ECE243

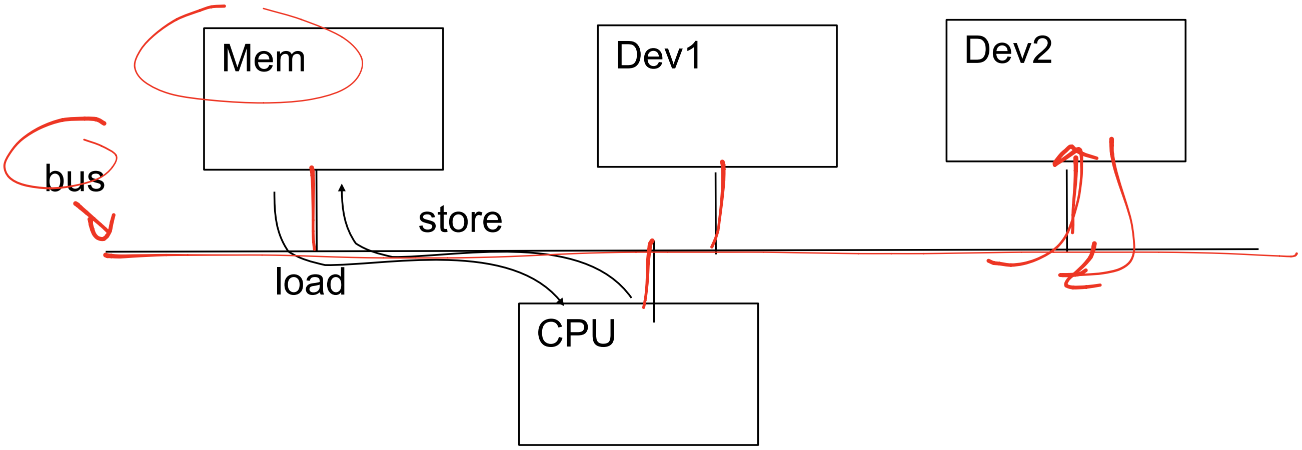
Input/Output (I/O) Software

Prof. Enright Jerger

Memory Mapped Devices

Connecting devices to a CPU

- memory is just a device
- CPU communicates with it
  - through loads and stores (addrs & data)
- memory responds to certain addresses
  - not usually all addresses
- CPU can talk with other devices too
  - using the same method: loads and stores
- devices will also respond to certain addrs
  - addrs reserved for each device



MEMORY MAPPED I/O

- a device:
  - ‘sits’ on the memory bus
  - watches for certain address(es)
  - responds like memory for those addresses
  - ‘real’ memory ignores those addresses
- the memory map:
  - map of which devices respond to which addrs

DESL NIOS SYSTEM MEM MAP

0x00000000: 8MB SDRAM (up to 0x007fffff)

0x10001000: JTAG UART

0x10001020: 7 segment display

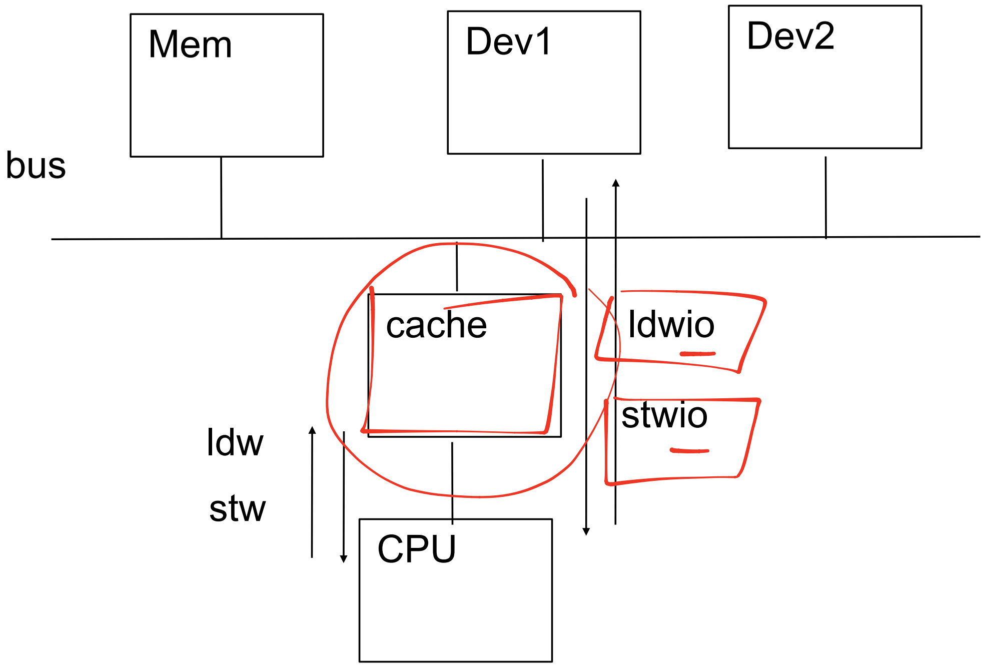
0x10000060: GPIO JP1

0x10000070: GPIO JP2

0x10003050: LCD display

- These are just a few example locations/devices
- see DESL website: NiosII: Reference: Device Address Map for full details

TALKING WITH DEVICES

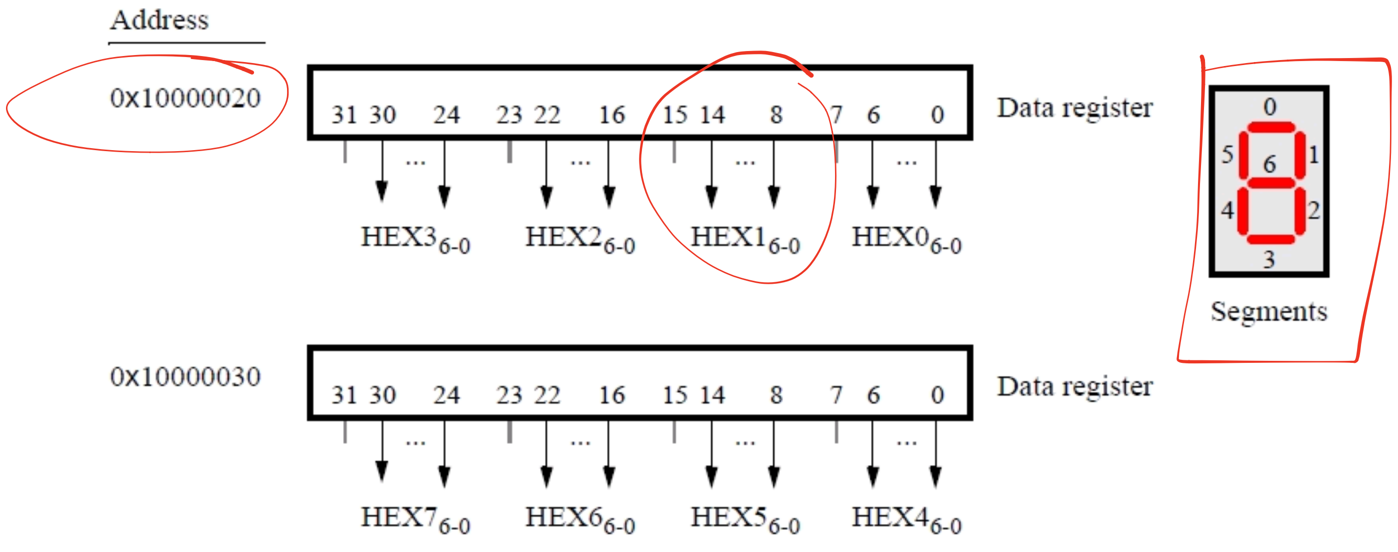


- Note1: use ldwio and stwio
  - to read/write memory mapped device locations
  - io means bypass cache if it exists (more later)
- Note2: use word size even if it is a byte location
  - potentially funny behaviour otherwise

see DESL website: NiosII: Devices for full specs on every device

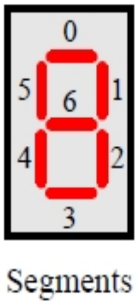
7-Segment Display

- **base: HEX3-Hex0:** 0x10000020
  HEX7-HEX4: 0x10000030
- Controls the individual ‘segments’ of the hex display
- write only
- handy for debugging,monitoring program status

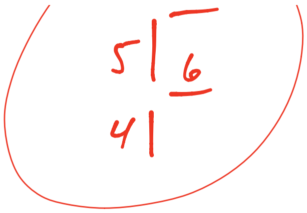


- **base: HEX3-Hex0:** 0x10000020
  HEX7-HEX4: 0x10000030

Example: write the letter ‘F’ to 7seg display:



0



→

```
0b1110001
  6543210
```

```
.equ 7SEGS_LOWER, 0x10000020
movia r8, 7SEGS_LOWER # addr of device
 movi r9, 0b1110001 ←
stwio r9, 0(r8)
```

POLLING

- devices often much slower than CPU
  - and run asynchronously with CPU
  - i.e., use a different clock
- how can CPU know when device is ready?
  - must check repeatedly
  - called “polling”
  - asking “are you ready yet? Are you ready yet?…”

TIMER

- like a stopwatch:
  - you set the value to count down from
  - can start/stop, check if done, reset, etc.
- counts at 50MHz

.equ TIMER, 0x10002000

0(TIMER): write zero here to reset the timer;

bit1: 1 if timer is running

bit0: 1 if timer has timed out

4(TIMER): bit3: write 1 to stop timer

bit2: write 1 to start timer

bit1: set to 0 to make timer wait after timeout before continuing

8(TIMER): low 16bits of timeout period

12(TIMER): high 16bits of timeout period

Example: 5-second-wait

- Wait for 5-seconds using timer0
- First: must compute the desired timer period
  - recall: timer runs at 50MHz

50 MHz = 50 million cycles/sec

timer period = 5 sec x 50 MHz

= 5 sec x 50 million cycles/sec

= 250 million cycles

$$= 0x\underbrace{0EE6}_{\text{upper}}\ \underbrace{B280}_{\text{lower}}$$

```
.equ TIMER, 0x10002000
.equ PERIOD, 0x0EE6B280
movia r8, TIMER

movui r9, %lo(PERIOD) # %lo - macro-assembler
```

```
                                   # extracts bits[15..0]
                                   # movui - move unsigned immed
    stwio r9, 8(r8)   # lower hword of period
  movui r9, %hi(PERIOD) # %hi . macro assembler
                                   # extracts bits[31..16]

    stwio r9, 12(r8)

    stwio r0, 0(r8) # reset timer

    movi r9, 0x6 # 0b110 - bit 2 - start timer
                                   bit 1 - timer won't wait
                                     after timeout

   stwio r9, 4(r8)

POLL:     ldwio r9, 0(r8)
          andi  r9, r9, 0x1    # check if timer has timed
                                                      out
          beq r9, r0, POLL     # loop & check again
          # 5 sec have elapsed, do action
          stwio r0, 0(r8)   # clear timer
          br POLL # wait another 5 secs
```

INTERFACES

- "serial"
  - means transmit one bit at a time
  - i.e., over a single data wire
- "parallel"
  - means transmit multiple bits at a time
  - over multiple wires

  - more expensive
    - more$$$, wires, pins, hardware
- which is faster?

depends on material, design. protocols etc

GENERAL PURPOSE IO Interfaces

- two parallel interfaces on DE2
  - aka general purpose IO interfaces, GPIO
  - called JP1 and JP2
- each interface has:
  - 32 pins
  - each pin can be configured as input or output
    - individually!
- pins configured as input default to 1
  - called "pull-up"
- pins configured as output default to 0
  - default value of output register

GPIO LOCATIONS

JP1: 0x10000060

JP2: 0x10000070

For each:

0(JPX): DR data in/out (32 bits)

4(JPX): DIR data direction register

each bit configures data pin as in or out (32 bits)

0 means input, 1 means output

Example1

- configure JP1 as all input bits
  - and read a byte

```
.equ JP1, 0x1000 0060
 movia r8, JP1
  stwio r0, 4(r8) # config as input
  ldwio r9, 0(r8) # read word from data pins
```

Example2

- configure JP2 as all output bits
  - and write a character to the lowest 8 bits

```
.equ JP2, 0x1000 0070
movia r8, JP2
 movi  r9, 0xFFFF # sign extend → r9 = 0xFFFFFFFF
stwio r9, 4(r8) #config as all outputs
movui r9, 'X' # ascii char x
 stwio r9, 0(r8) #write to data pins
```

Example3

- configure JP1
  - lower 16bits input, upper 16 bits output, read then write it back

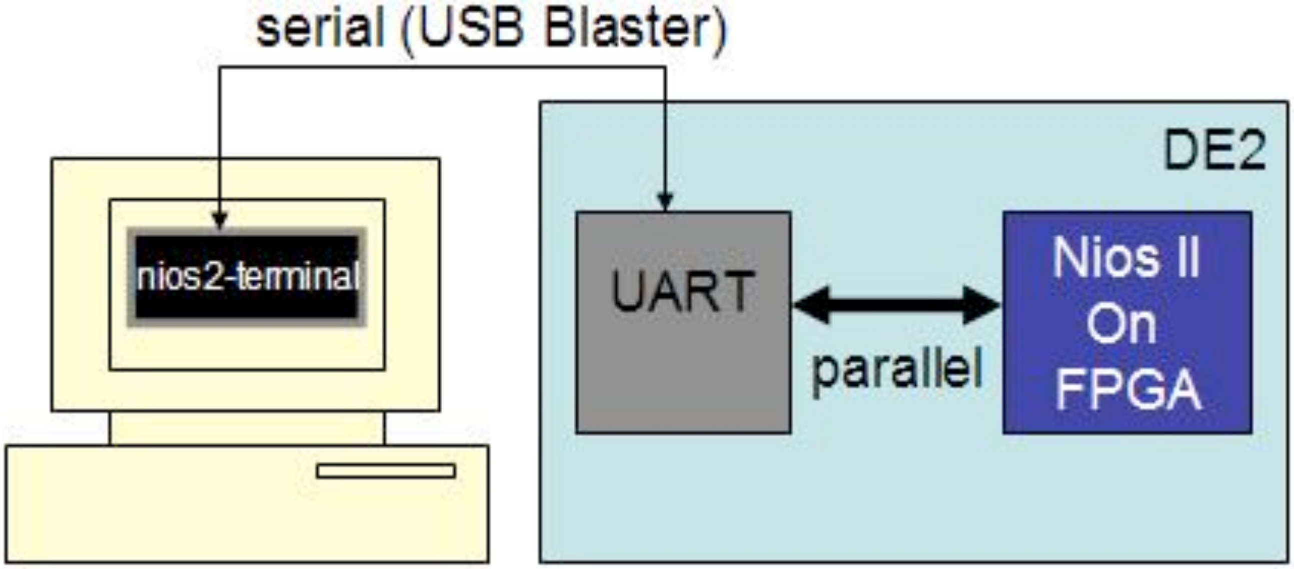
```
.equ JP1. 0x10000060
movia r8, JP1
movia r9, 0XFFFF0000  # config lower 16-input, upper 16
                        output

stwio r9, 4(r8)  # write dir reg
ldwio r9, 0(r8)  # read from data pins
andi r9, r9, 0xFFFF  # r9 & 0x0000FFFF . mask lower
                          16 bits

slli r9, r9, 16      #shift left 16
stwio r9, 0(r8)  # device will ignore lower 16 - config
                      as input
```

Serial Interfaces:

- send/recv 1 bit at a time
  - in each direction
- cheap
  - eg., only one data wire, plus a few control wires
- can be very fast

- ex: COM port on a PC, RS-232 is standard
  - Usually a nine pin connector
  - used to be very common in PCs
  - now replaced by USB
  - still very common in embedded systems

JTAG UART



- JTAG: Joint Test Action Group
  - standard interface for test and debug for ICs
  - connects to host PC via USB blaster cable
- UART:
  - Universal Asynchronous Receiver Transmitter
  - serial device
- Asynchronous:
  - data can be sent/rec'd at any time

JTAG UART

.equ JTAG_UART, 0x10001000

0(JTAG_UAR~~G~~T): data register: reading gets the next datum

bit15: read valid

bits7-0: data

4(JTAG_UART): control register:

bits31-16: number of character
spaces available to write

EXAMPLE: echo

- read a character then send it back

```
.equ JTAG_UART, 0x10001000
  movia r8, JTAG-UART

wait-recv: ldwio r9, 0(r8)
           andi r10, r9, 0x8000   # check if bit 15 is
                                  1
           beq r10, r0, wait-recv  # if read invalid
                                     try again

           andi r9, r9, 0xff  # mask-lowest byte
                                input char
```

NOTE: run “nios2-terminal” in NIOS command window to start a shell

OTHER DE2 MEM-MAPPED DEVICES

- slider switches
- push buttons
- LEDs
- LCD display
- RS232 UART
- audio codec
- VGA adapter
- ps2 connector (mouse)
- Digital protoboard
- see DESL www for full details

ECE243

Interrupts