

# Chapter 3 - CPUs

---

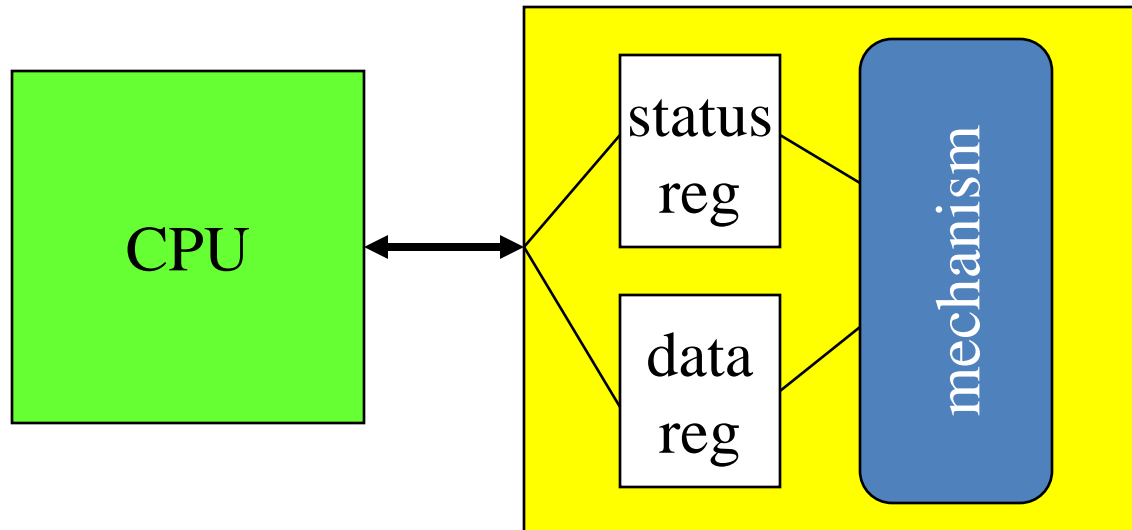
- ▶ Beyond the instruction set:
  - ▶ Input and output.
    - ▶ Busy-Wait and Interrupt-Driven
  - ▶ Supervisor mode, exceptions, traps.
  - ▶ Input/output on the LPC2292
    - ▶ I/O ports
    - ▶ Busy-wait programming
    - ▶ Interrupt processing
    - ▶ UART



# I/O devices

---

- ▶ Often includes some non-digital component.
- ▶ Typical digital interface to CPU:



# Example: LPC2292 UART

---

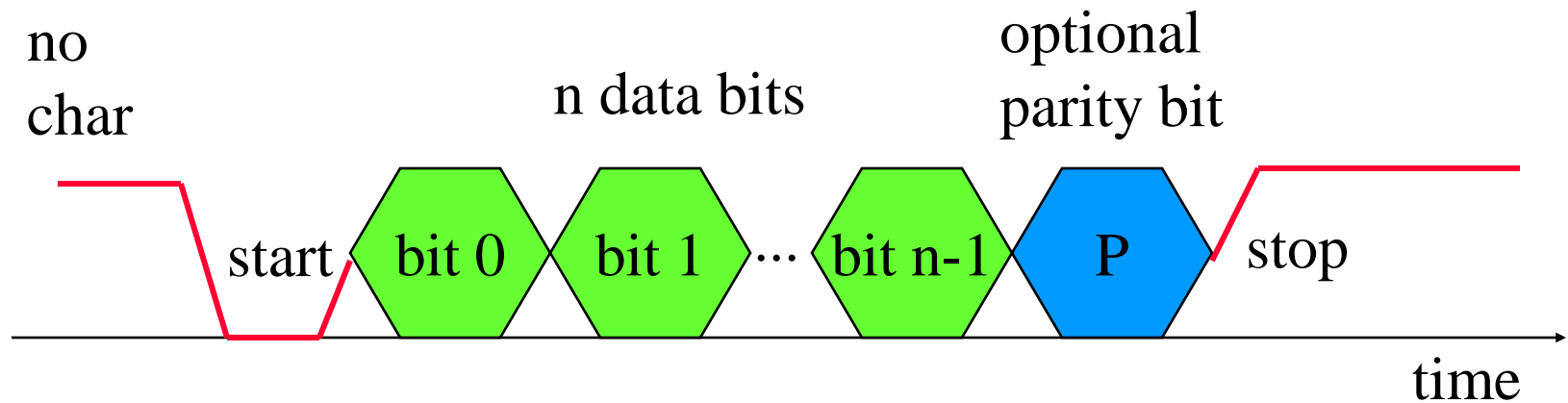
- ▶ **Universal asynchronous receiver transmitter (UART)** : provides serial communication.
- ▶ UARTs are integrated into most microcontrollers
  - ▶ Two UART modules on LPC2292
- ▶ Allows many communication parameters to be programmed.



# Asynchronous serial communication

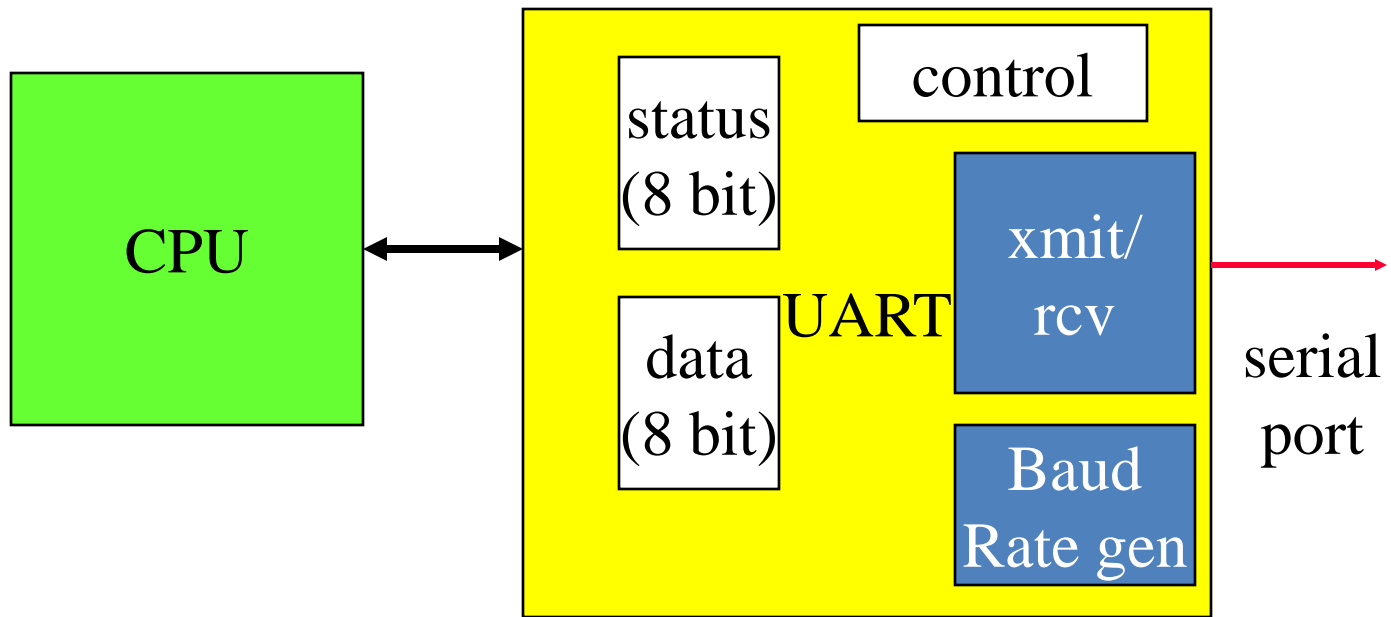
---

- ▶ Characters are transmitted separately:



# LPC2292 UART CPU interface

---



# Serial communication parameters set via UART “Line Control Register”

---

- ▶ Number of bits per character (5,6,7,8 bits).
- ▶ Enable/disable parity generation/checking.
- ▶ Type of parity bit: Even, Odd, Stuck-0, Stuck-1.
- ▶ Length of stop bit (1, 2 bits).



# UART status register

---

- ▶ Receiver data ready
  - ▶ Newly-received data in RBR
- ▶ Transmitter Holding empty
  - ▶ THR ready to accept new data
- ▶ Transmitter Empty
  - ▶ All data has been transmitted
- ▶ FE, OE, PE – framing/overrun/parity error in received data



# Programming I/O

---

- ▶ Two types of instructions can support I/O:
  - ▶ special-purpose I/O instructions;
  - ▶ memory-mapped load/store instructions.
- ▶ Intel x86 provides `in`, `out` instructions (“isolated I/O”).
- ▶ Most other CPUs use memory-mapped I/O.
- ▶ I/O instructions do not preclude memory-mapped I/O.



# 68HC11A8 – Memory address space (Von Neumann)

---

- ▶ All addresses mapped into a 64K-byte memory address space
  - ▶ RAM: 0000-00FF (can re-map to 4K page)
  - ▶ I/O Registers: I000-I03F (can re-map)
  - ▶ EEPROM: B600-B7FF
  - ▶ BOOT ROM: BF40-BFFF
  - ▶ USER ROM: E000-FFFF



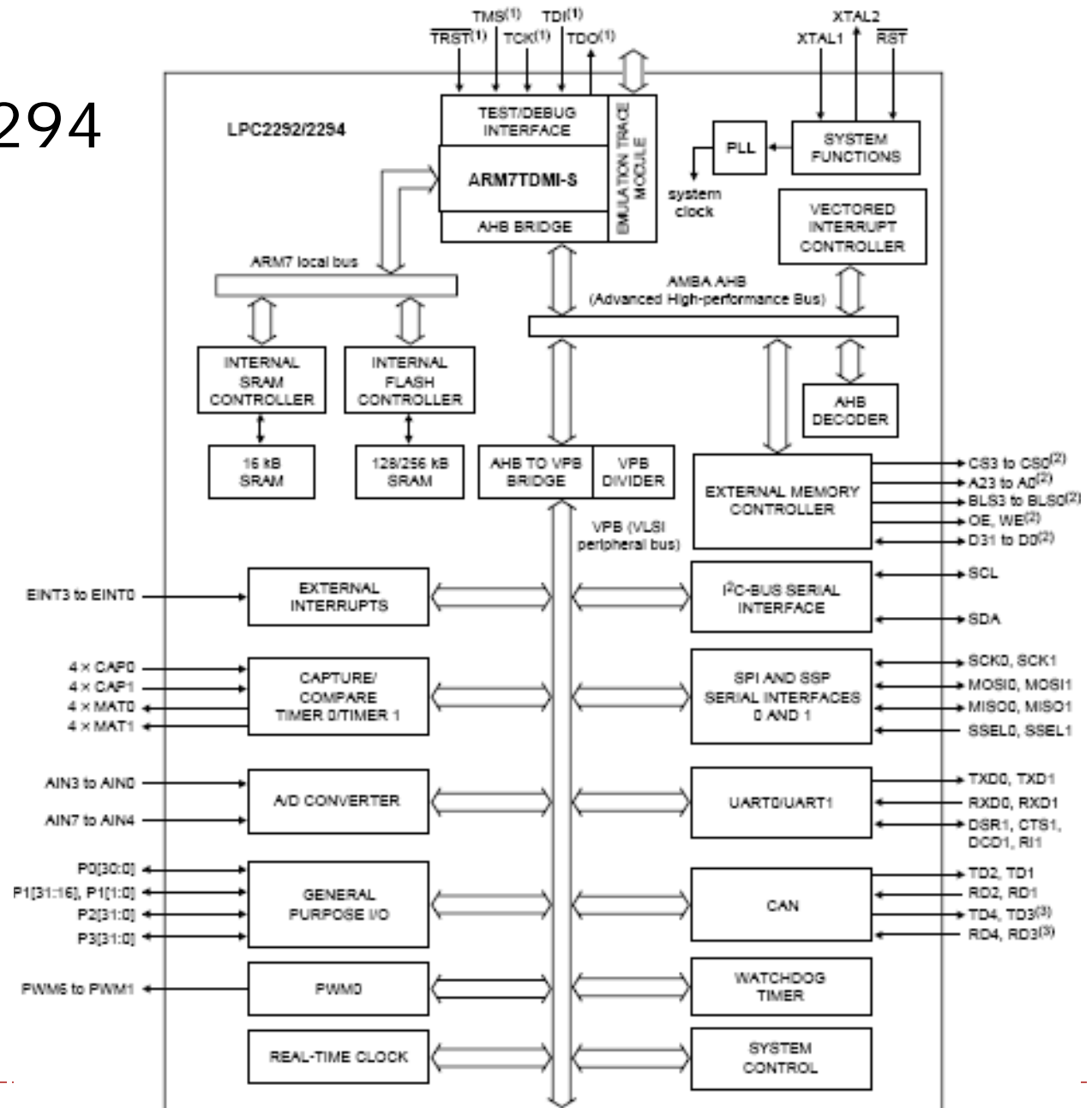
# Intel 8051 On-chip address spaces (Harvard architecture)

---

- ▶ Program storage: 0000-0FFF
- ▶ Data address space:
  - ▶ RAM: 00-7F
    - ▶ low 32 bytes in 4 banks of 8 registers R0-R7
  - ▶ Special function registers: 80-FF
    - ▶ includes “ports” P0-P3
- ▶ Special I/O instructions for ports P0-P3



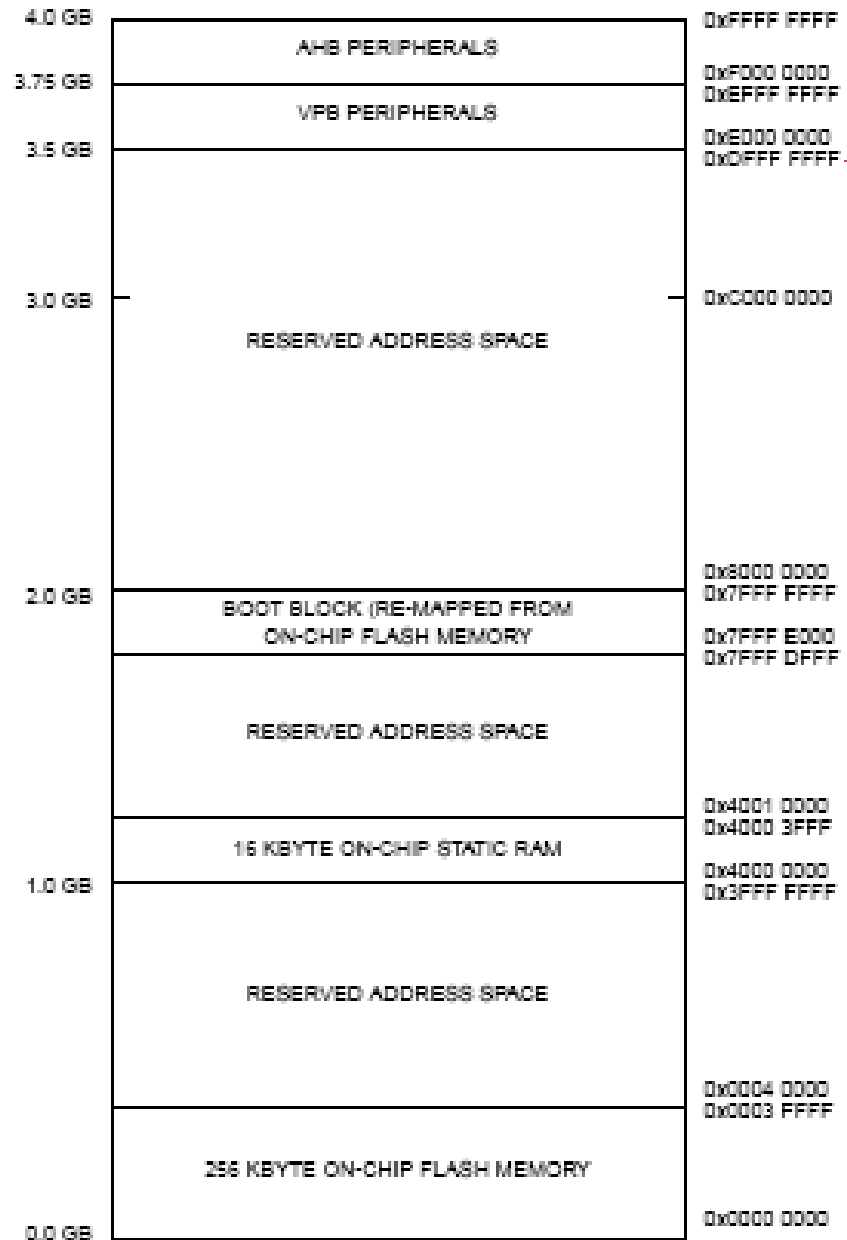
# NXP/Philips LPC2292/2294



# LPC2292/2294 Memory Map

AHB = Advanced,  
High-performance  
Bus

VPB = VLSI  
Peripheral Bus



# LPC2292

## Pin Configuration

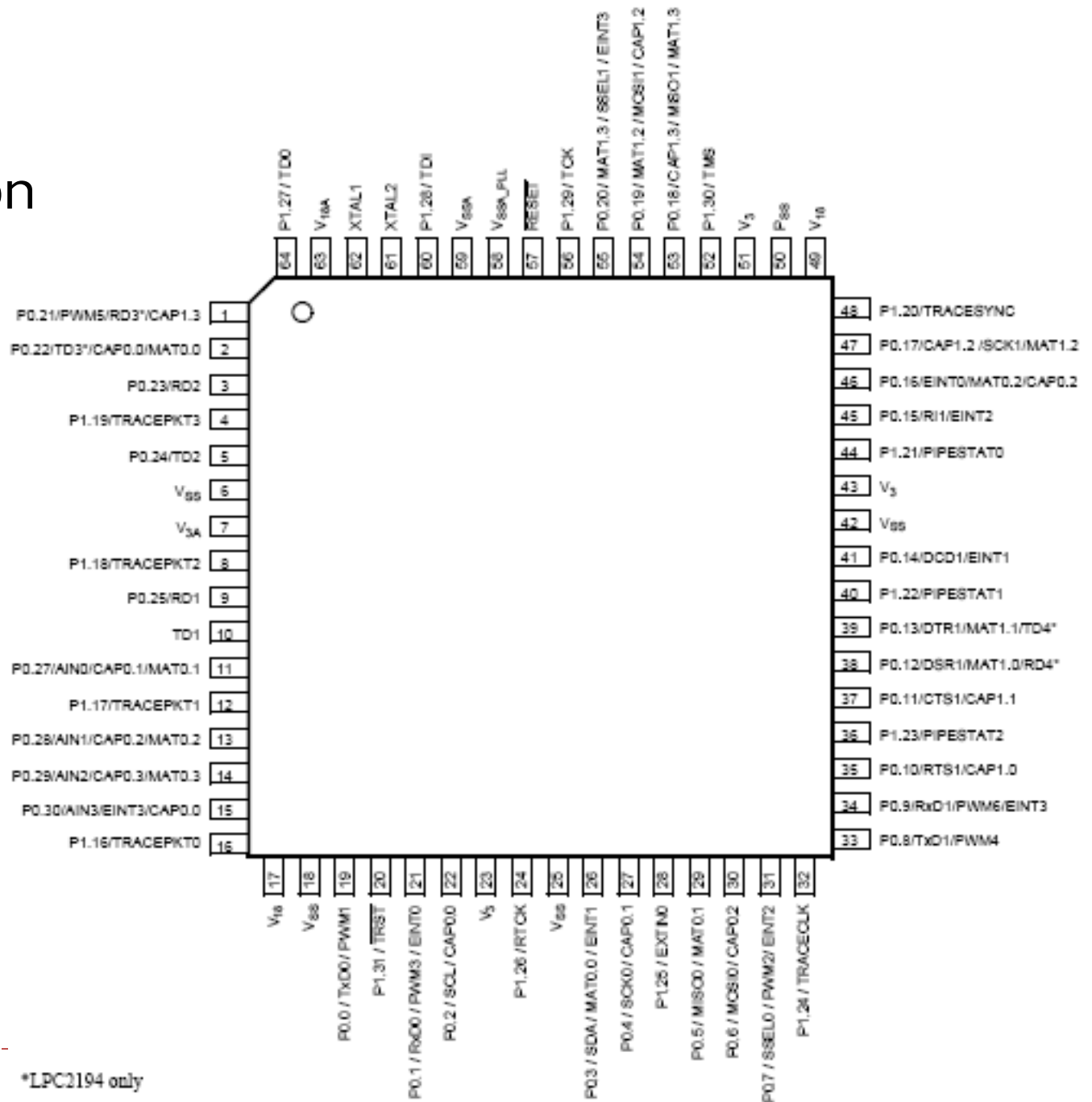
[GPIO pins](#)

P0.0-P0.31

P1.16-P1.31

P2.0-P2.31

P3.0-P3.21



\*LPC2194 only

# ARM memory-mapped I/O

---

- ▶ **Hardwired address for device:**

```
DEV1 EQU 0x1000
```

- ▶ **Read/write code:**

```
LDR r1,=DEV1    ; set up device address
LDR r0,[r1]     ; read DEV1
MOV r0,#8       ; set up value to write
STR r0,[r1]     ; write value to device
```



# Kiel ARM C compiler – GPIO access

---

- ▶ “volatile” indicates data can change outside of the program  
(example - data applied to an external port)

```
/* text example */
volatile unsigned *port=(unsigned int *)0x100;
*port = value;           //write value to port
value = *port;          //read value from port

/* From lpc21xx.h - IO Port 1 definitions */
#define IOSET1 (*(volatile unsigned long *) 0xE0028014))
#define IODIR1 (*(volatile unsigned long *) 0xE0028018))
#define IOCLR1 (*(volatile unsigned long *) 0xE002801C))
/* From Blinky example */
IODIR1 = 0xFF0000;      /* P1.16..23 defined as Outputs */
IOSET1 = j;             /* Turn on LED */
IOCLR1 = j;             /* Turn off LED */
```



# LPC2292 GPIO register map

Generic Name	Description	Access	Reset Value	PORT0 Address & Name	PORT1 Address & Name	PORT2 Address & Name	PORT3 Address & Name
IOPIN	GPIO Port Pin value register. The current state of the GPIO configured port pins can always be read from this register, regardless of pin direction and mode.  Activity on non-GPIO configured pins will not be reflected in this register.	Read Only	NA	0xE0028000 IO0PIN	0xE0028010 IO1PIN	0xE0028020 IO2PIN	0xE0028030 IO3PIN
IOSET	GPIO Port Output set register. This register controls the state of output pins in conjunction with the IOCLR register. Writing ones produces highs at the corresponding port pins. Writing zeroes has no effect.	Read/Write	0x0000 0000	0xE0028004 IO0SET	0xE0028014 IO1SET	0xE0028024 IO2SET	0xE0028034 IO3SET
IODIR	GPIO Port Direction control register. This register individually controls the direction of each port pin.	Read/Write	0x0000 0000	0xE0028008 IO0DIR	0xE0028018 IO1DIR	0xE0028028 IO2DIR	0xE0028038 IO3DIR
IOCLR	GPIO Port Output clear register. This register controls the state of output pins. Writing ones produces lows at the corresponding port pins and clears the corresponding bits in the IOSET register. Writing zeroes has no effect.	Write Only	0x0000 0000	0xE002800C IO0CLR	0xE002801C IO1CLR	0xE002802C IO2CLR	0xE002803C IO3CLR



# Peek and poke

---

- ▶ Traditional HLL interfaces for memory and memory-mapped I/O (compiler-dependent):

```
/* return contents of memory location */  
int peek(char *location) {  
    return *location; }  
  
/* modify contents of memory location */  
void poke(char *location, char newval) {  
    (*location) = newval; }
```

(Not used in Keil ARM C compiler)

---





# Busy/wait output ("program-controlled")

---

- ▶ Simplest way to program device.
  - ▶ Instructions test for device ready.
  - ▶ `OUT_CHAR` and `OUT_STATUS` are device addresses

```
/* send a character string */  
current_char = mystring;  
while (*current_char != '\0') {  
    OUT_CHAR = *current_char;  
    while (OUT_STATUS != 0);  
    current_char++;  
}
```



# Busy/wait output (ARM assy.lang.)

---

```
;output character from r0
```

```
#define OUT_STATUS 0x1000
```

```
#define OUT_CHAR 0x1004
```

```
ldr r1,=OUT_STATUS ;point to status  
w ldrb r2,[r1] ;read status reg  
ands r2,r2,#1 ;check ready bit  
beq w ;repeat until 1  
ldr r2,=OUT_CHAR ;point to char  
strb r0,[r2] ;send char to reg
```



# Simultaneous busy/wait input and output

---

```
while (TRUE) {
    /* read */
    while (IN_STATUS == 0);
    achar = IN_DATA;
    /* write */
    OUT_DATA = achar;
    while (OUT_STATUS != 0);
}
```

## **NOTE:**

**Assumes all 8 bits of IN\_STATUS = 0**

---

