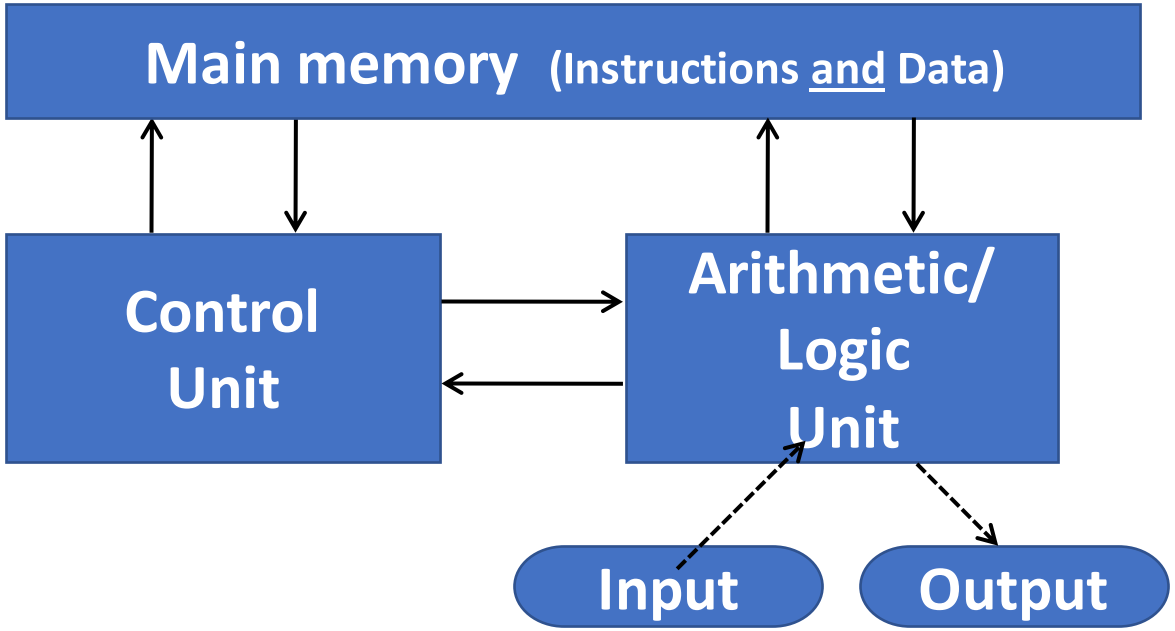
Topic 10 – Memory Mapped IO

Input/Output Devices

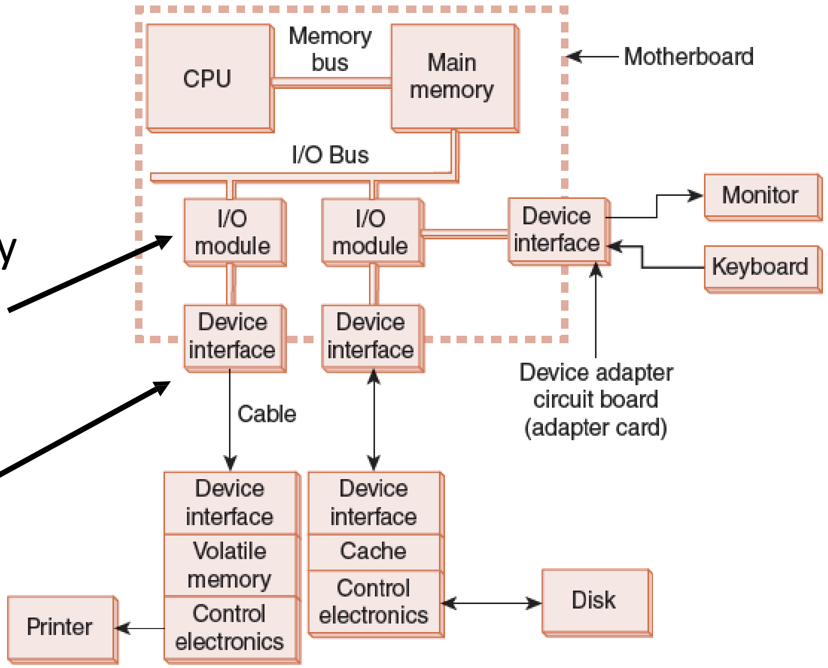
- I/O devices are what makes a system useful
  - Input devices
  - Output devices
  - Storage devices



A Model I/O Configuration

Moves data b/w memory and a device interface (a.k.a. controller)

One interface per device class, e.g., USB interface



Interfaces and Protocols

- Interfaces communicate with certain types of devices, such as keyboards, disks, or printers
- Interfaces make sure
  - Device is ready for next batch of data
  - Host is ready to receive the next batch of data coming in from the peripheral device

- The exact form and meaning of the signals exchanged between the sender and the receiver is called a protocol
  - Signals are of two types: command and data signals
  - Handshake: A protocol exchange in which the receiver sends an Acknowledgement for the commands and data sent or indicate that it is ready to receive data

Memory-Mapped I/O (MMIO)

- I/O devices and memory share the same address space
- Each I/O device has its own reserved block of memory
- A memory address is called an IO register
- Data transfers to and from the I/O device involve moving bytes to and from the memory address that is mapped to the device
- MMIO is like using regular load/store instructions from the programmer’s perspective
- Simplicity and convenience (Yes for the programmer)

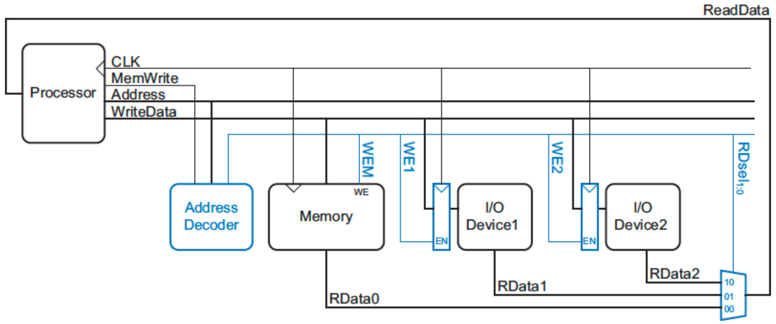
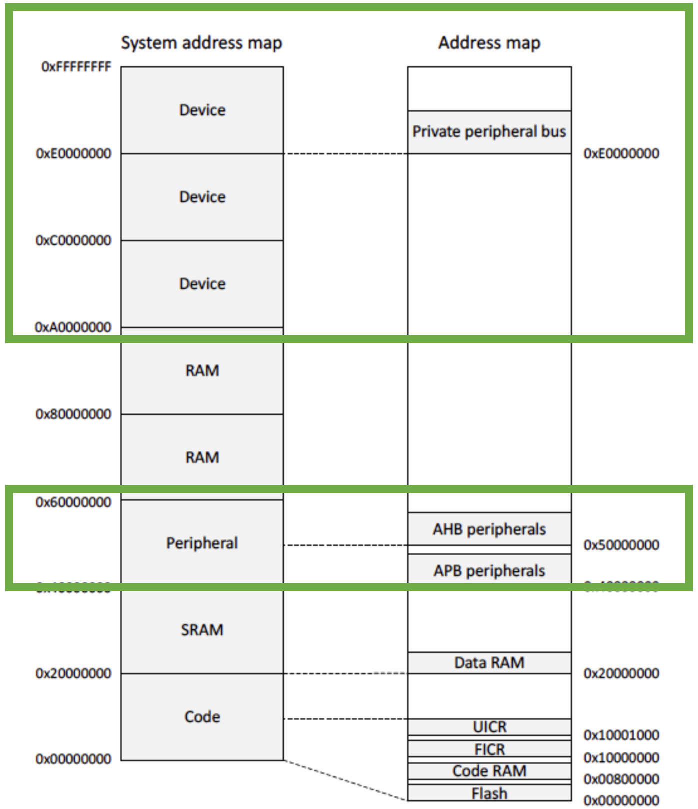


Figure e9.1 Support hardware for memory-mapped I/O

Memory Map revisited

- The *stack (SRAM region)* stores all local variables and function arguments.
- The *heap (SRAM region)* stores data allocated during runtime (e.g. malloc).
- The *global data (SRAM region)* area stores global variables
- *The text (CODE region)* stores the machine language program.

- 0x40000000-0x60000000 typically use for MMIO operations.



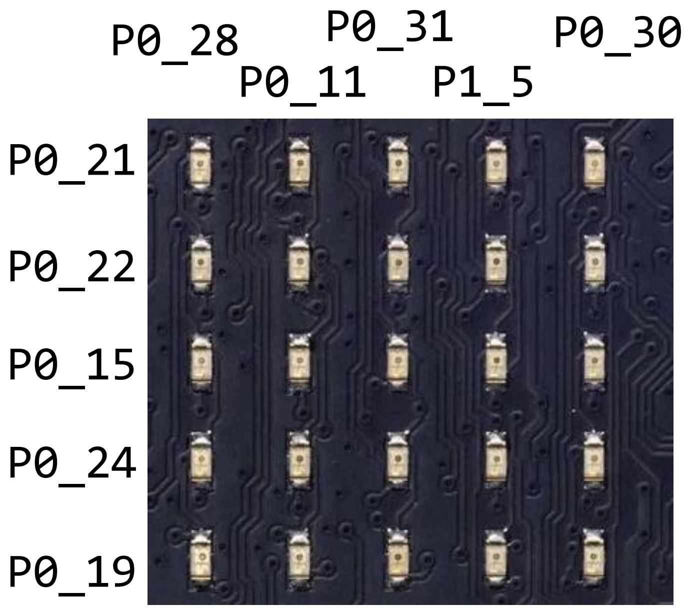
IO Devices

Data
- Stack
- Heap
- Global

Code

Micro:bit LEDs

- The CPU on the micro:bit uses the GPIO mechanism to control LEDs.
- There are two GPIO ports (0 and 1), with 32 I/O pins each.
- For scalability, the LED screen uses only 10 pins and allows the control of individual *rows* or *columns*.
- You need some clever programming to light complex LED patterns (week 18 lab task).



GPIO MMIO Register

Name	Address	Description
OUT	0x50000504 -	Write GPIO port
IN	0x50000510 -	Read GPIO port
DIRSET	0x50000518 -	Direction of GPIO pins

DIRSET

Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Id	f	e	d	c	b	a	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
Reset 0x00000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Id	RW	Field	Value Id	Value	Description
A	RW	PIN0			Set as output pin 0
			Input	0	Read: pin set as input
			Output	1	Read: pin set as output
			Set	1	Write: writing a '1' sets pin to output; writing a '0' has no effect
B	RW	PIN1			Set as output pin 1
			Input	0	Read: pin set as input
			Output	1	Read: pin set as output
			Set	1	Write: writing a '1' sets pin to output; writing a '0' has no effect
C	RW	PIN2			Set as output pin 2
			Input	0	Read: pin set as input
			Output	1	Read: pin set as output
			Set	1	Write: writing a '1' sets pin to output; writing a '0' has no effect

https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.nrf52832.ps.v1.1%2Fgpio.html

LED programming

```
@ Init LEDs
ldr r0, =0x50000514 @ GPIO0_DIR register
ldr r1, =0xd1688800 @ LED pins as outputs
str r1, [r0]
ldr r0, =0x50000814 @ Same for GPIO1_DIR
ldr r1, =0x00000020
str r1, [r0]

ldr r4, =0x50000504 @ Address of GPIO0_OUT
register
ldr r5, =0x50000804 @ Address of GPIO1_OUT
register
ldr r6, =0x50008800 @ Bit pattern 0 for dot
ldr r7, =0x00000020 @ Bit pattern 1 for dot
str r6, [r4]        @ Light an LED
str r7, [r5]
```

Set ports as outputs for every PIN used to control LEDs.

Set Row 3 and Columns 1,2,4,5 to 1.

Note: In order to light a led, you need to turn on ts input, and turn off the output, to allow current to flow.

Temperature Sensor

- The temperature sensor measures die temperature over the temperature range of the device.
  - Temperature range >= device operating temperature
  - Resolution is 0.25 degrees
- TEMP is started by triggering the START register.
- A DATARDY event is generated, when temp reading is reading and is available via the TEMP register.

TEMP MMIO Registers

Register	Offset	Description
TASKS_START	0x4000C000	Start temperature measurement (write 1 to start)
TASKS_STOP	0x4000C004	Stop temperature measurement (write 1 to start)
EVENTS_DATARDY	0x4000C100	Temperature measurement complete, data ready (readonly)
INTENSET	0x4000C304	Enable interrupt
INTENCLR	0x4000C308	Disable interrupt
TEMP	0x4000C508	Temperature in °C (0.25° steps) (32-bit int, multiply by 4 to get temp in celcius)

TEMP programming

```
        ldr r0, =0x4000C000
        mov r1, #1
        str r1, [r0]        @ Start measurement
wait:
        ldr r1, [r0, #0x100] @ Check if data are ready
        cmp r1, #1          @ If not ready, repeat
        bne wait

        ldr r1, [r0, #0x508] @ read the temp reading
```

Polled versus Interrupt I/O

- Polled I/O
  - CPU monitors a control/status register associated with a port
  - When a byte arrives in the port, a bit in the control register is set
  - The CPU eventually polls and notices that the “data ready” control bit is set
  - The CPU resets the control bit, retrieves the byte, and processes it
  - The CPU resumes polling the register as before
- Interrupt-driven I/O
  - CPU is not held up from doing other things
  - Interrupts are asynchronous signals
  - The devices tell the CPU when they have data to send
  - The CPU proceeds with other tasks until a device requesting service sends an interrupt to the CPU
  - Granularity is configurable: Interrupts for every word, or for an entire batch

Conclusions

- MMIO is a mechanism to connect IO devices with the CPU.
  - Use memory read/writes to access the device state.

- Use case: LED, Temp sensor
- Polling vs interrupt
- Next time: week 16/17 practical.