5) The Assembler: CS 241

6) Input/output

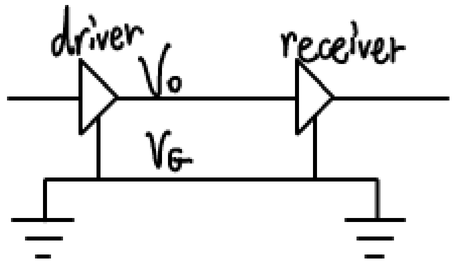
1) Physical Layer

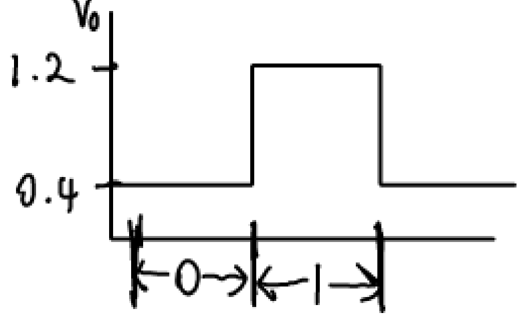
width: serial = 1 bit wide (e.g. SATA, USB)
parallel = n-bits wide (e.g. IDE, DVI)

timing: synchronous = with clock
asynchronous = without clock

direction: simplex = 1 way all the time
half-duplex = 1 way at a time
duplex = 2 ways all the time

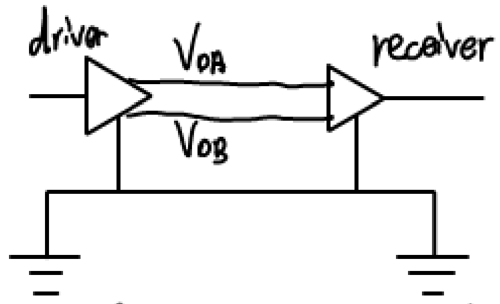
transmission: single-ended = 1 wire/trace per bit



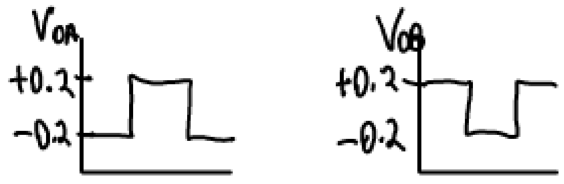


- receiver measures difference between V_o and V_G
- cheaper but susceptible to noise ⇒ slower (GTL: 120Mbps)

differential = 2 wires/traces per bit

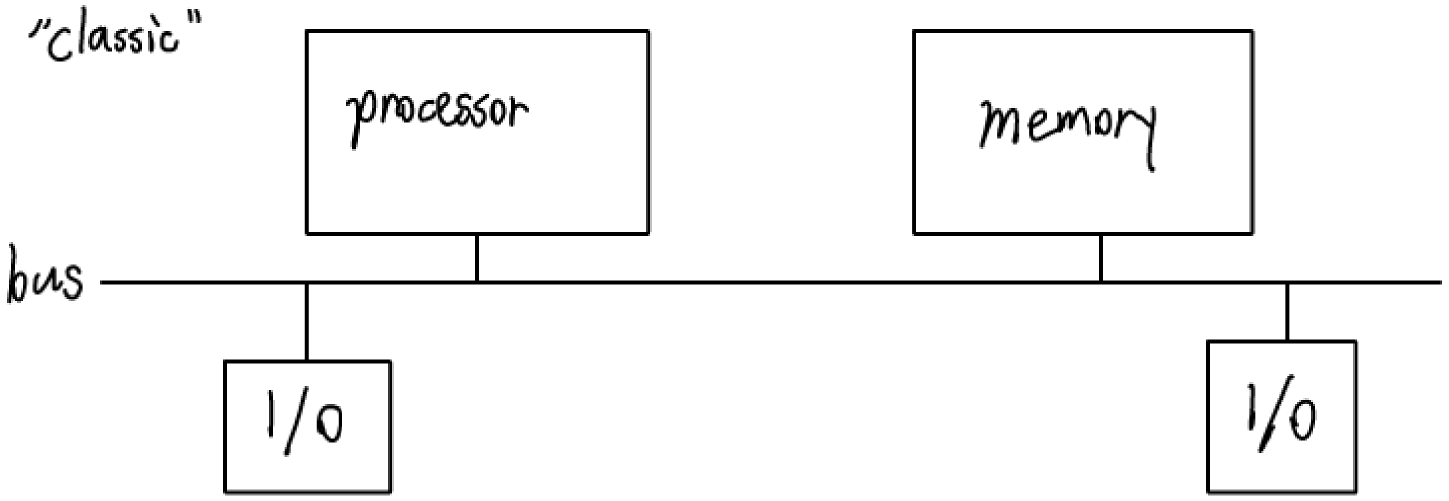


- receiver measures difference on the wire pair ($V_{OA} - V_{OB}$)
- high noise immunity ⇒ faster (LVDS: 2Gbps)

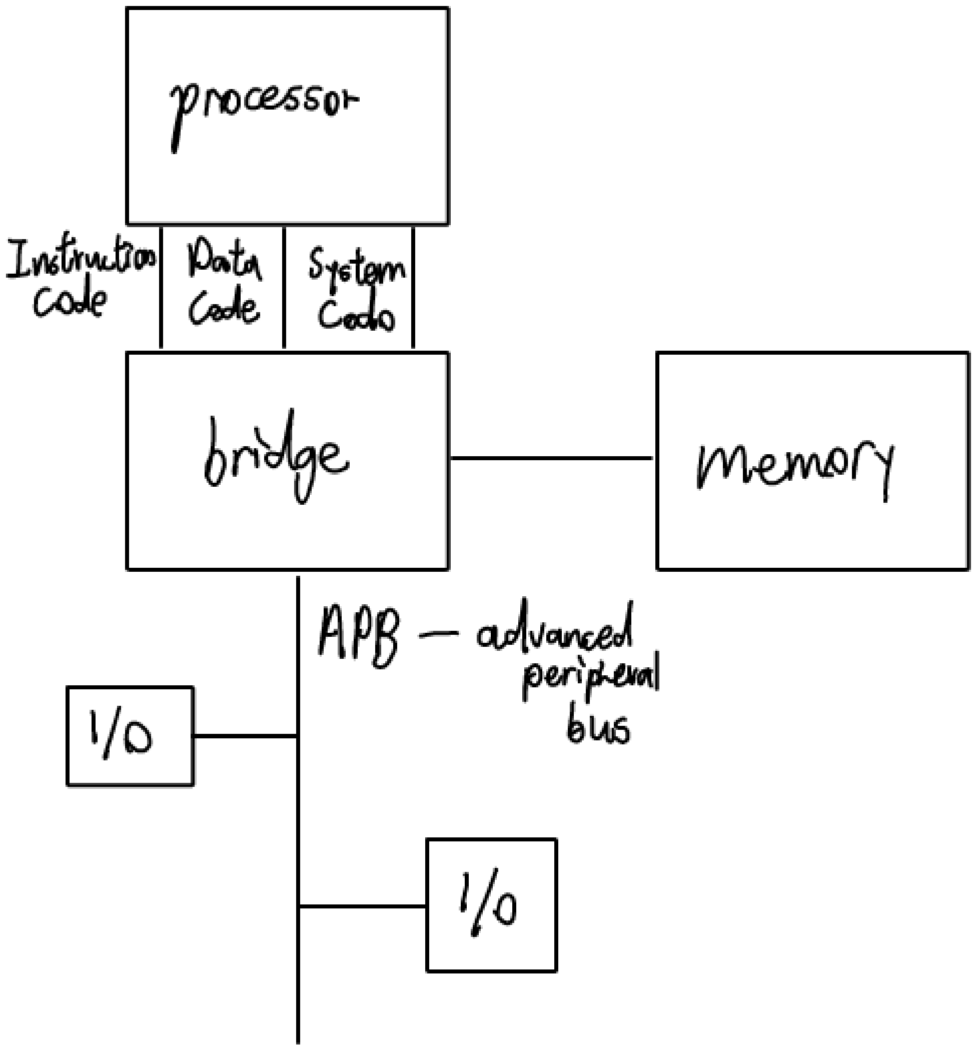


2) The System Bus

"classic"



Cortex-M3



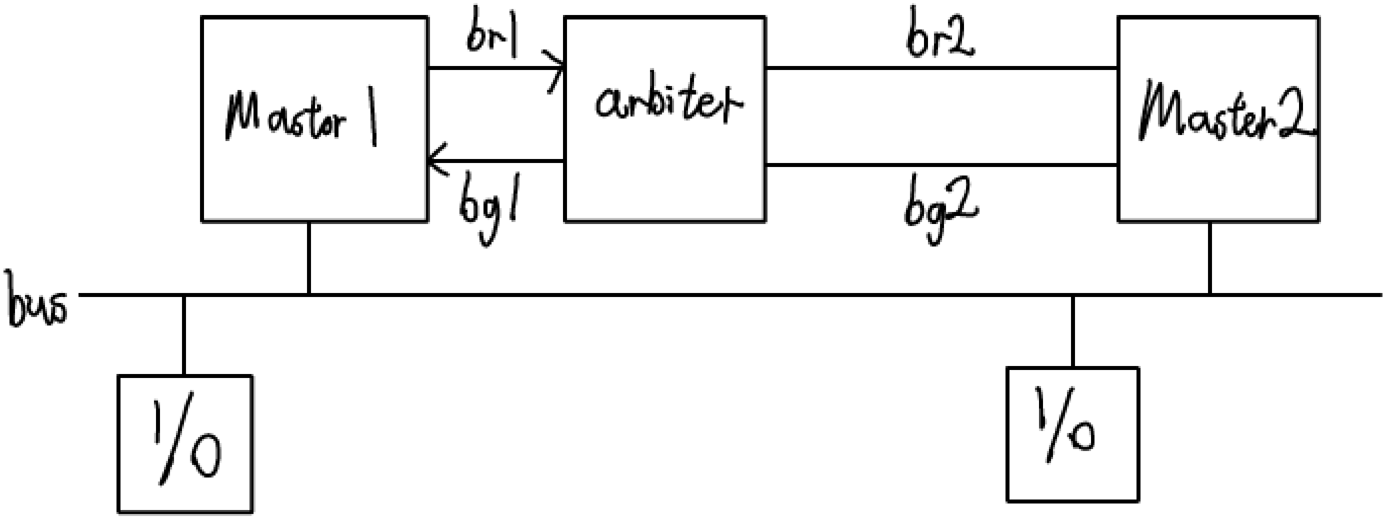
- bus master: controls bus and initiates transactions
- bus slave: responds to transactions
- bus write: master $\xrightarrow{\text{(data)}}$ slave
- bus read: master $\xleftarrow{\text{(data)}}$ slave

2.1) Bus Transfers

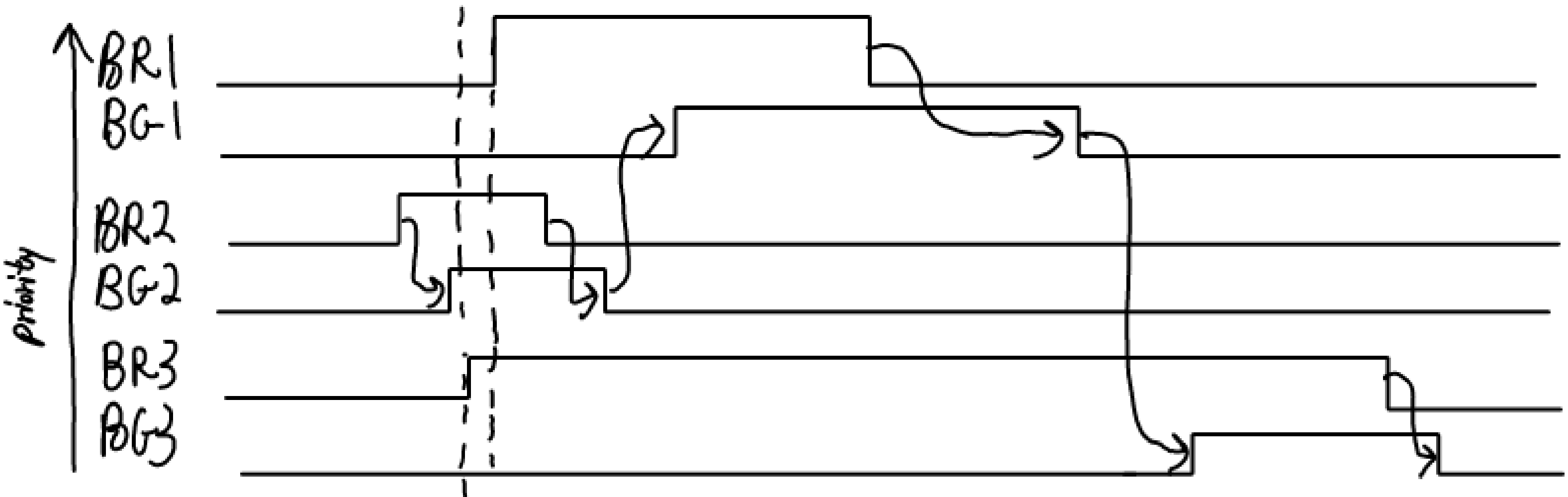
- see handout
- asynchronous transfers use a "hand shake" (Master Ready, Slave Ready) to time the transfer ($4 * t_{prop}$)
  - benefit: flexible (variable bus length)
- synchronous transfers are timed by common clock.
  - benefit: faster ($2 * t_{prop}$)

2.2) Arbitration

- The process of deciding which device becomes master
- centralized: an arbiter decides
- decentralized: the devices cooperate to decide
  e.g. centralized with priorities



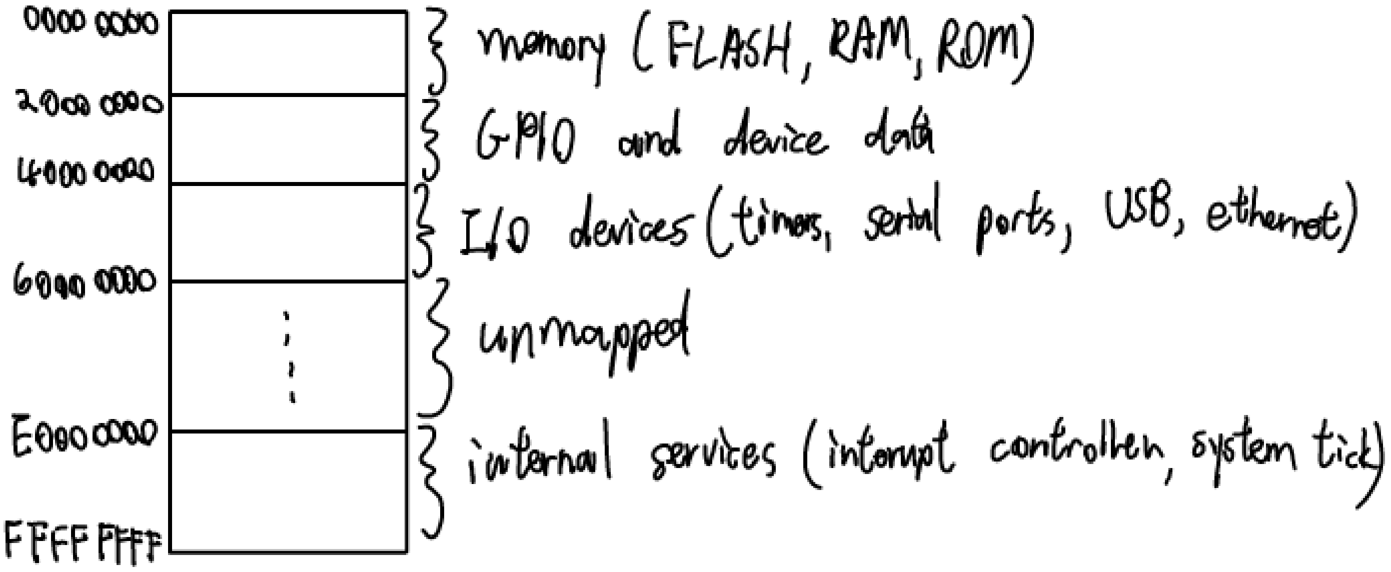
BR = bus request
BG = bus grant



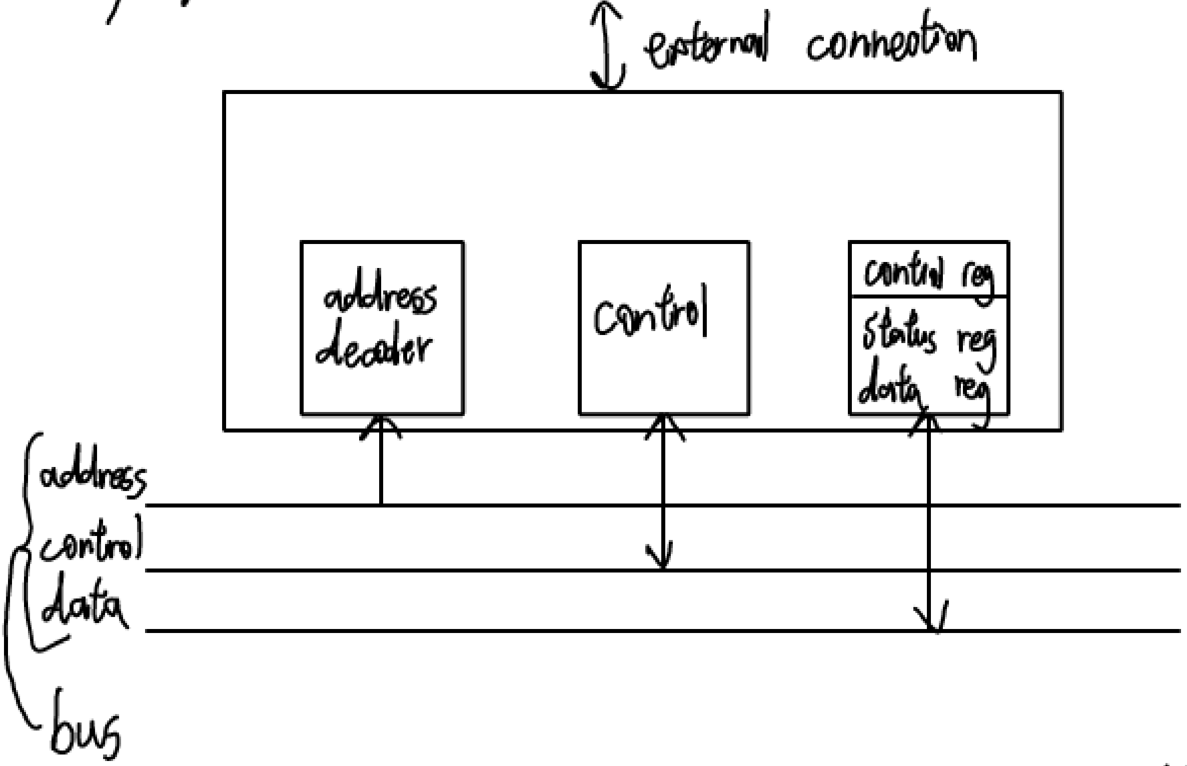
3) Memory - Mapped I/O

- bus addresses are used for memory and I/O devices.

e.g. LPC1768 memory map

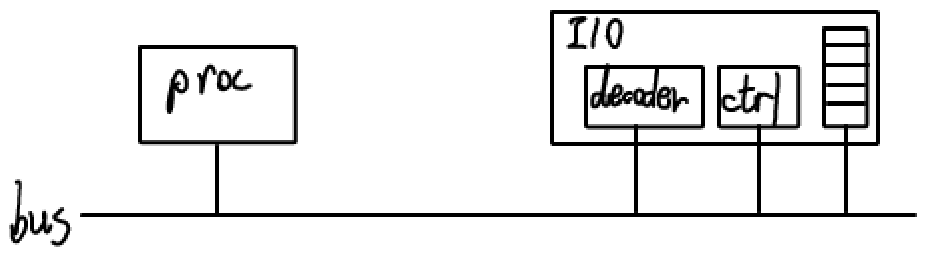


3.1) I/O Device Bus Interface



- device registers get mapped to specific addresses
- the control circuit responds to bus transactions

3.1) I/O device Bus I/f

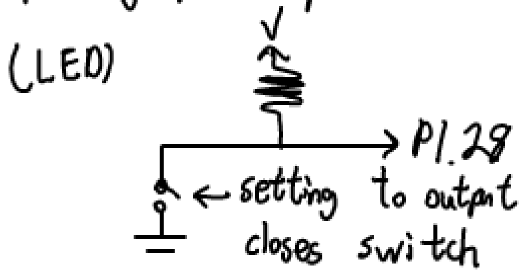


e.g. GPIO on LPC1768

- 5 ports of up to 32 pins each: P0, P1, P2, P3, P4
  P1.28, P1.29, P1.31, P2.2-6 are connected to LEDs

- FIO DIR : fast GPIO port direction control register
  FIO0DIR at 0x2009C000
  FIO1DIR at 0x2009C020
  FIO2DIR at 0x2009C040

- each bit controls the corresponding port pin
  0 = input
  1 = output
  FIO0DIR[0] = 1
  Port 0, pin0 is output



- turn off LEDs
  write 1's to the corresponding bits in the FIODIR regs.
  Part 1: bits 31, 29, 28

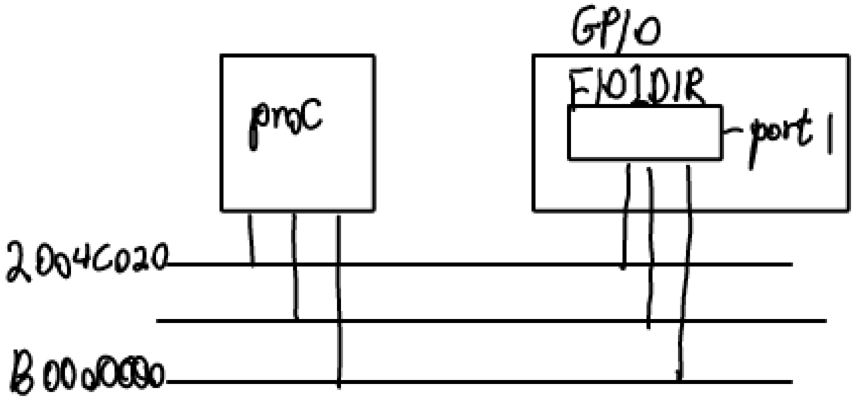
```
1  0  1  1  0000 ,--- 0000
31 30 29 28           3210
```

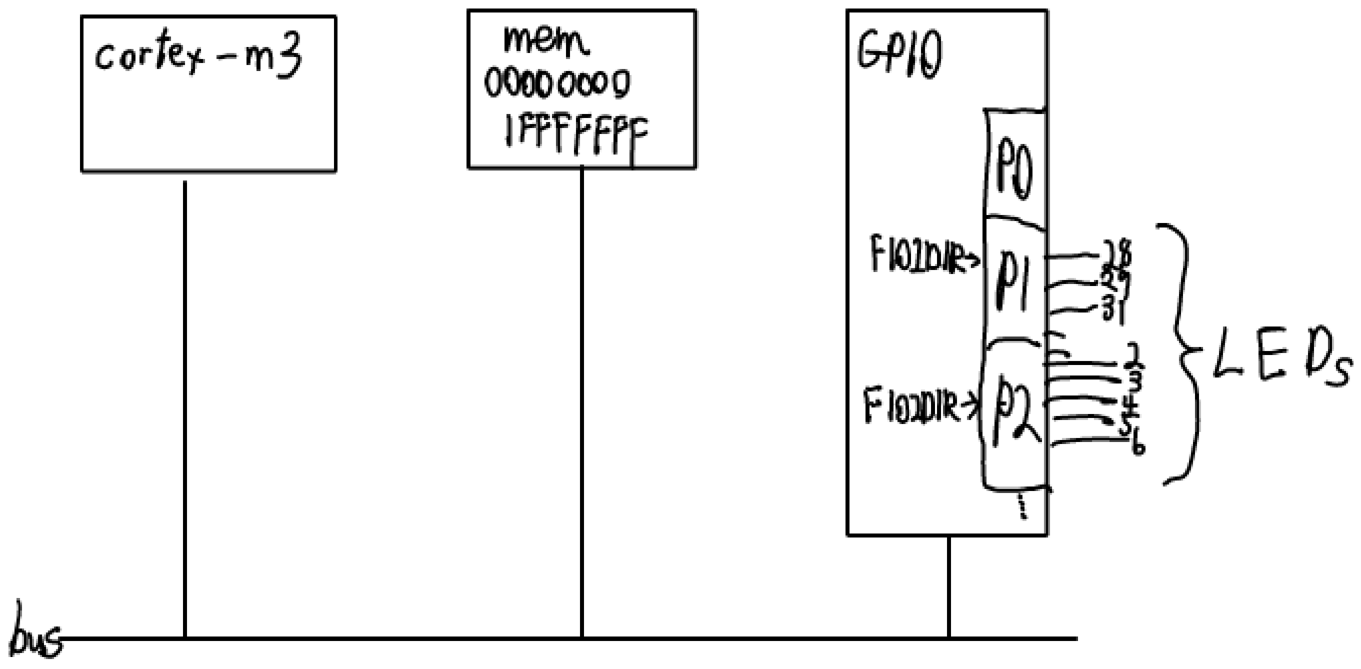
  = 0xB000 0000

  Part 2: bits 2,3,4,5,6

```
0000 ,--- 0111 1100 = 0x0000007C
31 30 29 28    7654 3210
```

```
LDR  r10, =0x2009000
MOV  r3, #0x B0000000
STR  r3, [r10, #0x20]
MOV  r3, #0x7c
STR  r3, [r10, #0x40]
```





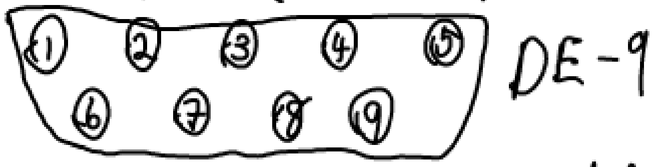
4) Polling I/O

- program receives/sends data thru an I/O device by reading/writing the device data registers
- must repeatedly check (poll) device status register until it is ready (with received data, or to send next data)

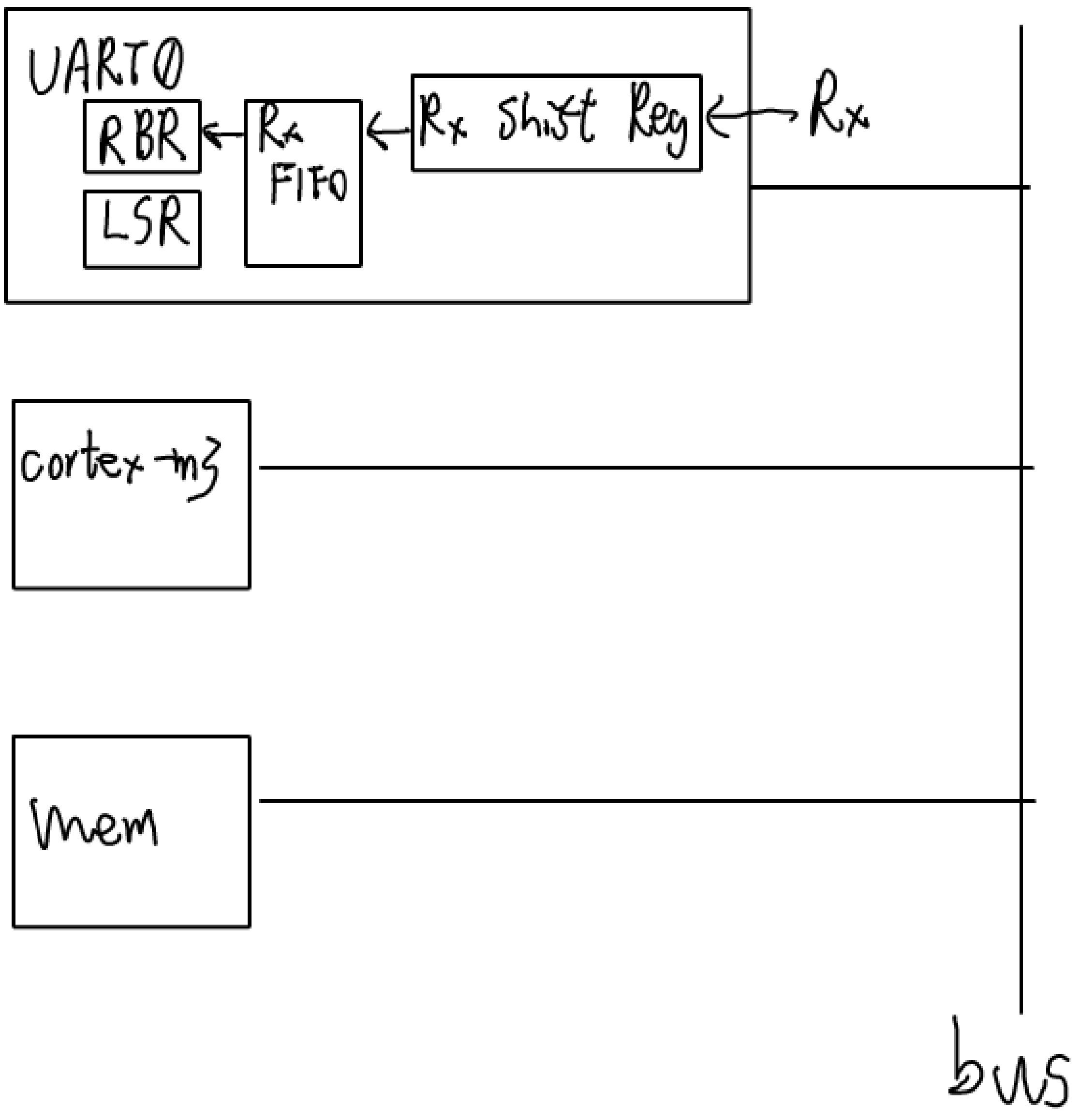
- pseudocode

```
1) read status register
2) check receive/transmit ready bit
        2.1) if not ready, goto 1)
3) read received data from data reg (buffer), or write
   data to data reg. (this changes device status)
4) process received data, or ready next transmit data
5) goto 1)
```

e.g. LPC1768 serial ports - 4: UART0, UART1, ...
- universal asynchronous receiver transmitter
- standart RS-232

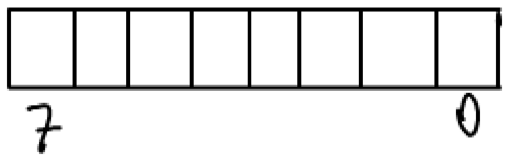


1	DCD	data carrier detect	
2	Rx	receive data	} full duplex
3	Tx	transmit data	
4	DTR	data terminal ready	
5	GND	ground	
6	DSR	data set ready	
7	RTS	ready to send	} handshake
8	CTS	clear to send	
9	RI	ring indicator	



- 10 registers including RBR, LSR

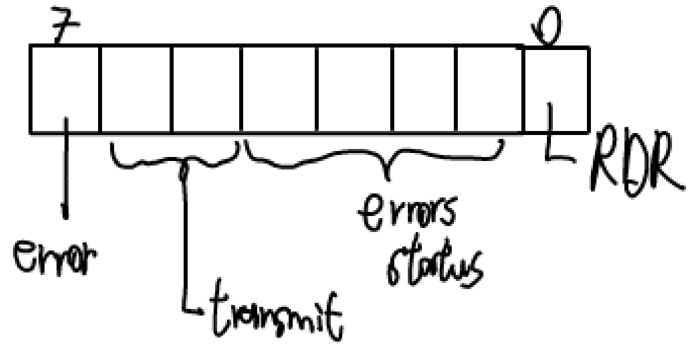
-RBR[7:0] receive buffer register



-holds oldest received byte
-reading pops from Rx FIFO

-U0RBR at 0x4000C000

-LSR[7:0] line status register



-receive data ready { 1=ready
0=not ready

-U0LSR at 0x4000C014

e.g. read data received on UART0 and place in memory

```
BUFFER    SPACE   256
UART0     EQU     0x4000C000
RBR_OFF   EQU     0x0
LSR_OFF   EQU     0x14
RDR_MASK  EQU     2_0000 0001
```

e.g.

```
                LDR   r0, =UART0                } setup
                LDR   r1, =BUFFER               }
POLL            LDRB  r2, [r0, #LSR_OFF]        }
                TST   r2, #RDR_MASK             } Polling
                BEQ   POLL                      }
                LDRB  r3, [r0, #RBR_OFF]        }
                STRB  r3, [r1], #1              } moves one byte
                B     POLL                      }
```