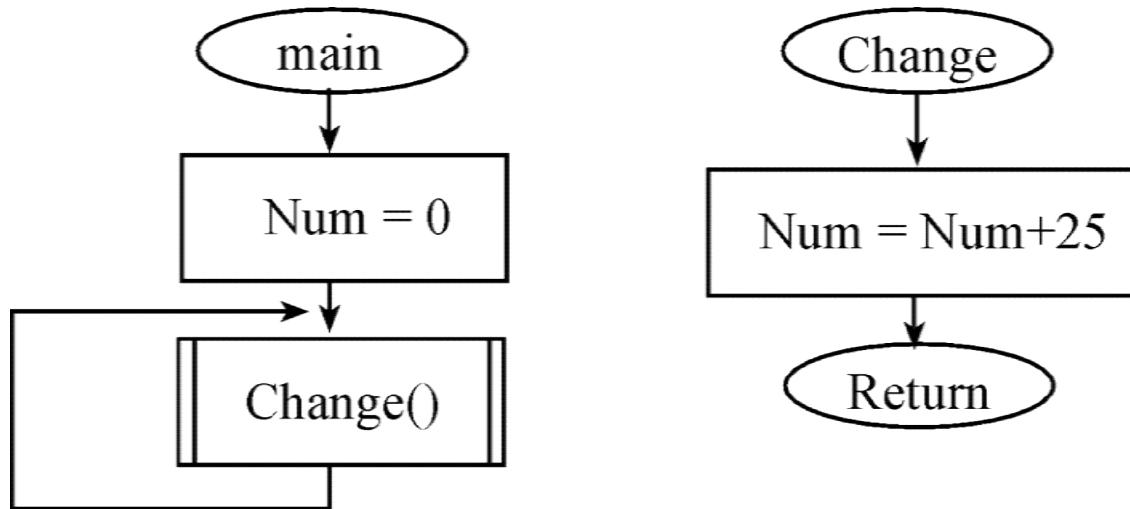
- Stack is last-in-first-out (LIFO) storage
 - ❖ 32-bit data
- Stack pointer, SP or R13, points to top element of stack
- Stack pointer *decremented* as data placed on stack
- **PUSH** and **POP** instructions used to load and retrieve data

The Stack

- Stack is last-in-first-out (LIFO) storage
 - ❖ 32-bit data
- Stack pointer, SP or R13, points to top element of stack
- Stack pointer *decremented* as data placed on stack (*incremented* when data is removed)
- **PUSH** and **POP** instructions used to load and retrieve data



Functions



Change	LDR R1,=Num ; 5) R1 = &Num
	LDR R0,[R1] ; 6) R0 = Num
	ADD R0,R0,#25 ; 7) R0 = Num+25
	STR R0,[R1] ; 8) Num = Num+25
	BX LR ; 9) return
main	LDR R1,=Num ; 1) R1 = &Num
	MOV R0,#0 ; 2) R0 = 0
	STR R0,[R1] ; 3) Num = 0
loop	BL Change ; 4) function call
	B loop ; 10) repeat

unsigned long Num;
void Change(void){
Num = Num+25;
}
void main(void){
Num = 0;
while(1){
Change();
}

Array Example

```
for(int i=0; i< 5;i++)  
{  
    aa[i] = i;  
    bb[i] = 5;  
}
```

AREA	DATA
aa	SPACE 4*10
bb	SPACE 4*10

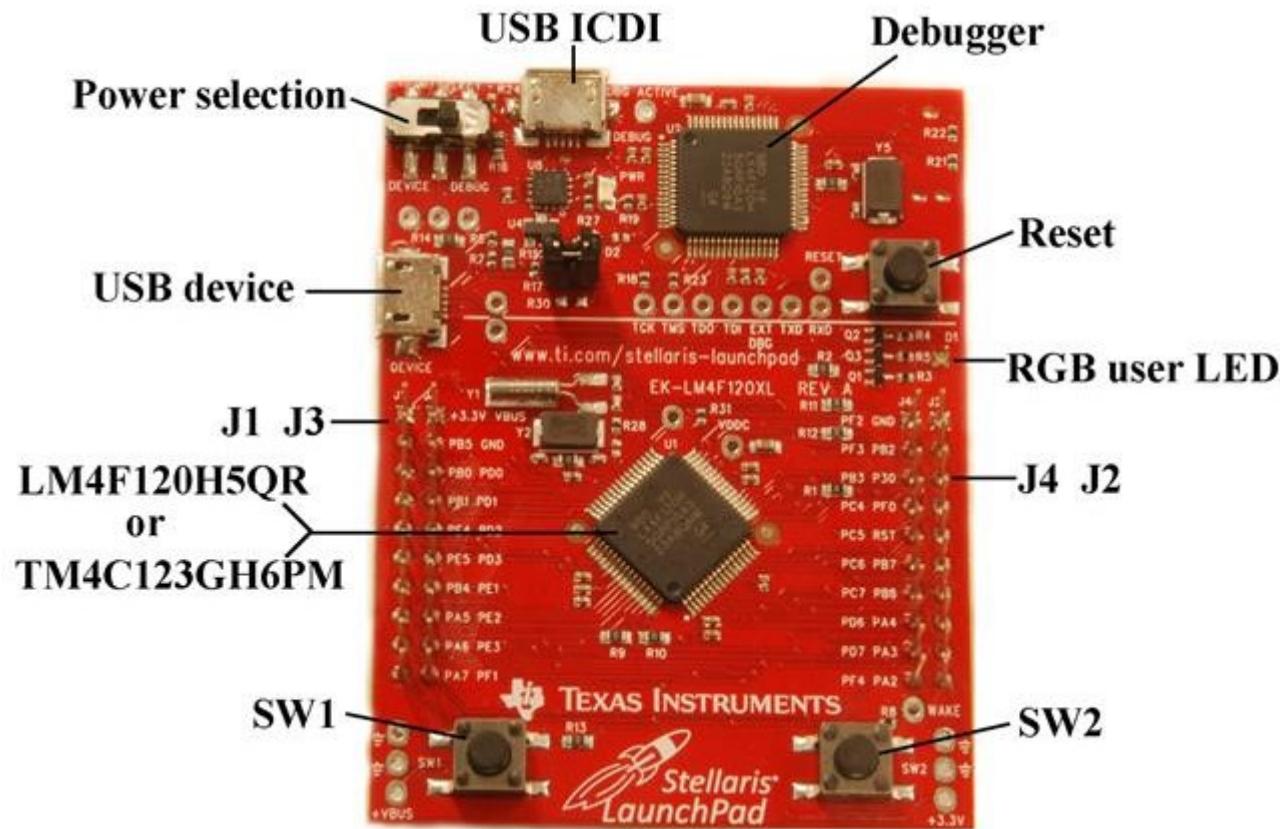
main	MOV R4,#0 ; i=0
	MOV R3,#5

forloop2

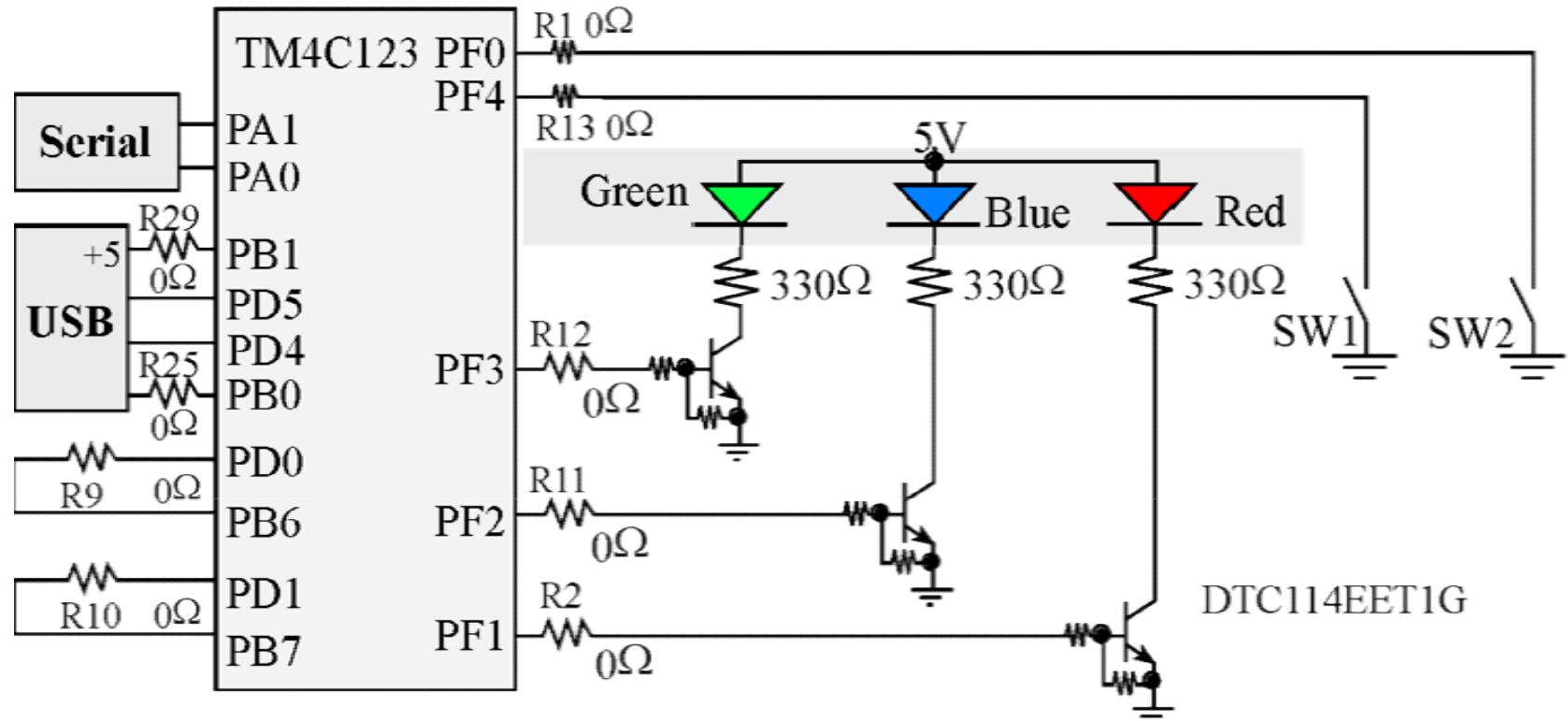
CMP R4,#5 ; is i<5
BGE forDone2
LDR R0,=aa ;aa[i] = aa+4*i
ASL R2,R4,#2 ; R2=i*4
STR R4,[R0,R2]
LDR R6,=bb
STR R3,[R6,R2] ; bb+i*4
ADD R4,R4,#1
B forloop2

forDone2

Tiva C Board



LaunchPad Switches and LEDs

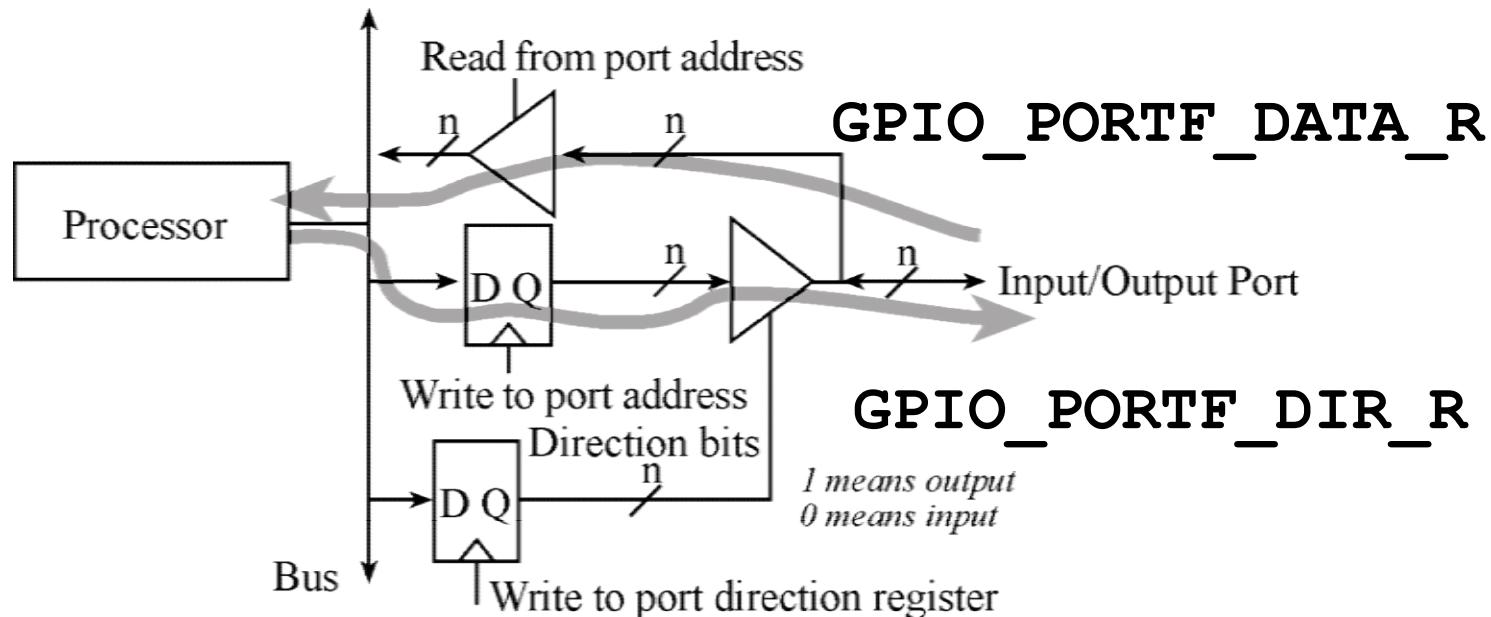


- The switches on the LaunchPad
 - ❖ Negative logic
 - ❖ Require internal pull-up (set bits in PUR)
- The PF3-1 LEDs are positive logic

I/O Programming

Address	7	6	5	4	3	2	1	0	Name
\$400F.E108	--	--	GPIOF	GPIOE	GPIOD	GPIOC	GPIOB	GPIOA	SYSCTL RCGC2 R
\$4000.43FC	DATA	GPIO PORTA DATA R							
\$4000.4400	DIR	GPIO PORTA DIR R							
\$4000.4420	SEL	GPIO PORTA AFSEL R							
\$4000.4510	PUE	GPIO PORTA PUR R							
\$4000.451C	DEN	GPIO PORTA DEN R							
\$4000.4524	1	1	1	1	1	1	1	1	GPIO PORTA CR R
\$4000.4528	0	0	0	0	0	0	0	0	GPIO PORTA AMSEL R
\$4000.53FC	DATA	GPIO PORTB DATA R							
\$4000.5400	DIR	GPIO PORTB DIR R							
\$4000.5420	SEL	GPIO PORTB AFSEL R							
\$4000.5510	PUE	GPIO PORTB PUR R							
\$4000.551C	DEN	GPIO PORTB DEN R							
\$4000.5524	1	1	1	1	1	1	1	1	GPIO PORTB CR R
\$4000.5528	0	0	AMSEL	AMSEL	0	0	0	0	GPIO PORTB AMSEL R
\$4000.63FC	DATA	DATA	DATA	DATA	JTAG	JTAG	JTAG	JTAG	GPIO PORTC DATA R
\$4000.6400	DIR	DIR	DIR	DIR	JTAG	JTAG	JTAG	JTAG	GPIO PORTC DIR R
\$4000.6420	SEL	SEL	SEL	SEL	JTAG	JTAG	JTAG	JTAG	GPIO PORTC AFSEL R
\$4000.6510	PUE	PUE	PUE	PUE	JTAG	JTAG	JTAG	JTAG	GPIO PORTC PUR R
\$4000.651C	DEN	DEN	DEN	DEN	JTAG	JTAG	JTAG	JTAG	GPIO PORTC DEN R
\$4000.6524	1	1	1	1	JTAG	JTAG	JTAG	JTAG	GPIO PORTC CR R
\$4000.6528	AMSEL	AMSEL	AMSEL	AMSEL	JTAG	JTAG	JTAG	JTAG	GPIO PORTC AMSEL R
\$4000.73FC	DATA	GPIO PORTD DATA R							
\$4000.7400	DID	GPIO PORTD DID D							

I/O Ports and Control Registers



The input/output direction of a bidirectional port is specified by its direction register.

GPIO_PORTF_DIR_R, specify if corresponding pin is input or output:

- ❖ 0 means input
- ❖ 1 means output

POR TA

```
GPIO_PORTA_DATA_R    EQU 0x400043FC  
GPIO_PORTA_DIR_R    EQU 0x40004400
```

GPIO Control

IO	Ain	0	1	2	3	4	5	6	7	8	9	14
PA0		Port	U0Rx							CAN1Rx		
PA1		Port	U0Tx							CAN1Tx		
PA2		Port		SSI0Clk								
PA3		Port		SSI0Fss								
PA4		Port		SSI0Rx								
PA5		Port		SSI0Tx								
PA6		Port			I ₂ C1SCL		M1PWM2					
PA7		Port			I ₂ C1SDA		M1PWM3					
PB0		Port	U1Rx							T2CCP0		
PB1		Port	U1Tx							T2CCP1		
PB2		Port			I ₂ C0SCL					T3CCP0		

I/O Programming & Direction Register

GPIO_PORTF_DIR_R

Which pins are input or output.

GPIO_PORTF_AFSEL_R

Activate the alternate functions

GPIO_PORTF_DEN_R:

Digital port

GPIO_PORTF_DATA_R

Perform input/output on the port.

Set Port Direction & Port Type

```
LDR    R1,=GPIO_PORTF_DIR_R  
MOV    R0,#0x0E  
STR    R0,[R1]
```

```
LDR    R1,=GPIO_PORTF_DEN_R  
MOV    R0,#0xFF  
STR    R0,[R1]
```

Set Port Direction & Port Type

```
GPIO_PORTF_DIR_R = 0x0E;      // PF4,PF0 in, PF3-1 out  
GPIO_PORTF_AFSEL_R = 0x00;    // disable alt funct on PF7-0  
GPIO_PORTF_DEN_R = 0x1F;     // enable digital I/O on PF4-0
```

I/O Ports and Control Registers

Address	7	6	5	4	3	2	1	0	Name
400F.E608	-	-	GPIOF	GPIOE	GPIOD	GPIOC	GPIOB	GPIOA	SYSCTL_RCGCGPIO_R
4002.53FC	-	-	-	DATA	DATA	DATA	DATA	DATA	GPIO_PORTF_DATA_R
4002.5400	-	-	-	DIR	DIR	DIR	DIR	DIR	GPIO_PORTF_DIR_R
4002.551C	-	-	-	DEN	DEN	DEN	DEN	DEN	GPIO_PORTF_DEN_R

- **Initialization (executed once at beginning)**
 1. Write *DIR* bit, 1 for output or 0 for input
 2. Set *DEN* bits to 1 to enable data pins
- **Input/output from pin**

Input: Read from **GPIO_PORTF_DATA_R**
Output: Write **GPIO_PORTF_DATA_R**

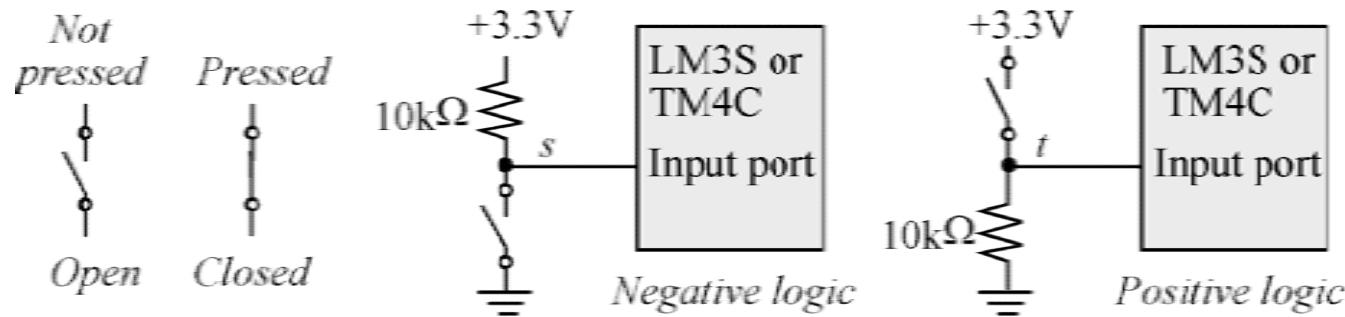
Port F LED Programming

```
DR R1, =GPIO_PORTF_DIR_R          ; R1 -> GPIO_PORTE_DIR_R
MOV R0, #0X0E                      ;PF0 , PF4 in and PF3-1 out
STR R0, [R1]                         ; set direction register

LDR R1, =GPIO_PORTF_DEN_R          ; R1 -> GPIO_PORTE_DEN_R
MOV R0, R0, #0xFF                   ; enable digital port
STR R0, [R1]                         ; set digital enable register

LDR R1, =GPIO_PORTF_DATA_R
MOV R0, #0x02
STR R0, [R1]
```

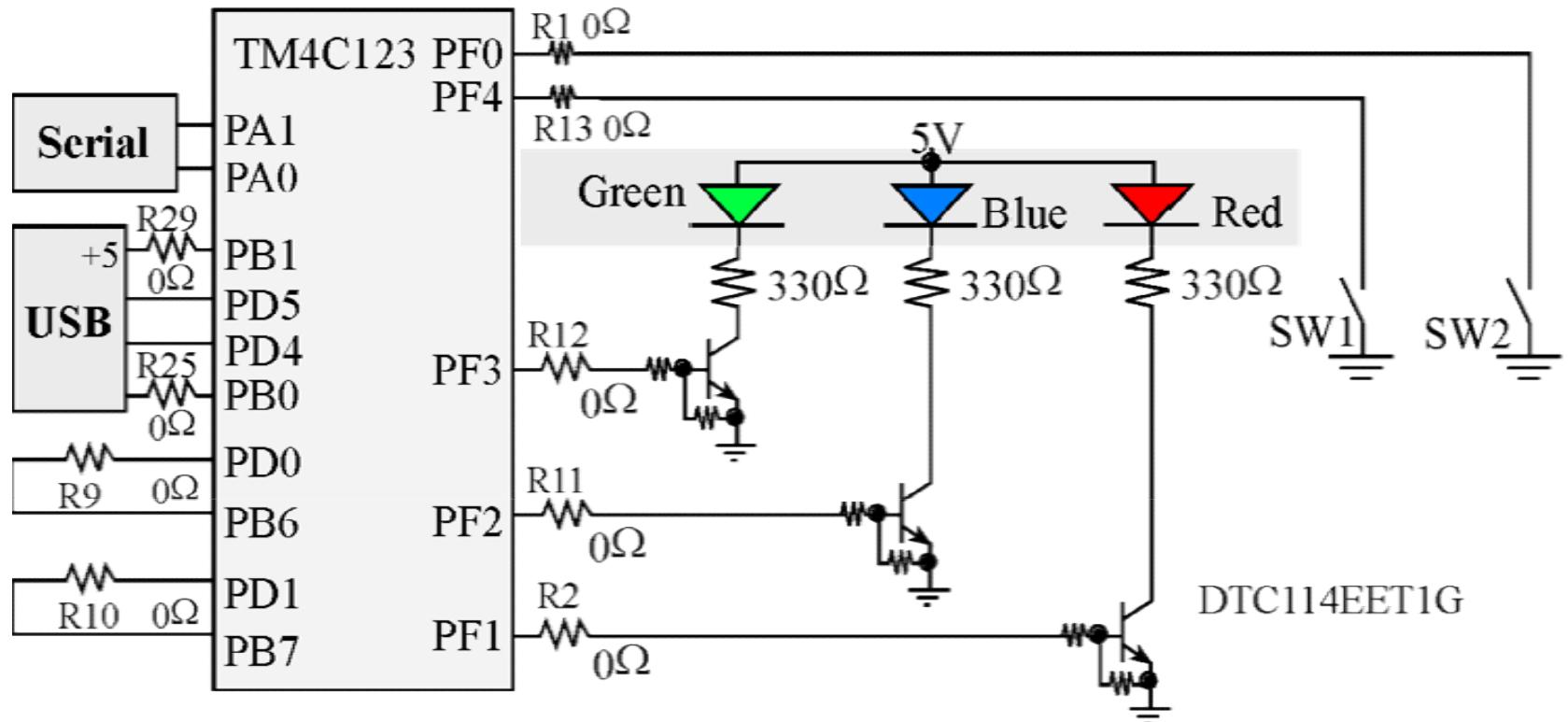
Switch Interfacing



Assembly:

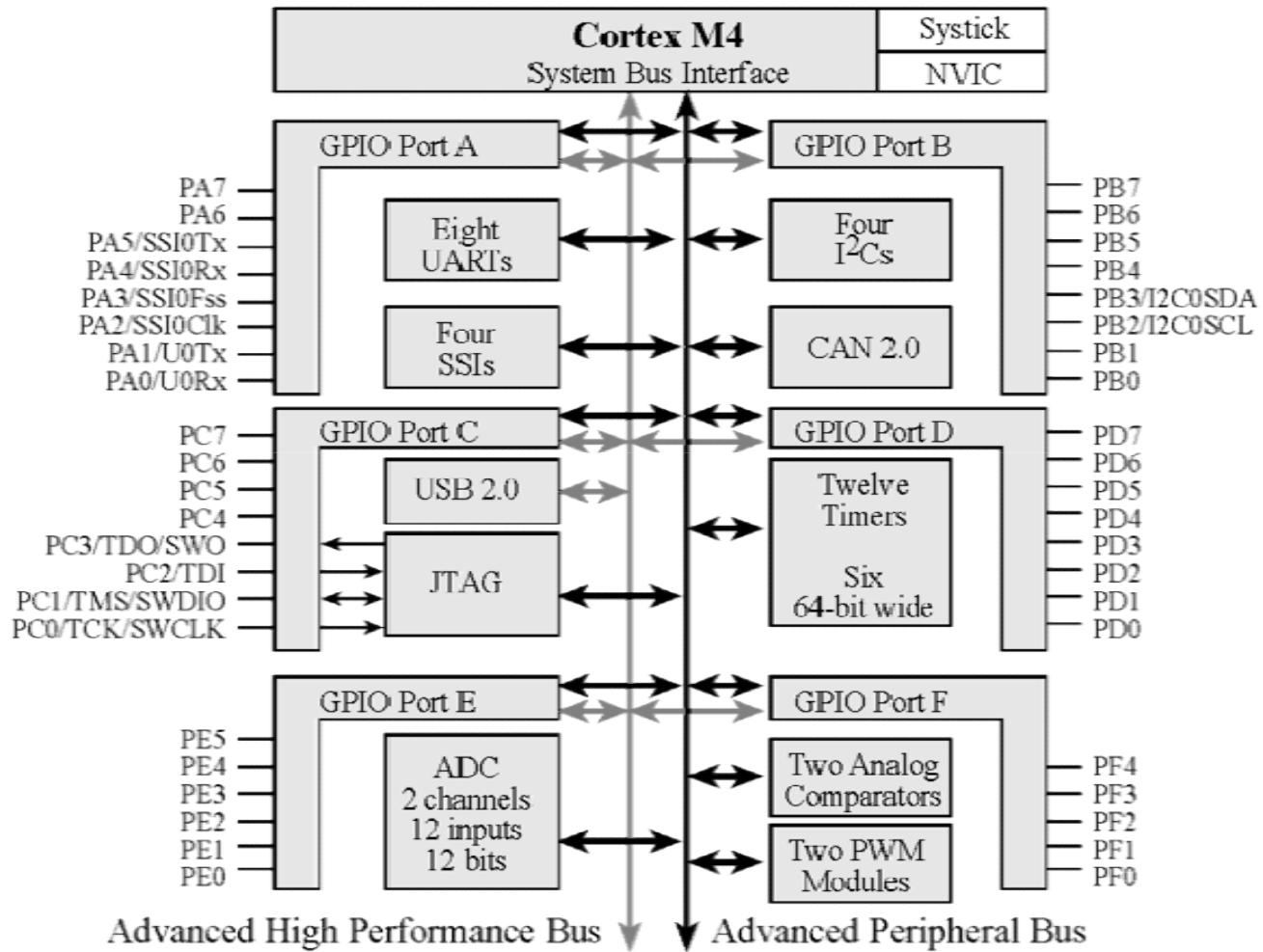
```
LDR R1,=GPIO_PORTF_DATA_R  
LDR R0,[R1]          ; read port F  
AND R0,R0,#0x11      ; PF4-PF0
```

LaunchPad Switches and LEDs



- The switches on the LaunchPad
 - ❖ Negative logic
 - ❖ Require internal pull-up (set bits in PUR)
- The PF3-1 LEDs are positive logic

Texas Instruments TM4C123



I/O Programming

Address	7	6	5	4	3	2	1	0	Name
\$400F.E108	--	--	GPIOF	GPIOE	GPIOD	GPIOC	GPIOB	GPIOA	SYSCTL RCGC2 R
\$4000.43FC	DATA	GPIO PORTA DATA R							
\$4000.4400	DIR	GPIO PORTA DIR R							
\$4000.4420	SEL	GPIO PORTA AFSEL R							
\$4000.4510	PUE	GPIO PORTA PUR R							
\$4000.451C	DEN	GPIO PORTA DEN R							
\$4000.4524	1	1	1	1	1	1	1	1	GPIO PORTA CR R
\$4000.4528	0	0	0	0	0	0	0	0	GPIO PORTA AMSEL R
\$4000.53FC	DATA	GPIO PORTB DATA R							
\$4000.5400	DIR	GPIO PORTB DIR R							
\$4000.5420	SEL	GPIO PORTB AFSEL R							
\$4000.5510	PUE	GPIO PORTB PUR R							
\$4000.551C	DEN	GPIO PORTB DEN R							
\$4000.5524	1	1	1	1	1	1	1	1	GPIO PORTB CR R
\$4000.5528	0	0	AMSEL	AMSEL	0	0	0	0	GPIO PORTB AMSEL R
\$4000.63FC	DATA	DATA	DATA	DATA	JTAG	JTAG	JTAG	JTAG	GPIO PORTC DATA R
\$4000.6400	DIR	DIR	DIR	DIR	JTAG	JTAG	JTAG	JTAG	GPIO PORTC DIR R
\$4000.6420	SEL	SEL	SEL	SEL	JTAG	JTAG	JTAG	JTAG	GPIO PORTC AFSEL R
\$4000.6510	PUE	PUE	PUE	PUE	JTAG	JTAG	JTAG	JTAG	GPIO PORTC PUR R
\$4000.651C	DEN	DEN	DEN	DEN	JTAG	JTAG	JTAG	JTAG	GPIO PORTC DEN R
\$4000.6524	1	1	1	1	JTAG	JTAG	JTAG	JTAG	GPIO PORTC CR R
\$4000.6528	AMSEL	AMSEL	AMSEL	AMSEL	JTAG	JTAG	JTAG	JTAG	GPIO PORTC AMSEL R
\$4000.73FC	DATA	GPIO PORTD DATA R							
\$4000.7400	DID	GPIO PORTD DID D							

SYSCTL_PGPIO_R

	7	6	5	4	3	2	1	0	Name
Address	-	-	GPIOF	GPIOE	GPIOD	GPIOC	GPIOB	GPIOA	SYSCTL_RCGCGPIO_R
\$400F.E608	-	-	GPIOF	GPIOE	GPIOD	GPIOC	GPIOB	GPIOA	SYSCTL_PGPIO_R
\$400F.EA08	-								
\$20043FC	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	GPIO_PORTA_DATA_R
	DIR	DIR	DIR	DIR	DIR	DIR	DIR	DIR	GPIO_PORTA_DIR_R

I/O Programming

```
#include "inc/tm4c123gh6pm.h"
void PortF_Init(void){
    SYSCTL_RCGCGPIO_R |= 0x00000020; // activate Port F
    while((SYSCTL_PRCPIO_R&0x00000020) == 0){};
    GPIO_PORTF_CR_R = 0x1F; // allow changes to PF4-0
    GPIO_PORTF_DIR_R = 0x0E; // PF4,PF0 in, PF3-1 out
    GPIO_PORTF_PUR_R = 0x11; // pull-up on PF0 and PF4
    GPIO_PORTF_DEN_R = 0x1F; // digital I/O on PF4-0
}
uint32_t PortF_Input(void){
    return (GPIO_PORTF_DATA_R&0x11); // read PF4,PF0
    inputs
}
void PortF_Output(uint32_t data){ // write PF3-PF1 outputs
    GPIO_PORTF_DATA_R = data;
}
```

For Loops for time delay

```
Delay SUBS R0,R0,#1
      BNE  Delay
      BX   LR

Main  BL   LED_Init
Loop   MOV  R0,#1
       BL   LED_Out
       LDR  R0,=250000
       BL   Delay
       MOV  R0,#0
       BL   LED_Out
       LDR  R0,=250000
       BL   Delay
       B    Loop
```

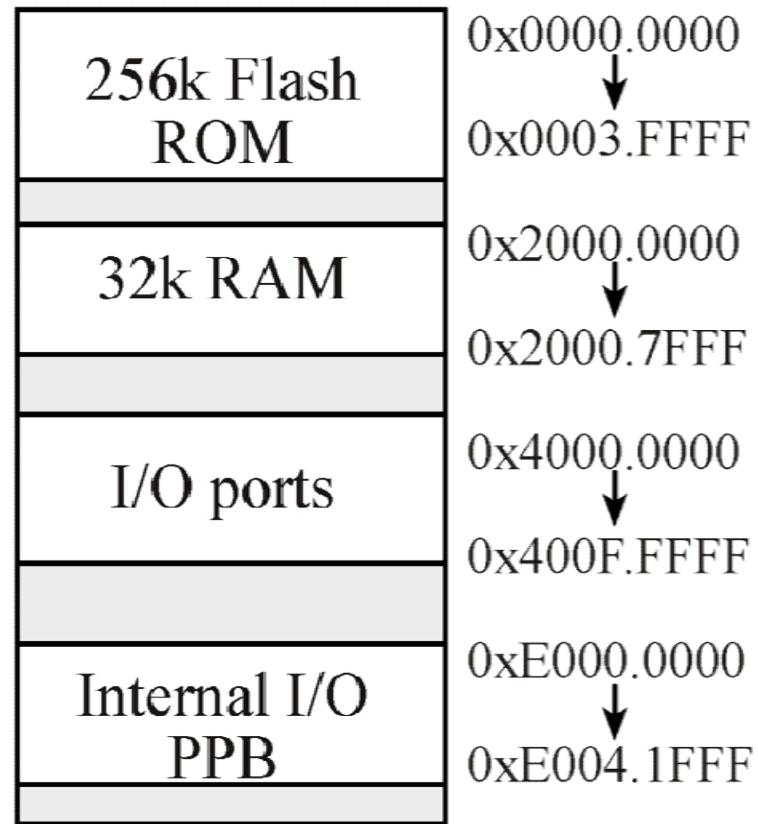
```
void Delay(uint32_t time) {
    while(time) {
        time--;
    }
}

int main(void) {
    LED_Init();
    while(1) {
        LED_Out(1);
        Delay(2500000);
        LED_Out(0);
        Delay(2500000);
    }
}
```

SysTick Timer

- Timer/Counter operation
 - 24-bit counter *decrements* at bus clock frequency
 - With 80 MHz bus clock, decrements every 12.5 ns
 - Counting is from $n \rightarrow 0$

ARM Memory-map



TI TM4C123
Microcontroller

SysTick Timer

Address	31-24	23-17	16	15-3	2	1	0	Name
\$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	NVIC_ST_CTRL_R
\$E000E014	0	24-bit RELOAD value						NVIC_ST_RELOAD_R
\$E000E018	0	24-bit CURRENT value of SysTick counter						NVIC_ST_CURRENT_R

- Initialization (4 steps)
 - Step1: Clear ENABLE to stop counter
 - Step2: Specify the RELOAD value
 - Step3: Clear the counter via NVIC_ST_CURRENT_R
 - Step4: Set NVIC_ST_CTRL_R
 - CLK_SRC = 1 (bus clock is the only option)
 - INTEN = 0 for no interrupts
 - ENABLE = 1 to enable

SysTick Timer Registers

```
#define NVIC_ST_CTRL_R(*((volatile uint32_t *)0xE000E010))  
  
#define NVIC_ST_RELOAD_R(*((volatile uint32_t *)0xE000E014))  
  
#define NVIC_ST_CURRENT_R(*((volatile uint32_t *)0xE000E018))
```

SysTick Timer Example

```
void SysTick_Init(void){  
    NVIC_ST_CTRL_R = 0; // 1) disable SysTick during setup  
    NVIC_ST_RELOAD_R = 0x00FFFFFF; // 2) maximum reload value  
    NVIC_ST_CURRENT_R = 0; // 3) any write to CURRENT clears it  
    NVIC_ST_CTRL_R = 0x00000005; // 4) enable SysTick with core clock  
}  
  
// The delay parameter is in units of the 80 MHz core clock(12.5 ns)  
void SysTick_Wait(uint32_t delay){  
    NVIC_ST_RELOAD_R = delay-1; // number of counts  
    NVIC_ST_CURRENT_R = 0; // any value written to CURRENT clears  
    while((NVIC_ST_CTRL_R&0x00010000)==0){ // wait for flag  
    }  
}  
// Call this routine to wait for delay*10ms  
void SysTick_Wait10ms(uint32_t delay){  
    unsigned long i;  
    for(i=0; i<delay; i++){  
        SysTick_Wait(800000); // wait 10ms  
    }  
}
```

SysTick Timer

SysTick_Init

```
; disable SysTick during setup
    LDR R1, =NVIC_ST_CTRL_R
    MOV R0, #0          ; Clear Enable
    STR R0, [R1]

; set reload to maximum reload value
    LDR R1, =NVIC_ST_RELOAD_R
    LDR R0, =0x00FFFFFF; ; Specify RELOAD value
    STR R0, [R1]          ; reload at maximum

; writing any value to CURRENT clears it
    LDR R1, =NVIC_ST_CURRENT_R
    MOV R0, #0
    STR R0, [R1]          ; clear counter

; enable SysTick with core clock
    LDR R1, =NVIC_ST_CTRL_R
    MOV R0, #0x0005      ; Enable but no interrupts (later)
    STR R0, [R1]          ; ENABLE and CLK_SRC bits set
    BX LR
```

24-bit Countdown Timer

SysTick Timer

```
;-----SysTick_Wait-----  
; Time delay using busy wait.  
; Input: R0 delay parameter in units of the core clock  
;        80 MHz (12.5 nsec each tick)  
; Output: none  
; Modifies: R1  
SysTick_Wait  
    SUB  R0, R0, #1      ; delay-1  
    LDR  R1, =NVIC_ST_RELOAD_R  
    STR  R0, [R1]        ; time to wait  
    LDR  R1, =NVIC_ST_CURRENT_R  
    STR  R0, [R1]        ; any value written to CURRENT clears  
    LDR  R1, =NVIC_ST_CTRL_R  
SysTick_Wait_loop  
    LDR  R0, [R1]        ; read status  
    ANDS R0, R0, #0x00010000 ; bit 16 is COUNT flag  
    BEQ  SysTick_Wait_loop ; repeat until flag set  
    BX   LR
```

SysTick Timer

```
;-----SysTick_Wait10ms-----
; Call this routine to wait for R0*10 ms
; Time delay using busy wait. This assumes 80 MHz clock
; Input: R0 number of times to wait 10 ms before returning
; Output: none
; Modifies: R0
DELAY10MS EQU 800000 ; clock cycles in 10 ms
SysTick_Wait10ms
    PUSH {R4, LR} ; save R4 and LR
    MOVS R4, R0 ; R4 = R0 = remainingWaits
    BEQ SysTick_Wait10ms_done ; R4 == 0, done
SysTick_Wait10ms_loop
    LDR R0, =DELAY10MS ; R0 = DELAY10MS
    BL SysTick_Wait ; wait 10 ms
    SUBS R4, R4, #1 ; remainingWaits--
    BHI SysTick_Wait10ms_loop ; if(R4>0), wait another 10 ms
SysTick_Wait10ms_done
    POP {R4, PC}
```

SysTick Timer

Address	31-24	23-17	16	15-3	2	1	0	Name
\$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	NVIC_ST_CTRL_R
\$E000E014	0	24-bit RELOAD value						NVIC_ST_RELOAD_R
\$E000E018	0	24-bit CURRENT value of SysTick counter						NVIC_ST_CURRENT_R

- Initialization (4 steps)
 - Step1: Clear ENABLE to stop counter
 - Step2: Specify the RELOAD value
 - Step3: Clear the counter via NVIC_ST_CURRENT_R
 - Step4: Set NVIC_ST_CTRL_R
 - CLK_SRC = 1 (bus clock is the only option)
 - INTEN = 0 for no interrupts
 - ENABLE = 1 to enable

SysTick Timer

SysTick_Init

```
; disable SysTick during setup
    LDR R1, =NVIC_ST_CTRL_R
    MOV R0, #0          ; Clear Enable
    STR R0, [R1]

; set reload to maximum reload value
    LDR R1, =NVIC_ST_RELOAD_R
    LDR R0, =0x00FFFFFF; ; Specify RELOAD value
    STR R0, [R1]          ; reload at maximum

; writing any value to CURRENT clears it
    LDR R1, =NVIC_ST_CURRENT_R
    MOV R0, #0
    STR R0, [R1]          ; clear counter

; enable SysTick with core clock
    LDR R1, =NVIC_ST_CTRL_R
    MOV R0, #0x0005      ; Enable but no interrupts
    STR R0, [R1]          ; ENABLE and CLK_SRC bits set
    BX LR
```

24-bit Countdown Timer

SysTick Timer

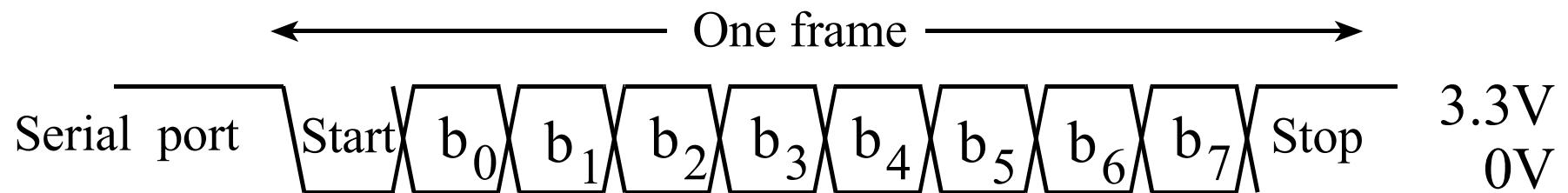
```
;-----SysTick_Wait-----  
; Time delay using busy wait.  
; Input: R0 delay parameter in units of the core clock  
;        80 MHz (12.5 nsec each tick)  
; Output: none  
; Modifies: R1  
SysTick_Wait  
    SUB  R0, R0, #1      ; delay-1  
    LDR  R1, =NVIC_ST_RELOAD_R  
    STR  R0, [R1]        ; time to wait  
    LDR  R1, =NVIC_ST_CURRENT_R  
    STR  R0, [R1]        ; any value written to CURRENT clears  
    LDR  R1, =NVIC_ST_CTRL_R  
SysTick_Wait_loop  
    LDR  R0, [R1]        ; read status  
    ANDS R0, R0, #0x00010000 ; bit 16 is COUNT flag  
    BEQ  SysTick_Wait_loop ; repeat until flag set  
    BX   LR
```

SysTick Timer

```
;-----SysTick_Wait10ms-----
; Call this routine to wait for R0*10 ms
; Time delay using busy wait. This assumes 80 MHz clock
; Input: R0 number of times to wait 10 ms before returning
; Output: none
; Modifies: R0
DELAY10MS EQU 800000 ; clock cycles in 10 ms
SysTick_Wait10ms
    PUSH {R4, LR} ; save R4 and LR
    MOVS R4, R0 ; R4 = R0 = remainingWaits
    BEQ SysTick_Wait10ms_done ; R4 == 0, done
SysTick_Wait10ms_loop
    LDR R0, =DELAY10MS ; R0 = DELAY10MS
    BL SysTick_Wait ; wait 10 ms
    SUBS R4, R4, #1 ; remainingWaits--
    BHI SysTick_Wait10ms_loop ; if(R4>0), wait another 10 ms
SysTick_Wait10ms_done
    POP {R4, PC}
```

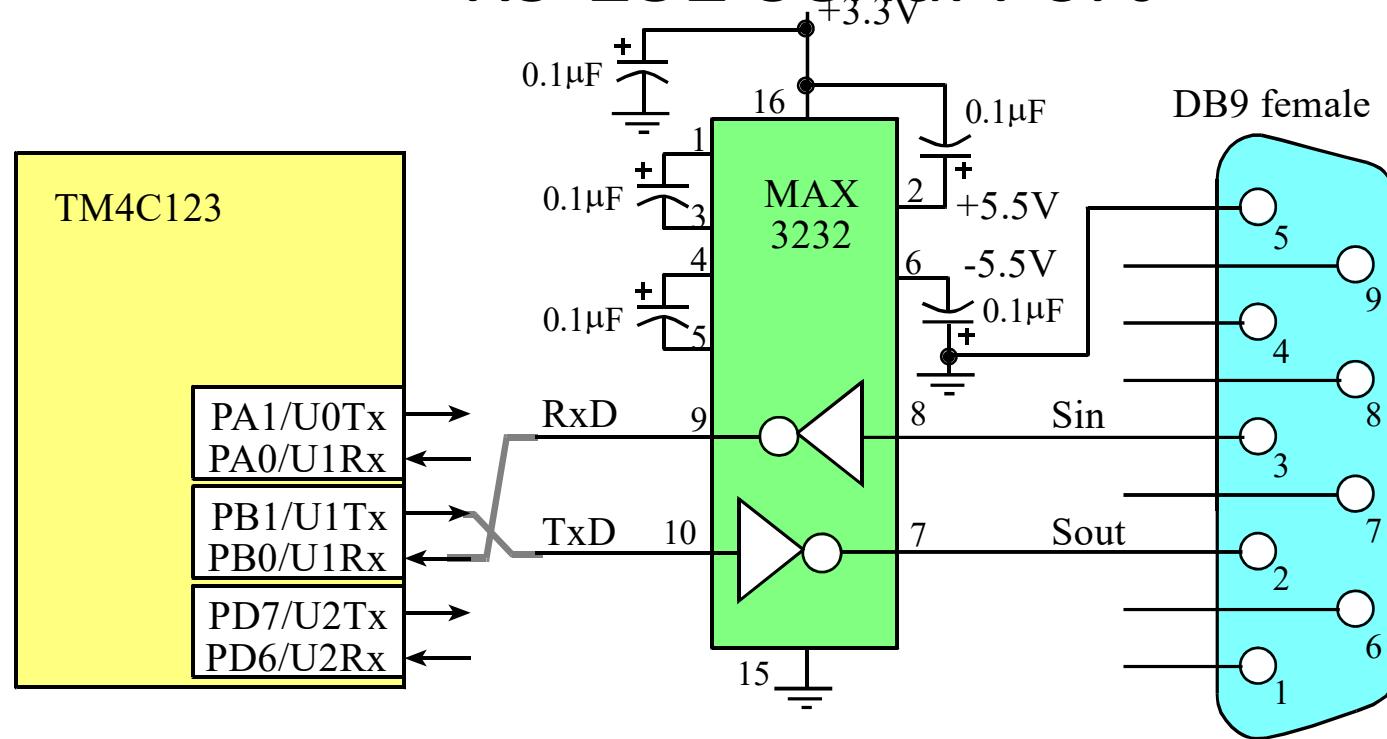
Universal Asynchronous Receiver/Transmitter (UART)

- UART (Serial Port) Interface



- Send/receive a *frame* of (5-8) data bits with a single (start) bit prefix and a 1 or 2 (stop) bit suffix
- Baud rate is total number of bits per unit time
 - Baudrate = 1 / bit-time
- Bandwidth is data per unit time
 - Bandwidth = (data-bits / frame-bits) * baudrate

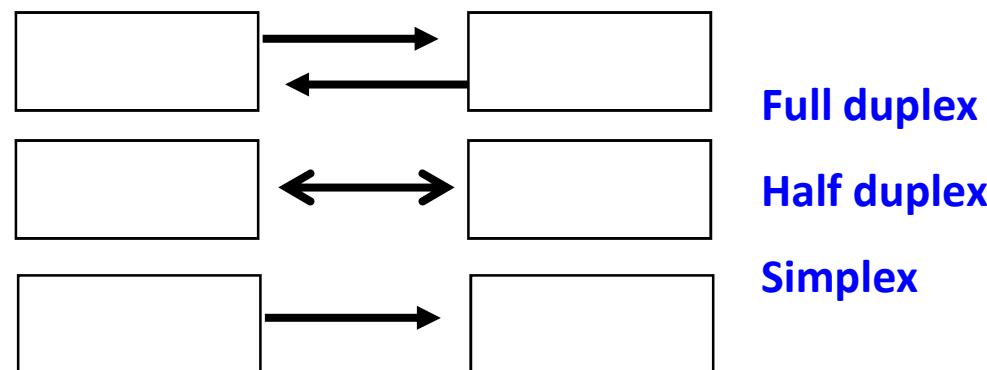
RS-232 Serial Port



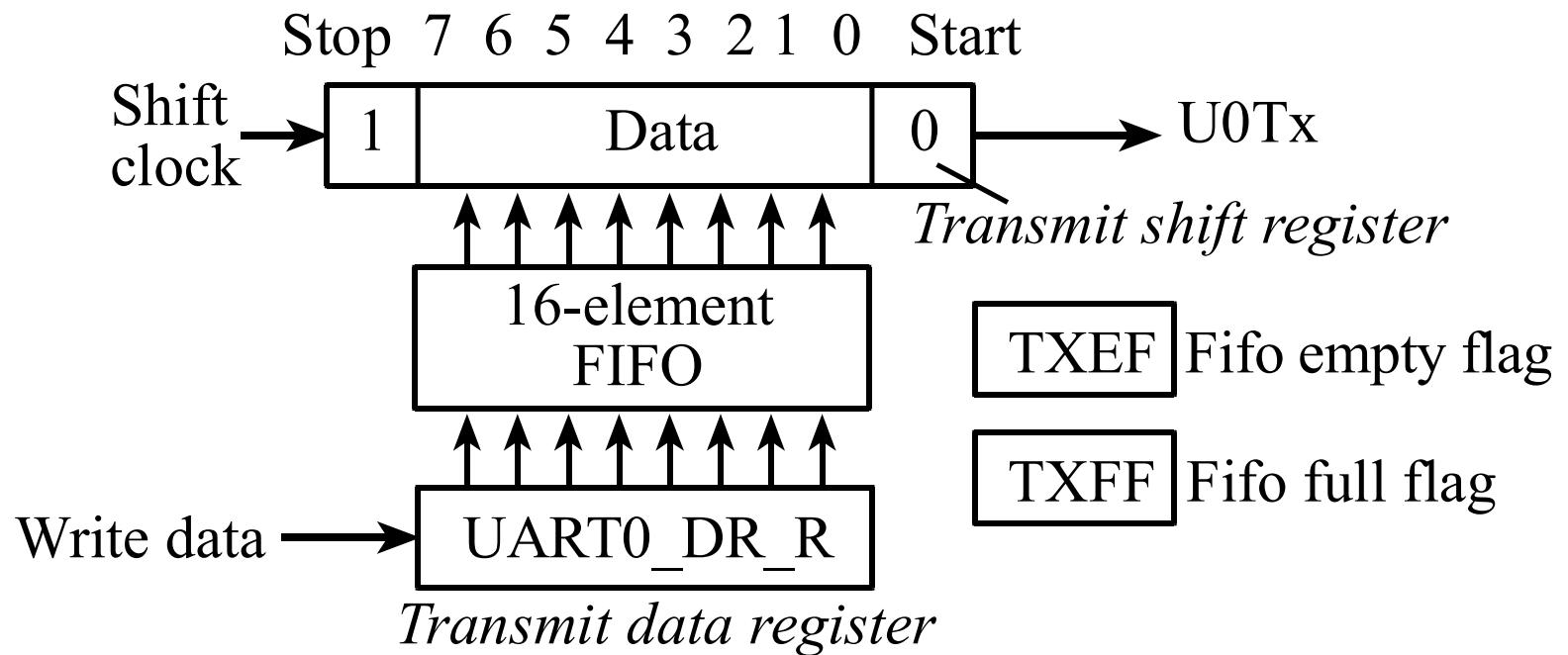
DB25 Pin	RS232 Name	DB9 Pin	EIA-574 Name	Signal	Description	True	DTE	DCE
2	BA	3	103	TxD	Transmit Data	-5.5V	out	in
3	BB	2	104	RxD	Receive Data	-5.5V	in	out
7	AB	5	102	SG	Signal Ground			

Serial I/O

- Serial communication
 - Transmit Data (TxD), Receive Data (RxD), and Signal Ground (SG) implement *duplex* communication link
 - Both communicating devices must operate at the same bit rate

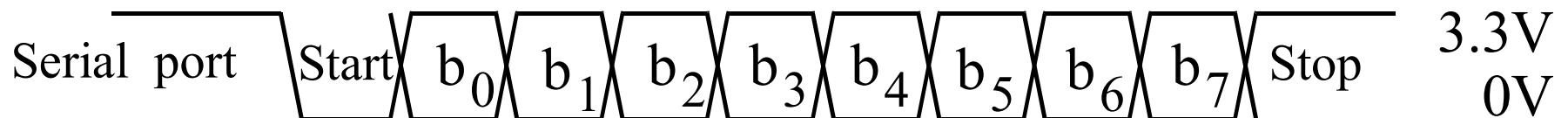


UART - Transmitter

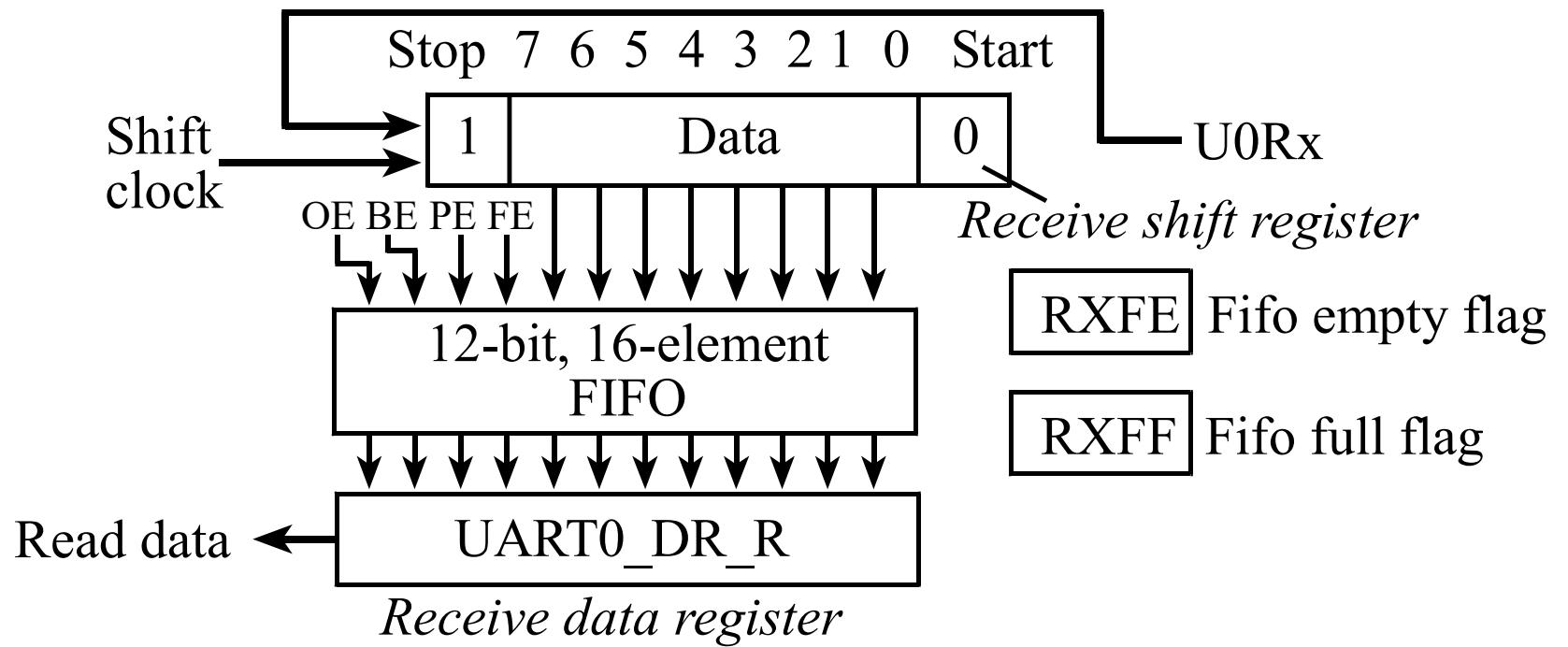


UART - Transmitter

- Tx Operation
 - Data written to `UART0_DR_R`
 - passes through 16-element FIFO
 - permits small amount of data rate matching between processor and UART
 - Shift clock is generated from 16x clock
 - ~~permits difference~~: One frame ~~1x clocks to be reconciled~~



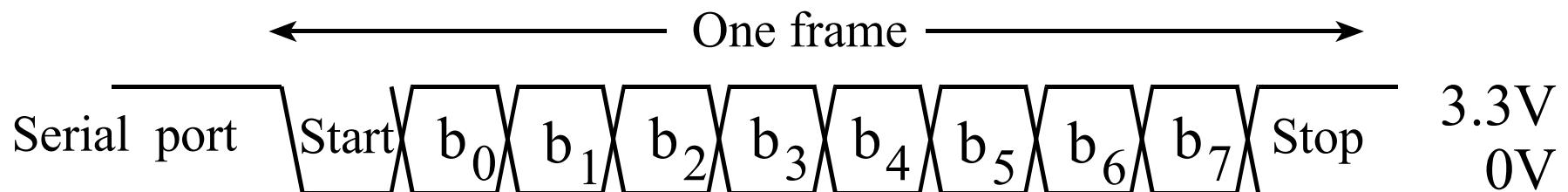
UART - Receiver



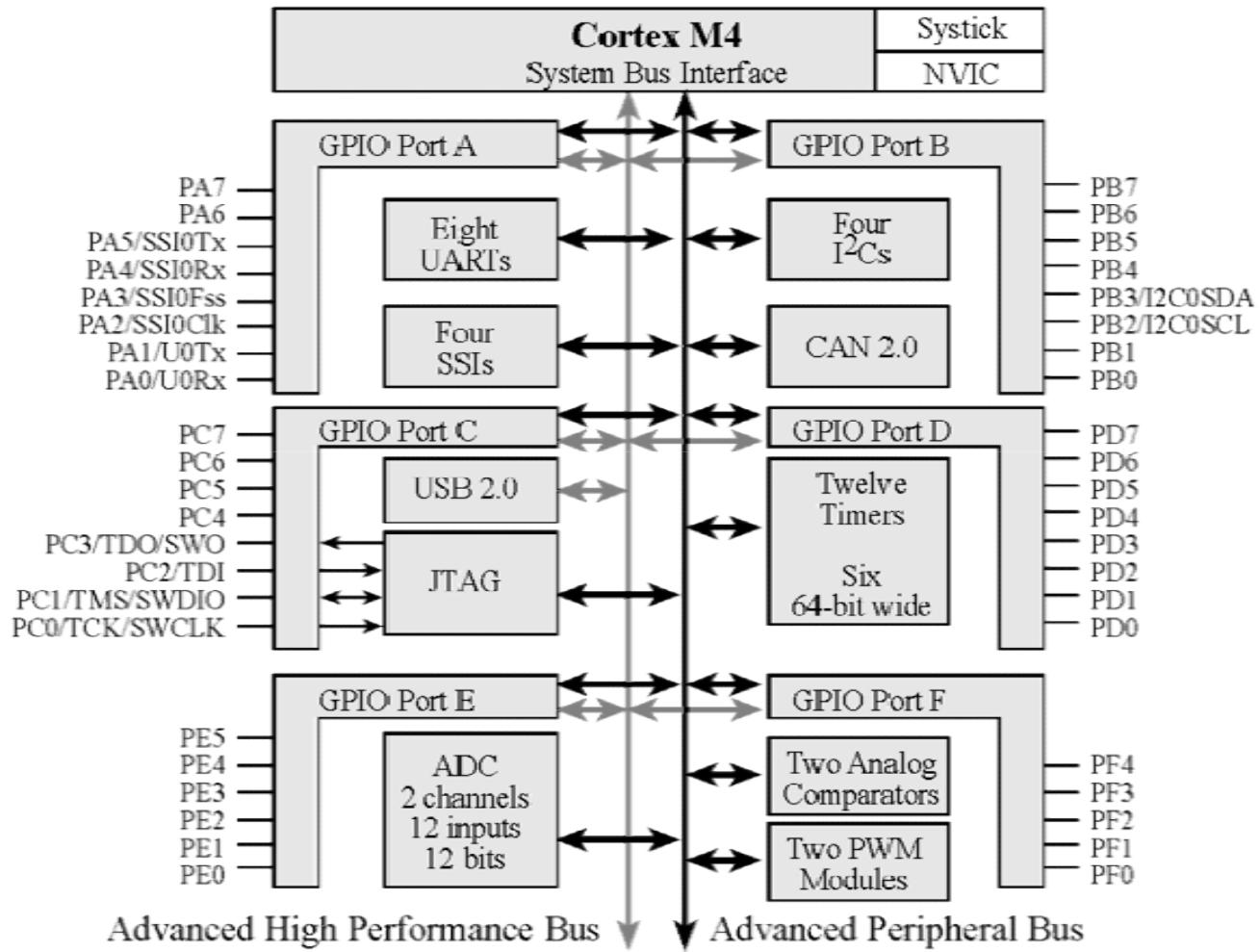
UART - Receiver

- Rx Operation

- **RXFE** is 0 when data are available
- **RXFF** is 1 when FIFO is full
- FIFO entries have four control bits
 - **BE** set when Tx signal held low for more than one frame (break)
 - **OE** set when FIFO is full and new frame has arrived
 - **PE** set if frame parity error
 - **FE** set if stop bit timing error



Texas Instruments TM4C123



TM4C UART0 – Registers

GPIO PCTL Register

IO	Ain	0	1	2	3	4	5	6	7	8	9	14
PA2		Port		SSI0Clk								
PA3		Port		SSI0Fss								
PA4		Port		SSI0Rx								
PA5		Port		SSI0Tx								
PA6		Port		I ₂ C1SCL		M1PWM2						
PA7		Port		I ₂ C1SDA		M1PWM3						
PB0		Port	U1Rx					T2CCP0				
PB1		Port	U1Tx					T2CCP1				
PB2		Port		I ₂ C0SCL				T3CCP0				
PB3		Port		I ₂ C0SDA				T3CCP1				
PB4	Ain10	Port		SSI2Clk		M0PWM2		T1CCP0	CAN0Rx			
PB5	Ain11	Port		SSI2Fss		M0PWM3		T1CCP1	CAN0Tx			
PB6		Port		SSI2Rx		M0PWM0		T0CCP0				
PB7		Port		SSI2Tx		M0PWM1		T0CCP1				
PC4	C1-	Port	U4Rx	U1Rx		M0PWM6	IDX1	WT0CCP0	U1RTS			
PC5	C1+	Port	U4Tx	U1Tx		M0PWM7	PhA1	WT0CCP1	U1CTS			
PC6	C0+	Port	U3Rx				PhB1	WT1CCP0	USB0open			
PC7	C0-	Port	U3Tx					WT1CCP1	USB0pflt			
PD0	Ain7	Port	SSI3Clk	SSI1Clk	I ₂ C3SCL	M0PWM6	M1PWM0	WT2CCP0				
PD1	Ain6	Port	SSI3Fss	SSI1Fss	I ₂ C3SDA	M0PWM7	M1PWM1	WT2CCP1				
PD2	Ain5	Port	SSI3Rx	SSI1Rx		M0Fault0		WT3CCP0	USB0open			
PD3	Ain4	Port	SSI3Tx	SSI1Tx			IDX0	WT3CCP1	USB0pflt			
PD6		Port	U2Rx			M0Fault0	PhA0	WT5CCP0				
PD7		Port	U2Tx				PhB0	WT5CCP1	NMI			
PE0	Ain3	Port	U7Rx									
PE1	Ain2	Port	U7Tx									
PE2	Ain1	Port										
PE3	Ain0	Port										
PE4	Ain9	Port	U5Rx		I ₂ C2SCL	M0PWM4	M1PWM2		CAN0Rx			
PE5	Ain8	Port	U5Tx		I ₂ C2SDA	M0PWM5	M1PWM3		CAN0Tx			
PF0		Port	U1RTS	SSI1Rx	CAN0Rx		M1PWM4	PhA0	T0CCP0	NMI	C0o	
PF1		Port	U1CTS	SSI1Tx			M1PWM5	PhB0	T0CCP1		C1o	TRD1
PF2		Port		I ₂ C1Clk		M0Fault0	M1PWM6		T1CCP0			TRD0
PF3		Port		SSI1Fss	CAN0Tx		M1PWM7		T1CCP1			TRCLK
PF4		Port				M1Fault0	IDX0	T2CCP0	TISSR0open			

TM4C UART Setup

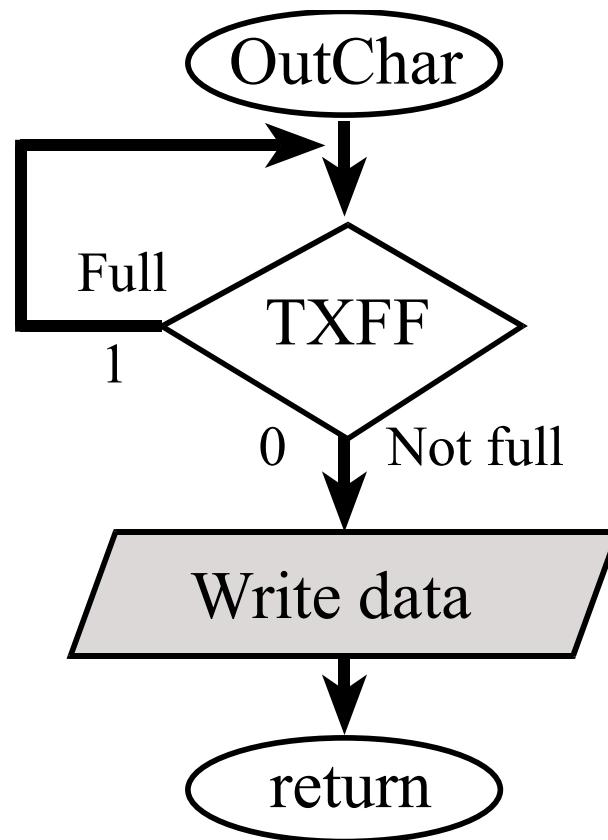
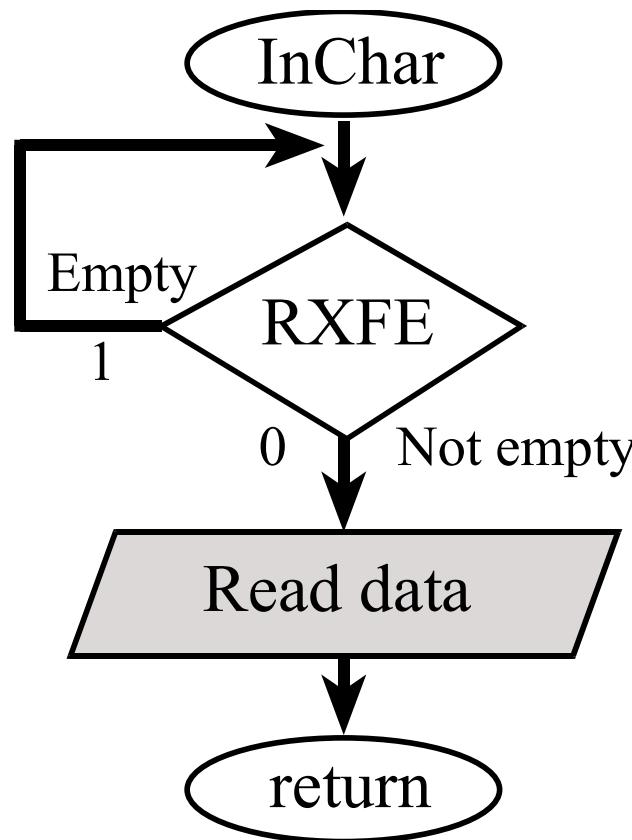
- UART0 operation
 - UART clock started in **SYSCTL_RCGCUART_R**
 - Digital port clock started in **SYSCTL_RCGCGPIO_R**
 - **UART0_CTL_R** contains UART enable (UARTEN), Tx (TXE), and Rx enable (RXE)
 - set each to 1 to enable
 - **UART disabled during initialization**
 - **UART1_IBRD_R** and **UART1_FBRD_R** specify baud rate
 - bit rate = (bus clock frequency)/(16*divider)
 - ex: want 19.2 kb/s and bus clock is 80 MHz
 - $80\text{ MHz}/(16 \cdot 19.2\text{ k}) = 26.04167 = 11010.000011_2$
 - Tx and Rx clock rates must be within 5% to avoid errors
 - **GPIO_PORTC_AFSEL_R** to choose alternate function
 - **GPIO_PORTC_DEN_R** Enable digital I/O on pins 1-0
 - **GPIO_PORTC_AMSEL_R** no Analog I/O on pins 1-0
 - **write to UART0_LCRH_R to activate**

TM4C UART Programming

```
// Assumes a 80 MHz bus clock, creates 115200 baud rate
void UART_Init(void){          // should be called only once
    SYSCTL_RCGCUART_R |= 0x00000002; // activate UART1
    SYSCTL_RCGCGPIO_R |= 0x00000004; // activate port C
    UART1_CTL_R &= ~0x00000001; // disable UART
    UART1_IBRD_R = 43;           // IBRD = int(80,000,000/(16*115,200)) = int(43.40278)
    UART1_FBRD_R = 26;           // FBRD = round(0.40278 * 64) = 26
    UART1_LCRH_R = 0x00000070; // 8 bit, no parity bits, one stop, FIFOs
    UART1_CTL_R |= 0x00000001; // enable UART
    GPIO_PORTC_AFSEL_R |= 0x30; // enable alt funct on PC5-4
    GPIO_PORTC_DEN_R |= 0x30; // configure PC5-4 as UART1
    GPIO_PORTC_PCTL_R = (GPIO_PORTC_PCTL_R&0xFF00FFFF)+0x00220000;
    GPIO_PORTC_AMSEL_R &= ~0x30; // disable analog on PC5-4
}
```

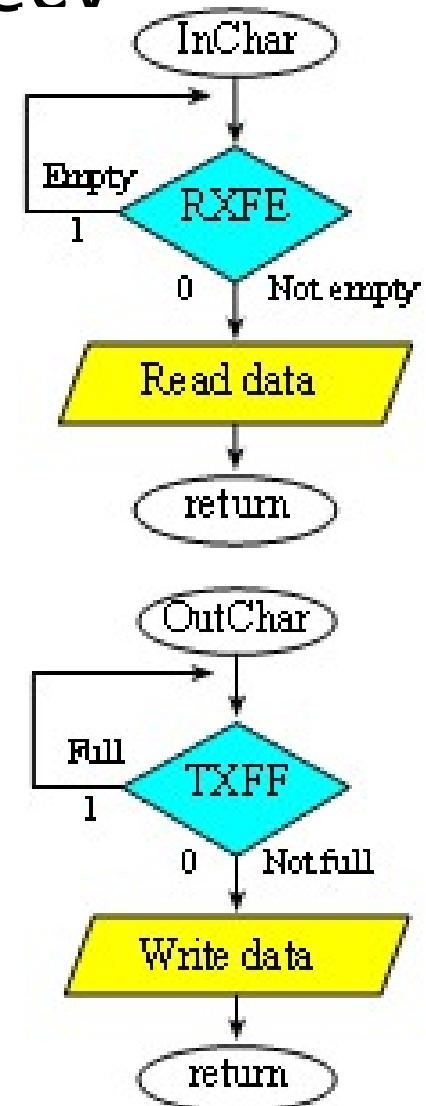
UART Synchronization

- Busy-wait operation



UART Busy-Wait Send/Recv

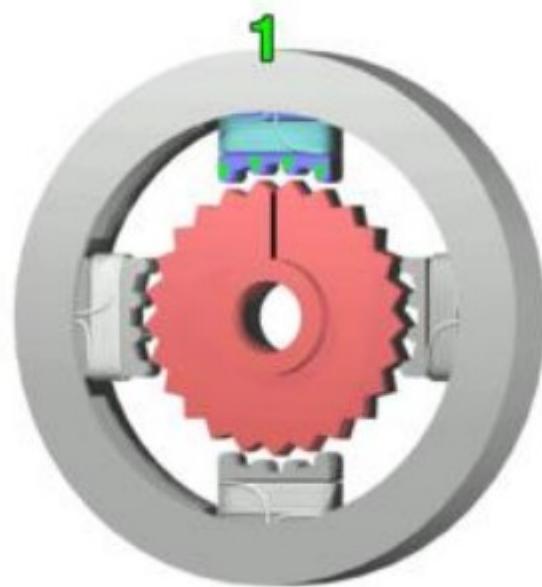
```
// Wait for new input,  
// then return ASCII code  
uint8_t UART_InChar(void) {  
    while((UART1_FR_R&0x0010) != 0);  
    // wait until RXFE is 0  
    return((uint8_t)(UART1_DR_R&0xFF));  
}  
  
// Wait for buffer to be not full,  
// then output  
void UART_OutChar(uint8_t data) {  
    while((UART1_FR_R&0x0020) != 0);  
    // wait until TXFF is 0  
    UART1_DR_R = data;  
}
```



Stepper Motor

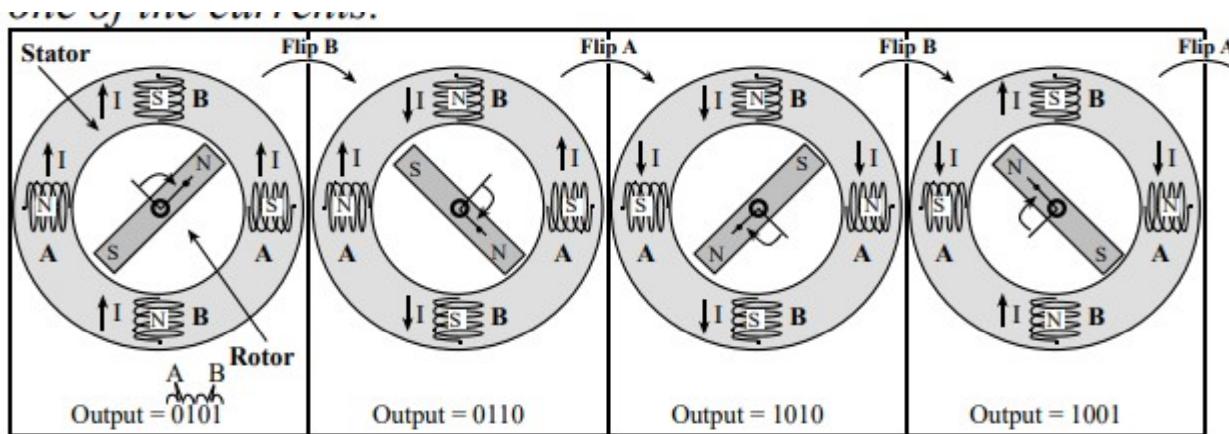


Stepper Motor

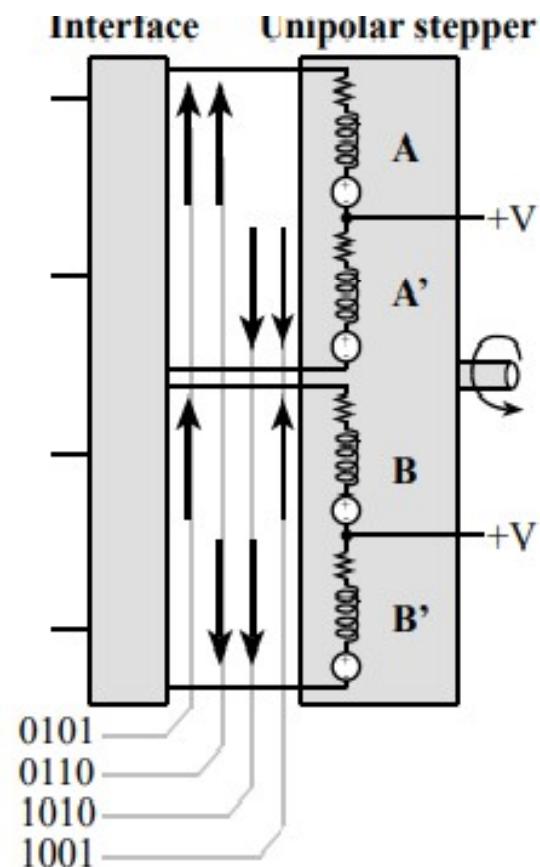


Steps / rev

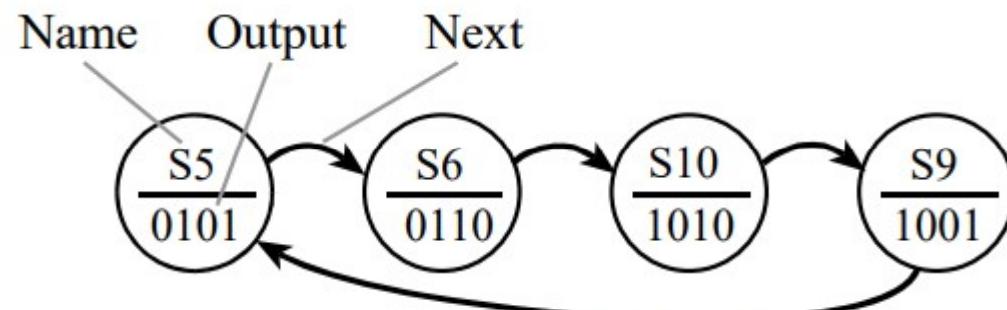
Stepper Motor



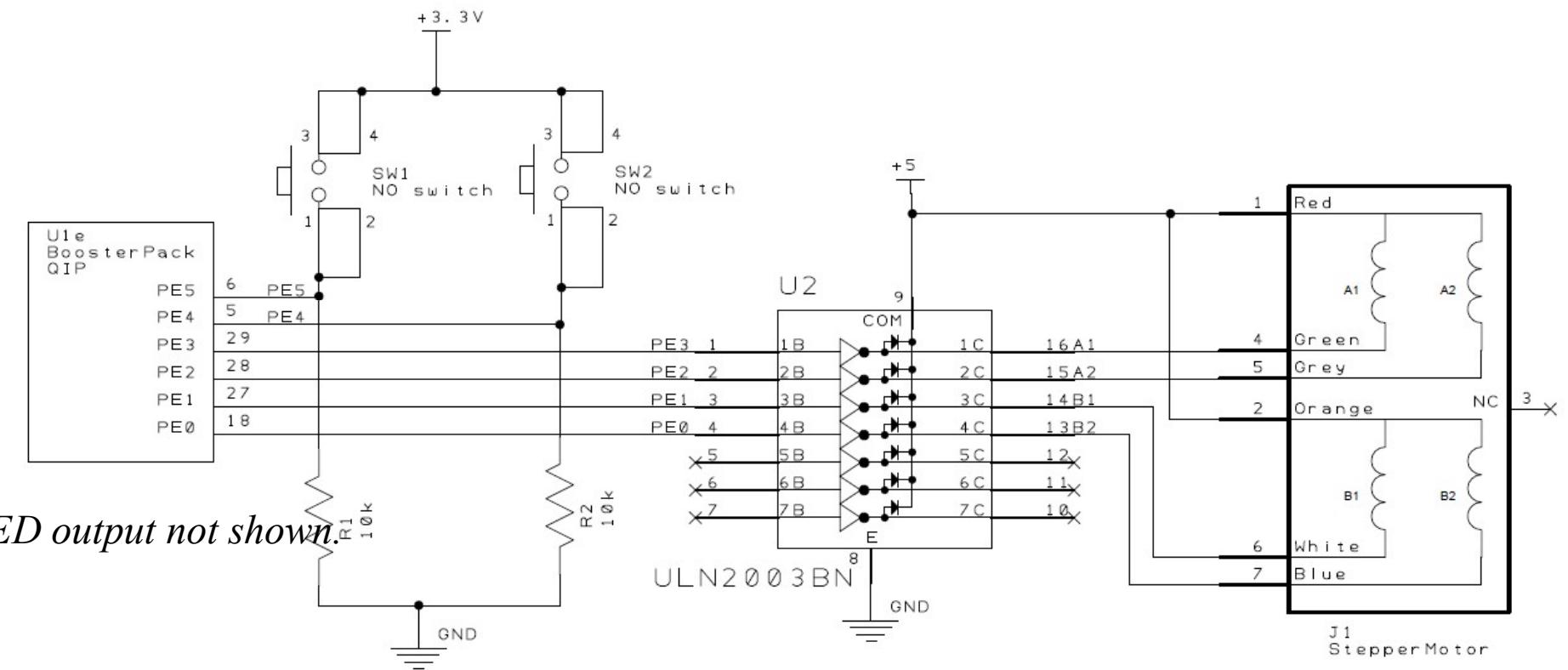
Stepper Motor



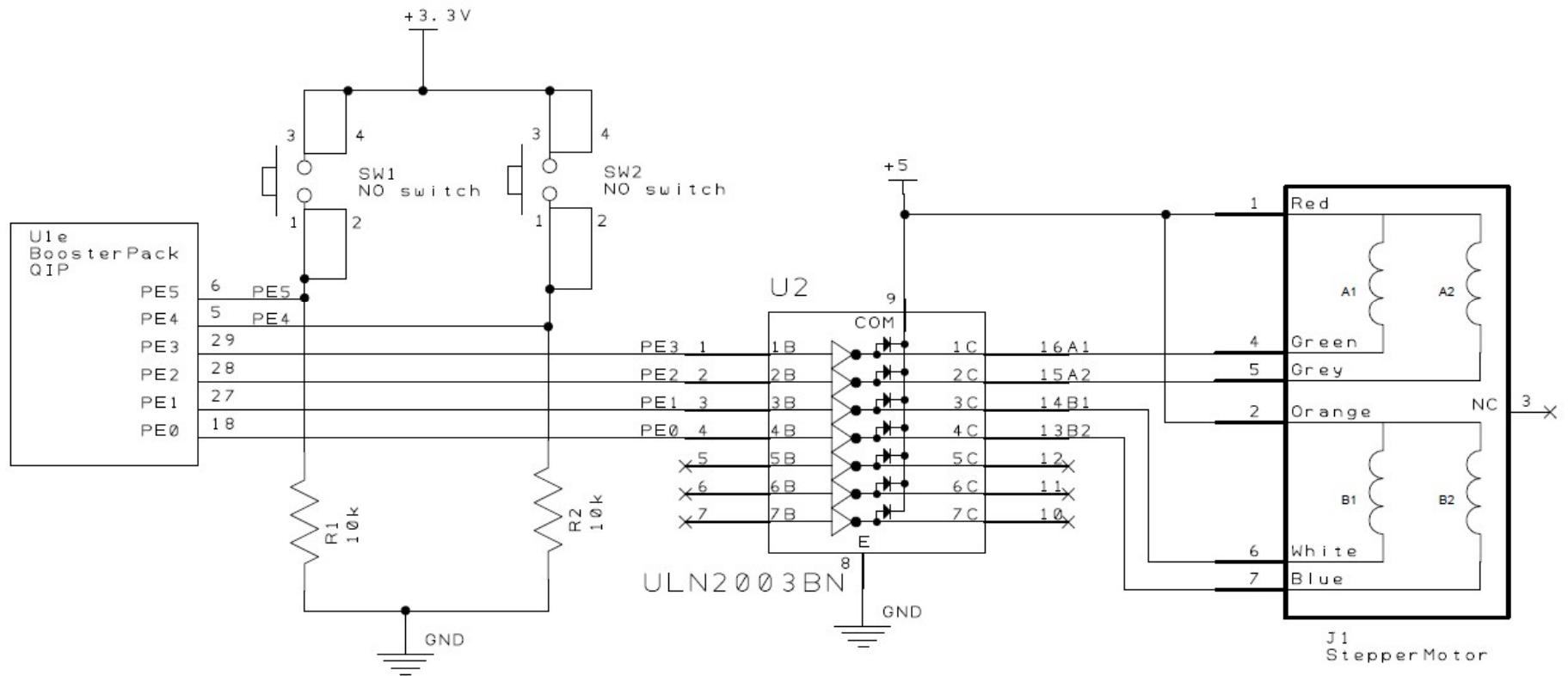
Stepper Motor FSM



Stepper Motor Interface



Stepper Motor Interface



Clockwise 5,6,10,9,5,6,10,9,5,6,10,9,5,6,10,9,5,6,10,9,5,6,10,9,5,6,10,9,...

36 steps/revolution means each step changes angle by 10 degrees

Analog Watch Example

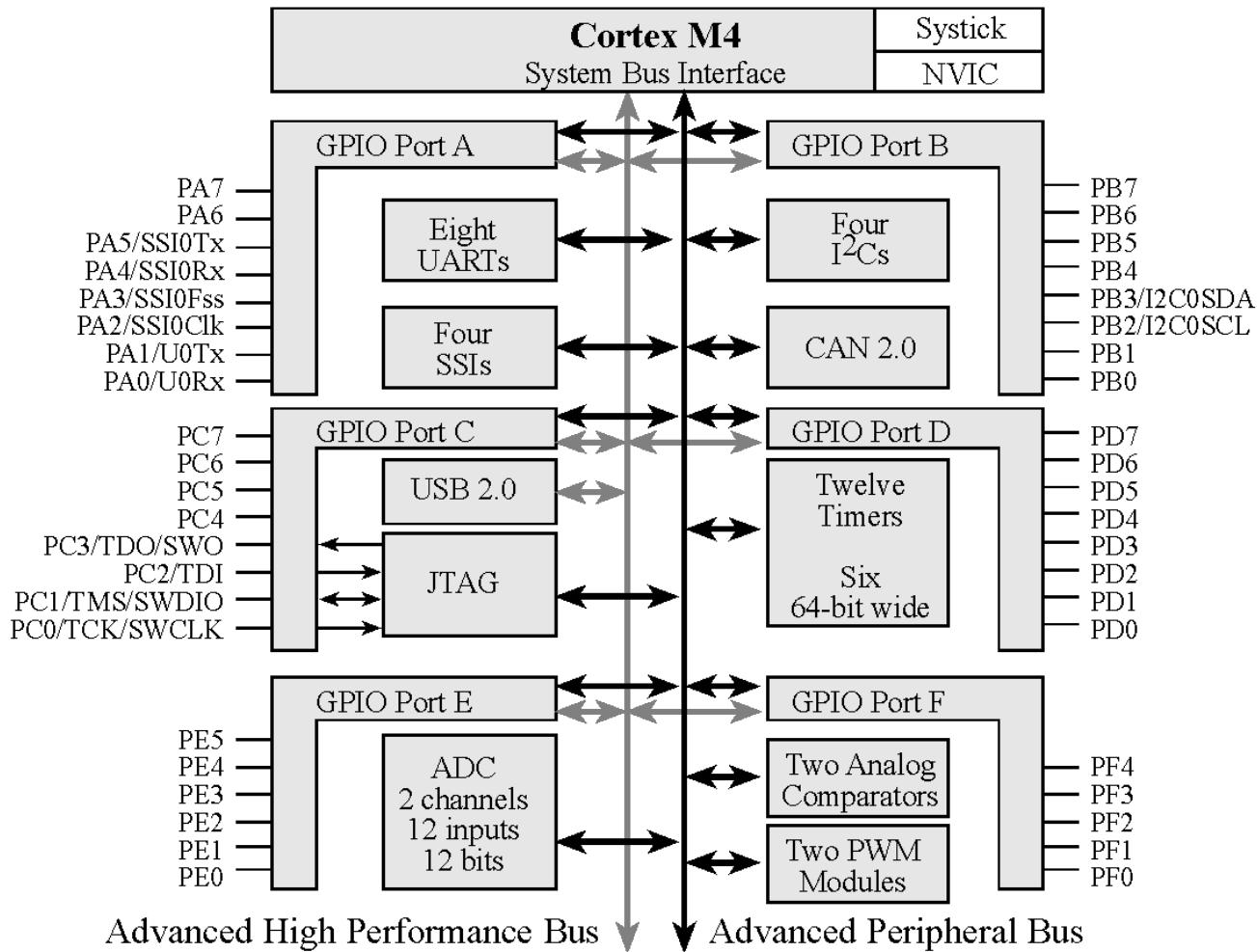
```
void SysTick_Wait(uint32_t delay){
    NVIC_ST_RELOAD_R = delay -1;
    NVIC_ST_CURRENT_R =0;
    while ((NVIC_ST_CTRL_R & 0x00010000) ==0){}
}

void SysTick_wait10ms(uint32_t delay){
    uint32_t i;
    for(i=0; i<delay;i++){ SysTick_Wait(800000);}
};

#define STEPPER (*((volatile unit32_t *) 0x4000703C))

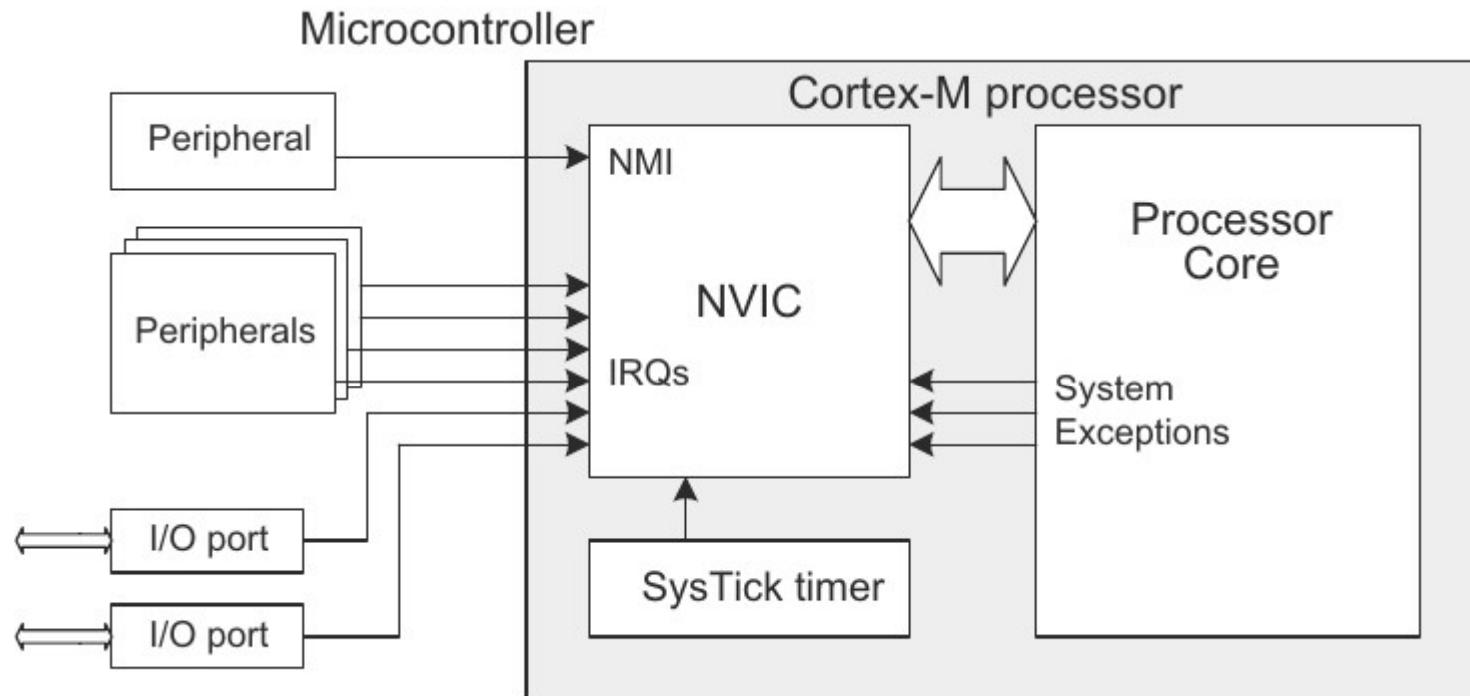
int main(void)
{
    int i,j;
    SYSCTL_RCGCGPIO_R |=0x08;
    GPIO_PORTD_DIR_R |=0xF;
    GPIO_PORTD_DEN_R |=0xF;
    while(1){
        STEPPER = 10;
        SysTick_wait10ms(5);
        STEPPER = 9;
        SysTick_wait10ms(5);
        STEPPER = 5;
        SysTick_wait10ms(5);
        STEPPER = 6;
        SysTick_wait10ms(5);}
}
```

Texas Instruments TM4C123



ARM Cortex-M4
+ 256K Flash
+ 32K RAM
+ JTAG
+ 43 Ports
+ SysTick
+ ADC
+ UART

ARM Cortex-M Interrupts



ARM Cortex-M Interrupts

- ❑ Each interrupt source has a separate **arm** bit
 - ❖ Set for those devices from which it wishes to accept interrupts,
`GPIO_PORTF_IM_R |= 0x10; // arm interrupt on PF4`
 - ❖ Deactivate in those devices from which interrupts are not allowed
- ❑ Each interrupt source has a separate **flag** bit
 - ❖ hardware sets the flag when it wishes to request an interrupt
`GPIO_PORTF_ICR_R = 0x10; // acknowledge flag4`
 - ❖ software clears the flag in ISR to signify it is processing the request
- ❑ Interrupt **enable** conditions in processor
 - ❖ Global interrupt enable bit, I, in PRIMASK register

```
EnableInterrupts();
```

Interrupt Processing

1. The execution of the main program is suspended
 1. the current instruction is finished,
 2. suspend execution and push 8 registers (R0-R3, R12, LR, PC, PSR) on the stack
 3. IPSR set to interrupt number
 4. sets PC to ISR address
2. The interrupt service routine (ISR) is executed
 - ❖ clears the flag that requested the interrupt
 - ❖ performs necessary operations
 - ❖ communicates using global variables
3. The main program is resumed when ISR executes **BX LR**
 - ❖ pulls the 8 registers from the stack

Interrupt Program Status Register (ISPR)

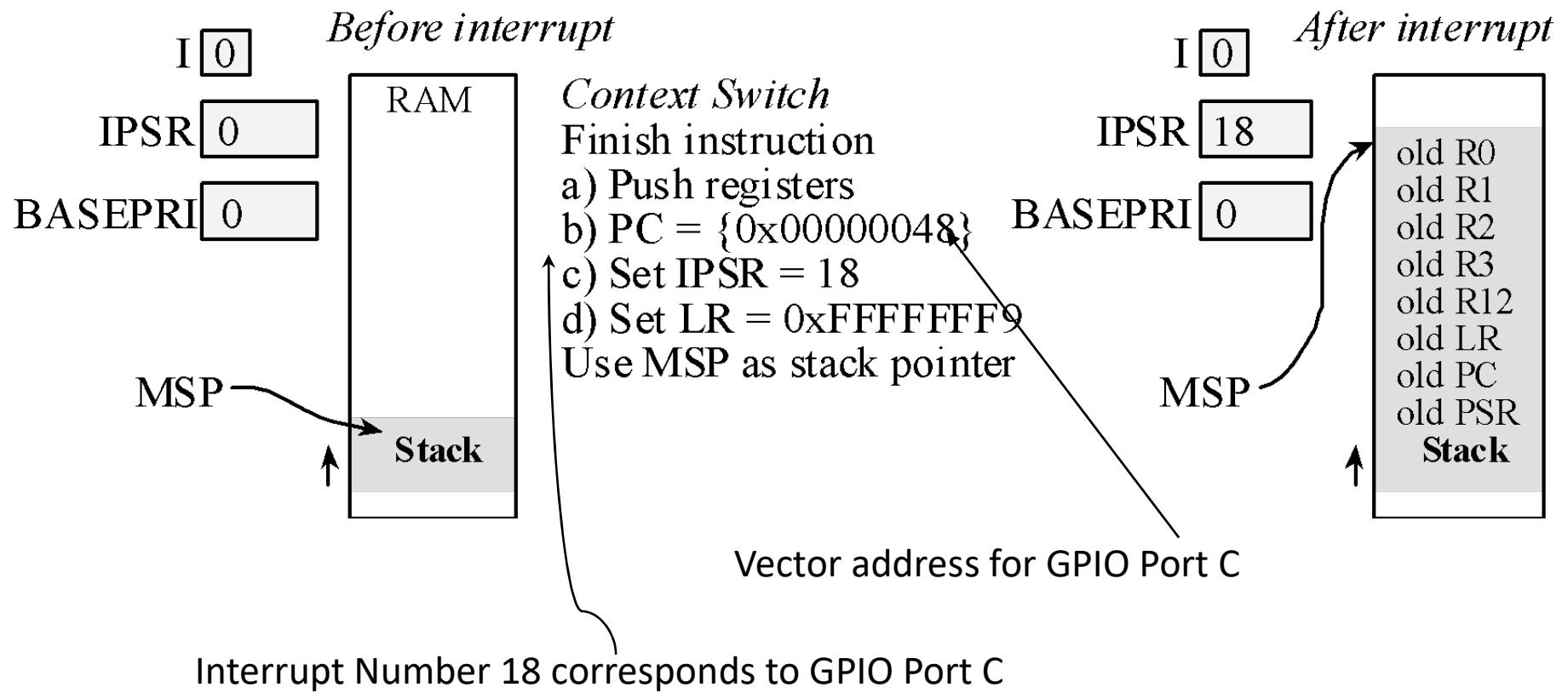
Bits	Description
Bits 31:9	Reserved
Bits 8:0	ISR_NUMBER: This is the number of the current exception: 0: Thread mode 1: Reserved 2: NMI 3: Hard fault 4: Memory management fault 5: Bus fault 6: Usage fault 7: Reserved 10: Reserved 11: SVCALL 12: Reserved for Debug 13: Reserved 14: PendSV 15: SysTick 16: IRQ0 ⁽¹⁾



The diagram illustrates the bit structure of the ISPR register. It shows a horizontal line with vertical dotted lines indicating bit boundaries. Bit 31 is at the far left and is labeled 'Reserved'. To its right are several dotted lines, followed by bits 9 and 8, which are both labeled 'ISR NUMBER'. The bits are numbered from 31 down to 0.

Figure 2-3, The ISPR Register.

Interrupt Context Switch



Interrupt Vectors

Vector address	Number	IRQ	ISR name in Startup.s	NVIC	Priority bits
0x00000038	14	-2	PendSV_Handler	NVIC_SYS_PRI3_R	23 - 21
0x0000003C	15	-1	SysTick_Handler	NVIC_SYS_PRI3_R	31 - 29
0x00000040	16	0	GPIOPortA_Handler	NVIC_PRI0_R	7 - 5
0x00000044	17	1	GPIOPortB_Handler	NVIC_PRI0_R	15 - 13
0x00000048	18	2	GPIOPortC_Handler	NVIC_PRI0_R	23 - 21
0x0000004C	19	3	GPIOPortD_Handler	NVIC_PRI0_R	31 - 29
0x00000050	20	4	GPIOPortE_Handler	NVIC_PRI1_R	7 - 5
0x00000054	21	5	UART0_Handler	NVIC_PRI1_R	15 - 13
0x00000058	22	6	UART1_Handler	NVIC_PRI1_R	23 - 21
0x0000005C	23	7	SSIO_Handler	NVIC_PRI1_R	31 - 29
0x00000060	24	8	I2C0_Handler	NVIC_PRI2_R	7 - 5
0x00000064	25	9	PWM0Fault_Handler	NVIC_PRI2_R	15 - 13
0x00000068	26	10	PWM0_Handler	NVIC_PRI2_R	23 - 21
0x0000006C	27	11	PWM1_Handler	NVIC_PRI2_R	31 - 29
0x00000070	28	12	PWM2_Handler	NVIC_PRI3_R	7 - 5
0x00000074	29	13	Quadrature0_Handler	NVIC_PRI3_R	15 - 13
0x00000078	30	14	ADC0_Handler	NVIC_PRI3_R	23 - 21
0x0000007C	31	15	ADC1_Handler	NVIC_PRI3_R	31 - 29
0x00000080	32	16	ADC2_Handler	NVIC_PRI4_R	7 - 5
0x00000084	33	17	ADC3_Handler	NVIC_PRI4_R	15 - 13
0x00000088	34	18	WDT_Handler	NVIC_PRI4_R	23 - 21
0x0000008C	35	19	Timer0A_Handler	NVIC_PRI4_R	31 - 29
0x00000090	36	20	Timer0B_Handler	NVIC_PRI5_R	7 - 5
0x00000094	37	21	Timer1A_Handler	NVIC_PRI5_R	15 - 13
0x00000098	38	22	Timer1B_Handler	NVIC_PRI5_R	23 - 21
0x0000009C	39	23	Timer2A_Handler	NVIC_PRI5_R	31 - 29
0x000000A0	40	24	Timer2B_Handler	NVIC_PRI6_R	7 - 5
0x000000A4	41	25	Comp0_Handler	NVIC_PRI6_R	15 - 13
0x000000A8	42	26	Comp1_Handler	NVIC_PRI6_R	23 - 21
0x000000AC	43	27	Comp2_Handler	NVIC_PRI6_R	31 - 29
0x000000B0	44	28	SysCtl_Handler	NVIC_PRI7_R	7 - 5
0x000000B4	45	29	FlashCtl_Handler	NVIC_PRI7_R	15 - 13

Interrupt Vectors

Vector address	Number	IRQ	ISR	NVIC	Priority bit
0x000000B8	46	30	GPIOPortFHandler	NVIC_PRI7_R	23-21

Priority registers on the NVIC

Address	31 – 29	23 – 21	15 – 13	7 – 5	Name
0xE000E400	GPIO Port D	GPIO Port C	GPIO Port B	GPIO Port A	NVIC_PRI0_R
0xE000E404	SSI0, Rx Tx	UART1, Rx Tx	UART0, Rx Tx	GPIO Port E	NVIC_PRI1_R
0xE000E408	PWM Gen 1	PWM Gen 0	PWM Fault	I2C0	NVIC_PRI2_R
0xE000E40C	ADC Seq 1	ADC Seq 0	Quad Encoder	PWM Gen 2	NVIC_PRI3_R
0xE000E410	Timer 0A	Watchdog	ADC Seq 3	ADC Seq 2	NVIC_PRI4_R
0xE000E414	Timer 2A	Timer 1B	Timer 1A	Timer 0B	NVIC_PRI5_R
0xE000E418	Comp 2	Comp 1	Comp 0	Timer 2B	NVIC_PRI6_R
0xE000E41C	GPIO Port G	GPIO Port F	Flash Control	System Control	NVIC_PRI7_R
0xE000E420	Timer 3A	SSI1, Rx Tx	UART2, Rx Tx	GPIO Port H	NVIC_PRI8_R
0xE000E424	CAN0	Quad Encoder 1	I2C1	Timer 3B	NVIC_PRI9_R
0xE000E428	Hibernate	Ethernet	CAN2	CAN1	NVIC_PRI10_R
0xE000E42C	uDMA Error	uDMA Soft Tfr	PWM Gen 3	USB0	NVIC_PRI11_R
0xE000ED20	SysTick	PendSV	--	Debug	NVIC_SYS_PRI3_R

NVIC enable registers

Address	31	30	29- 7	6	5	4	3	2	1	0	Name
0xE000E100	G	F	...	UART1	UART0	E	D	C	B	A	NVIC EN0_R
0xE000E104			...						UART2	H	NVIC EN1_R

PortF Interrupt Initialization

```
void EdgeCounter_Init(void){  
    SYSCTL_RCGCGPIO_R |= 0x00000020; // activate port F  
    FallingEdges = 0;  
    GPIO_PORTF_DIR_R &= ~0x10; // make PF4 in (built-in button)  
    GPIO_PORTF_DEN_R |= 0x10; // enable digital I/O on PF4  
    GPIO_PORTF_PUR_R |= 0x10; // enable weak pull-up on PF4  
    GPIO_PORTF_IS_R &= ~0x10; // PF4 is edge-sensitive  
    GPIO_PORTFIBE_R &= ~0x10; // PF4 is not both edges  
    GPIO_PORTFIEV_R &= ~0x10; // PF4 falling edge event  
    GPIO_PORTF_ICR_R = 0x10; // clear flag4  
    GPIO_PORTF_IM_R |= 0x10; // arm interrupt on PF4  
    NVIC_PRI7_R = (NVIC_PRI7_R&0xFF00FFFF)|0x00A00000; // priority 5  
    NVIC_EN0_R = 0x40000000; // enable interrupt 30 in NVIC  
    EnableInterrupts(); // Enable global Interrupt flag (I)  
}
```

PortF Handler

```
void GPIOPortF_Handler(void){  
    GPIO_PORTF_ICR_R = 0x10; // acknowledge flag4  
  
    FallingEdges = FallingEdges + 1;  
}
```