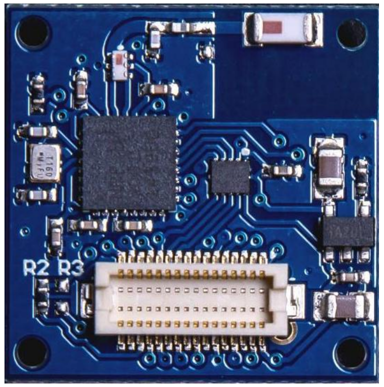
EE107 Spring 2019

Lecture 2

MCUs and IO



Embedded Networked Systems

Sachin Katti

*slides adapted from Aaron Schulman's CSE190

Reading for next week

- Posted on course website – Please skim.
  - “ARM Cortex-M for Beginners”

WHITE PAPER

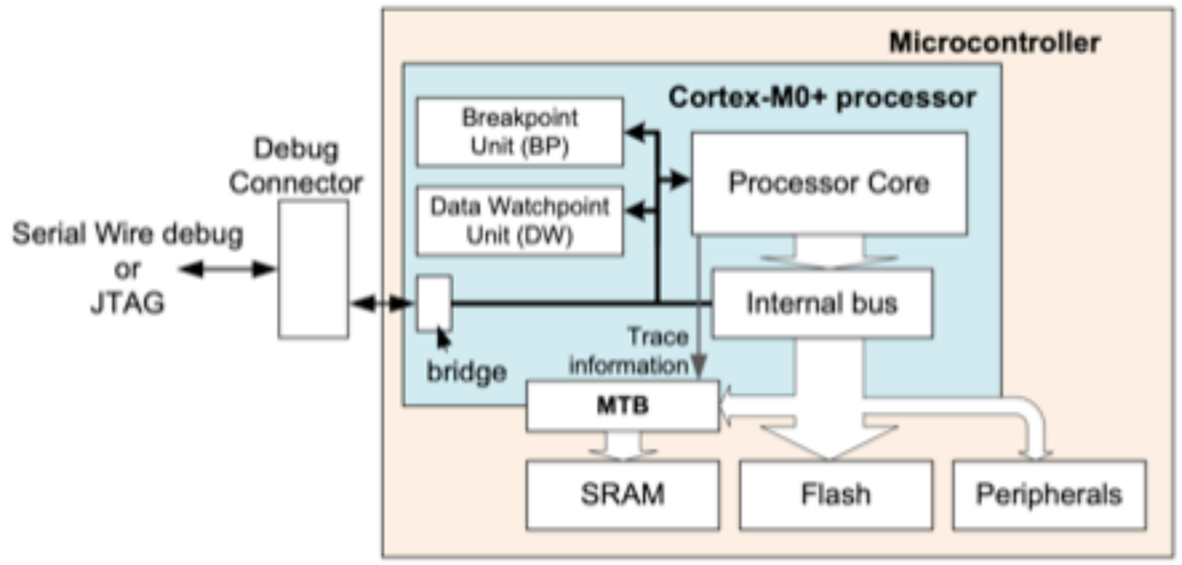
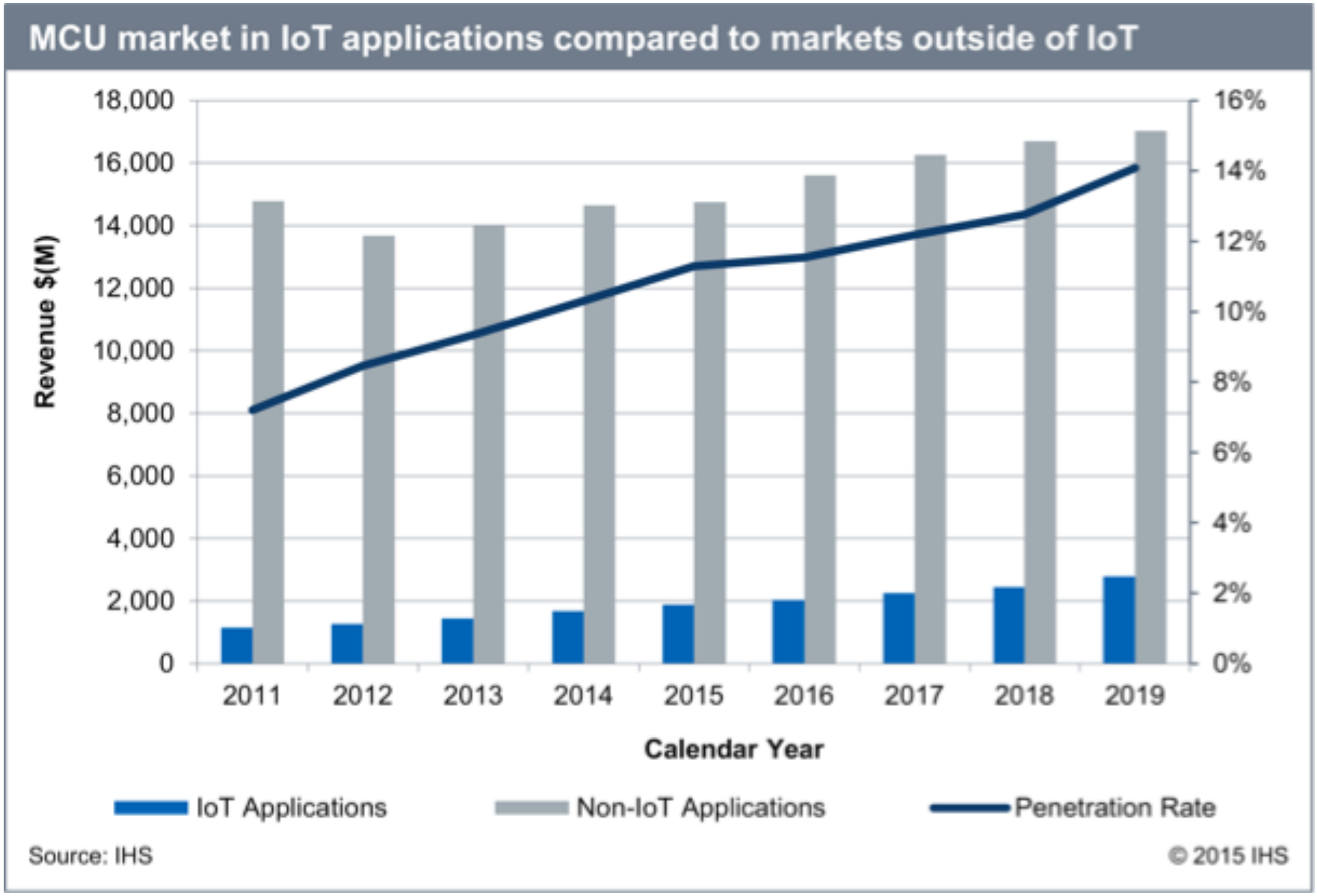


Figure 13: MTB feature in Cortex-M0+ processor provides low cost instruction trace solution

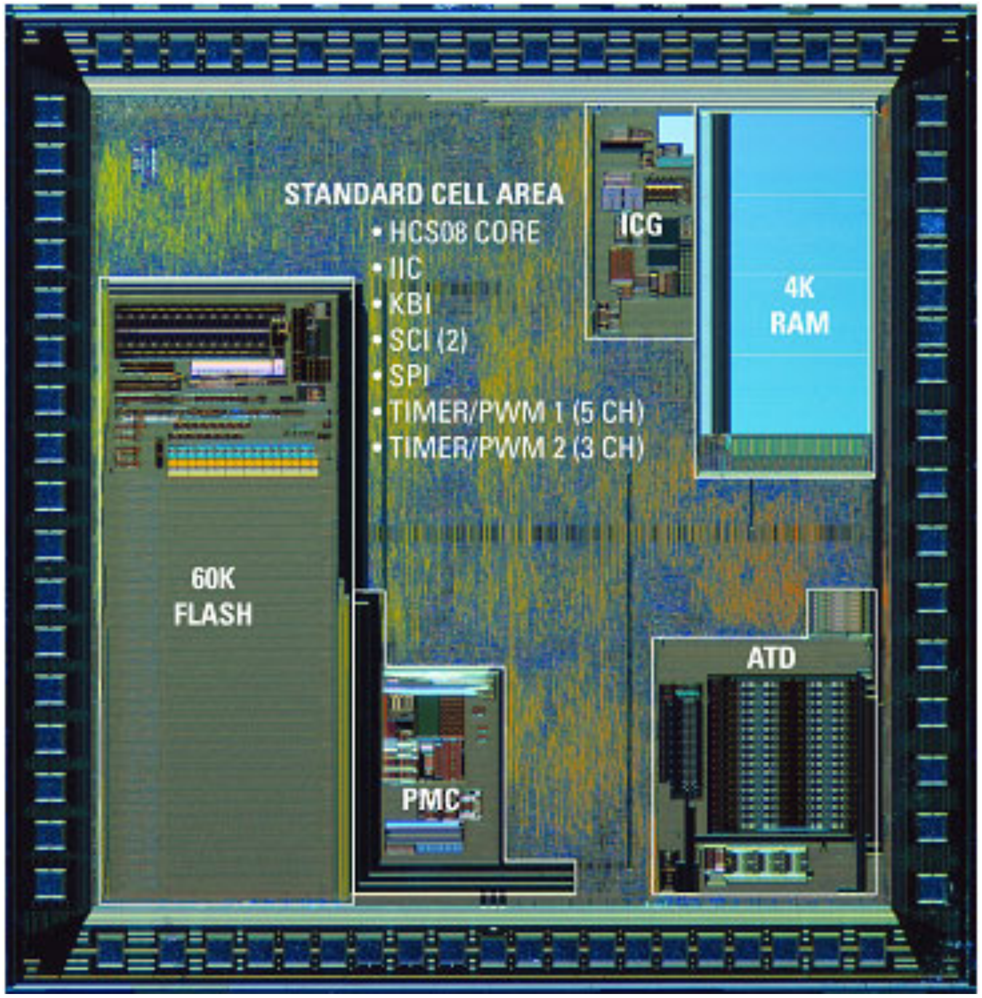
Introduction to Microcontrollers



Introduction to Microcontrollers

- A microcontroller (MCU) is a small computer on a single integrated circuit consisting of a relatively simple central processing unit (CPU) combined with peripheral devices such as memories, I/O devices, and timers.
  - By some accounts, more than half of all CPUs sold worldwide are microcontrollers

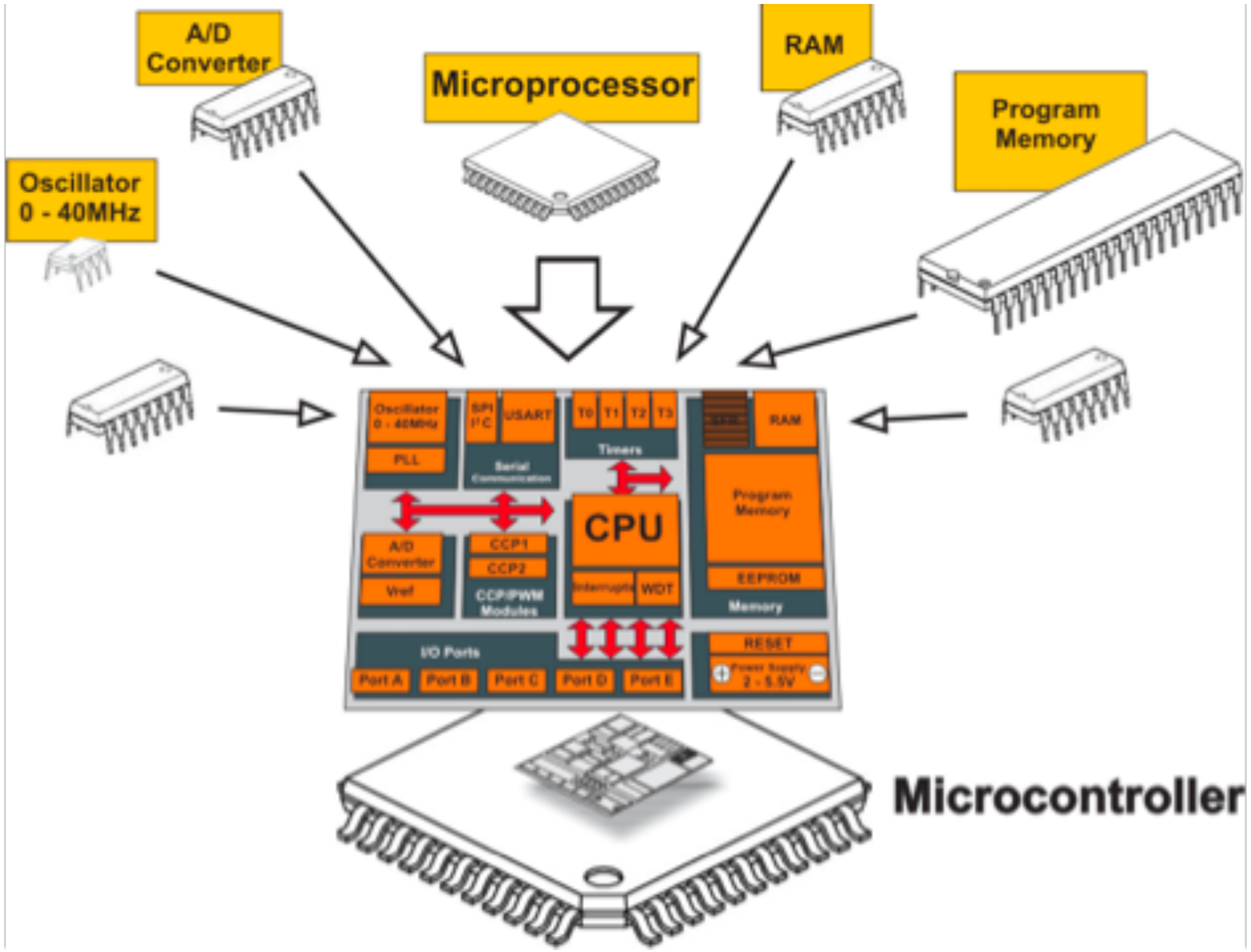
Die shot of a microcontroller



Microcontroller VS Microprocessor

- A microcontroller is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals.
- A microprocessor incorporates the functions of a computer’s central processing unit (CPU) on a single integrated circuit.

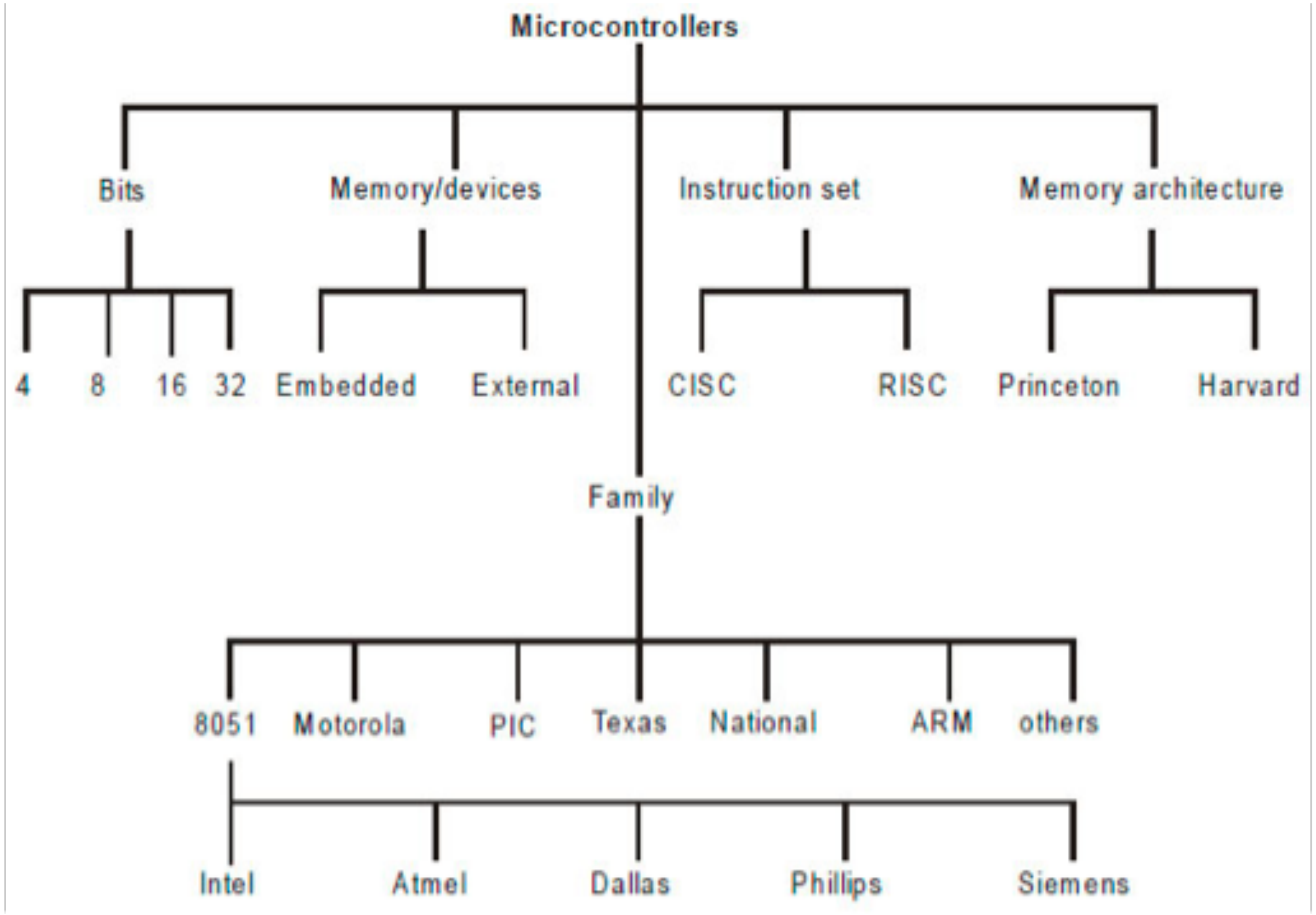
Microcontroller VS Microprocessor



Types of Processors

- In general-purpose computing, the variety of instruction set architectures today is limited, with the Intel x86 architecture overwhelmingly dominating all.
- There is no such dominance in embedded computing. On the contrary, the variety of processors can be daunting to a system designer.
- Things that matter
  - Peripherals, Concurrency & Timing, Clock Rates, Memory sizes (SRAM & flash), Package sizes

Types of Microcontrollers



How to choose MCU for our project?

- What metrics we need to consider?
  - Power consumption
  - Clock frequency
  - IO pins
  - Memory
  - Internal functions
  - Others

How to choose MCU for our project?

- What metrics we need to consider?
  - Power consumption
    - We cannot afford powerful MCU because the power budget of the system is 0.2mA (assuming running for a month on 150mAh battery).
  - Clock frequency
    - kHz is too slow…
    - 100MHz is over kill...
  - IO pins
    - Lots of peripherals - Image sensor, UART debugger, SD card, DAC, ADC, microphone, LED

How to choose MCU for our project?

- What metrics we need to consider?
  - Memory
    - We need to have sufficient memory for storing sensor data
  - Internal functions
    - Migrating data from the sensor to the radio (DMA)

How to choose MCU for our project?

- Clock frequency
  - kHz is too slow
    - Image sensor clock rate is ~4MHz
  - 100MHz is too fast
    - Power consumption is high
  - Several MHz would be ideal

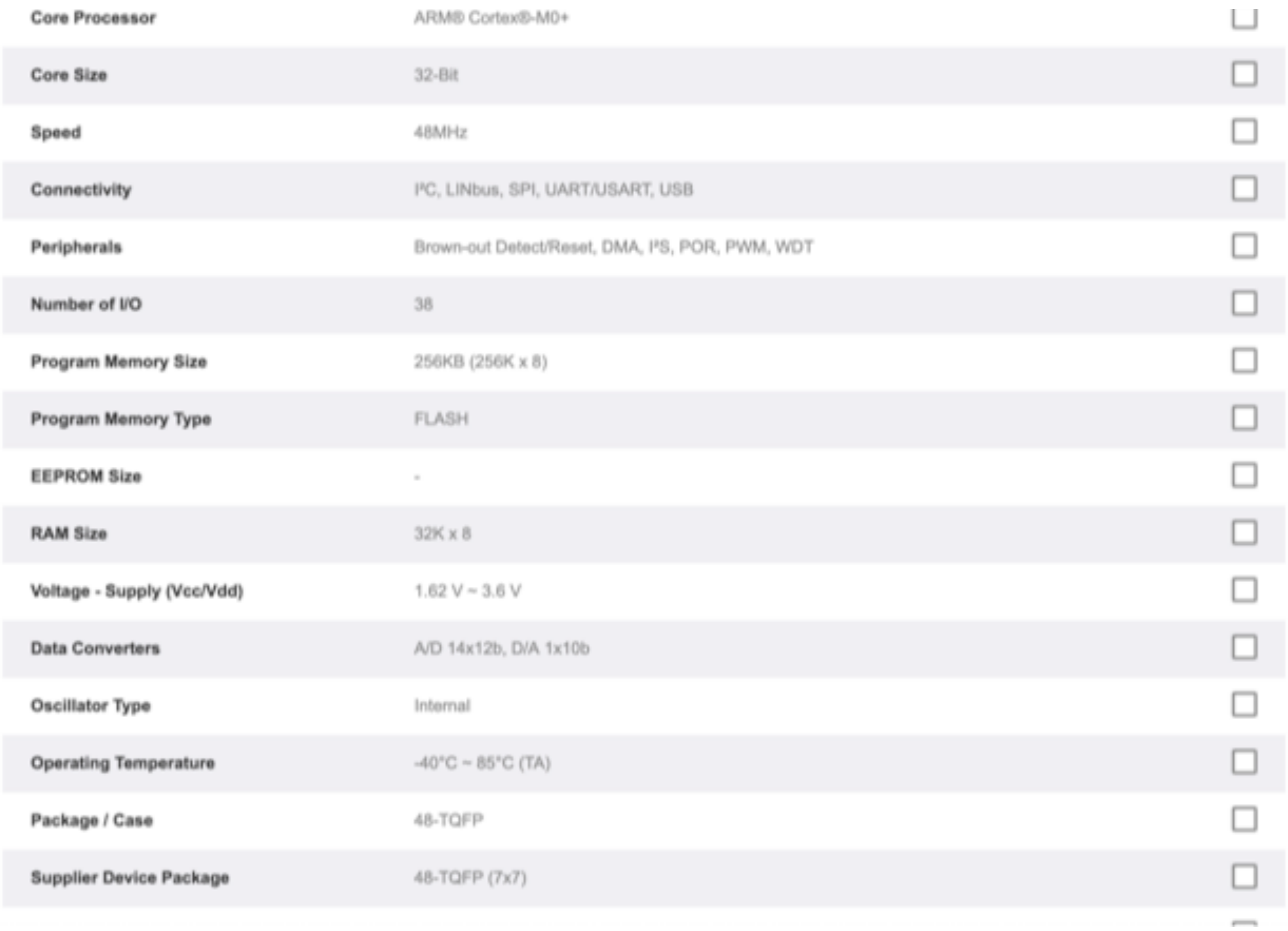
How to choose MCU for our project?

- IO pins
  - Interfacing sensors, UART debugger, SD card, DAC, ADC, LED
  - We need a large number of IO pins
  - We need various types of IO pins
    - This is not a problem for FPGA. Why?

How to choose MCU for our project?

- Memory
  - Store image sensor data
    - 360*240*8=84.3kB = 675kbits
  - What types of memory are available on an MCU?
    - Internal memory: RAM, too small 0.5~32kB
    - External memory – Flash: high power consumption, ~5mA for read and ~10mA for erase
    - External memory - Ferroelectric RAM: low power consumption, ~1.5mA for read and write @40MHz, expensive

The MCU used in our projects



Core Processor	ARM® Cortex®-M0+	☐
Core Size	32-Bit	☐
Speed	48MHz	☐
Connectivity	I^2C, LINbus, SPI, UART/USART, USB	☐
Peripherals	Brown-out Detect/Reset, DMA, I^2S, POR, PWM, WDT	☐
Number of I/O	38	☐
Program Memory Size	256KB (256K x 8)	☐
Program Memory Type	FLASH	☐
EEPROM Size	-	☐
RAM Size	32K x 8	☐
Voltage - Supply (Vcc/Vdd)	1.62 V ~ 3.6 V	☐
Data Converters	A/D 14x12b, D/A 1x10b	☐
Oscillator Type	Internal	☐
Operating Temperature	-40°C ~ 85°C (TA)	☐
Package / Case	48-TQFP	☐
Supplier Device Package	48-TQFP (7x7)	☐

What operations does software need to perform on peripherals?

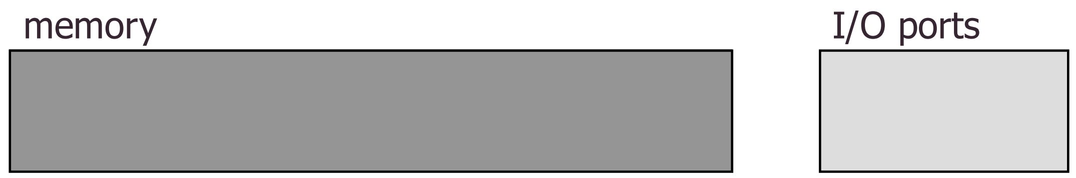
1. Get and set parameters
2. Receive and transmit data
3. Enable and disable functions

How can we imagine providing this interface to software?

1. Specialized CPU instructions (x86 in/out)

Port I/O

- Devices registers mapped onto “ports”; a separate address space



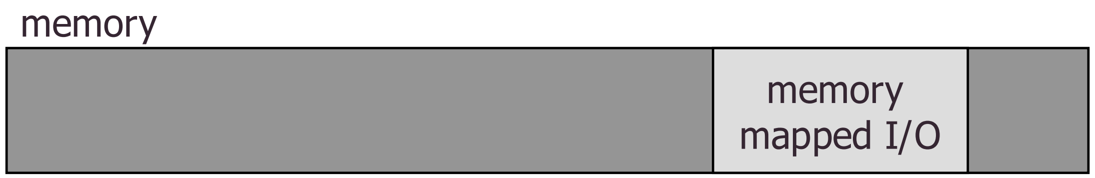
- Use special I/O instructions to read/write ports
- Protected by making I/O instructions available only in kernel/supervisor mode
- Used for example by IBM 360 and successors

How can we imagine providing this interface to software?

1. Specialized CPU instructions (x86 in/out)
2. Accessing devices like they are memory

Memory Mapped IO

- Device registers mapped into regular address space



- Use regular move (assignment) instructions to read/ write registers
- Use memory protection mechanism to protect device registers

Why MMIO for embedded systems?

- Ports I/O:
  - special I/O instructions are CPU dependent
- Memory mapped I/O:
  - memory protection mechanism allows greater flexibility than protected instructions
  - may use all memory reference instructions for I/O

Reading and writing with MMIO is not like talking to RAM

- MMIO reads and writes registers
- Reads and write to registers can cause peripherals to execute a function
- By reading data, it may cause the hardware to do something
  - E.g., Clear the interrupt flags, get the next BYTE on UART
- By writing data, it may cause the hardware to do something with it
  - E.g., Send this data over the UART bus

GPIOs are important in our project

- GPIOs are not only used for blinking LEDs

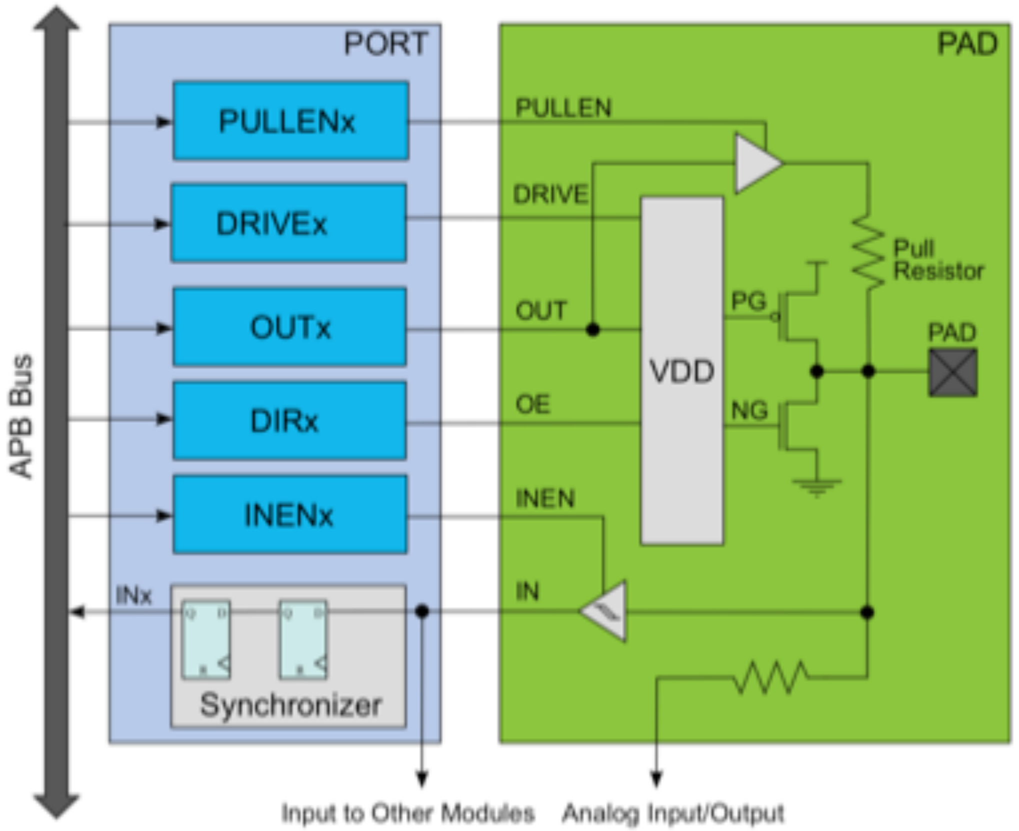
- Passing messages
  - Interrupt the radio for transmitting the data
  - Read pin status to receive configuration messages

- Debugging
  - Did I execute my interrupt service routine?
  - Is the timer running as expected?
  - Why using GPIO?
    - GPIO ops are lightweight

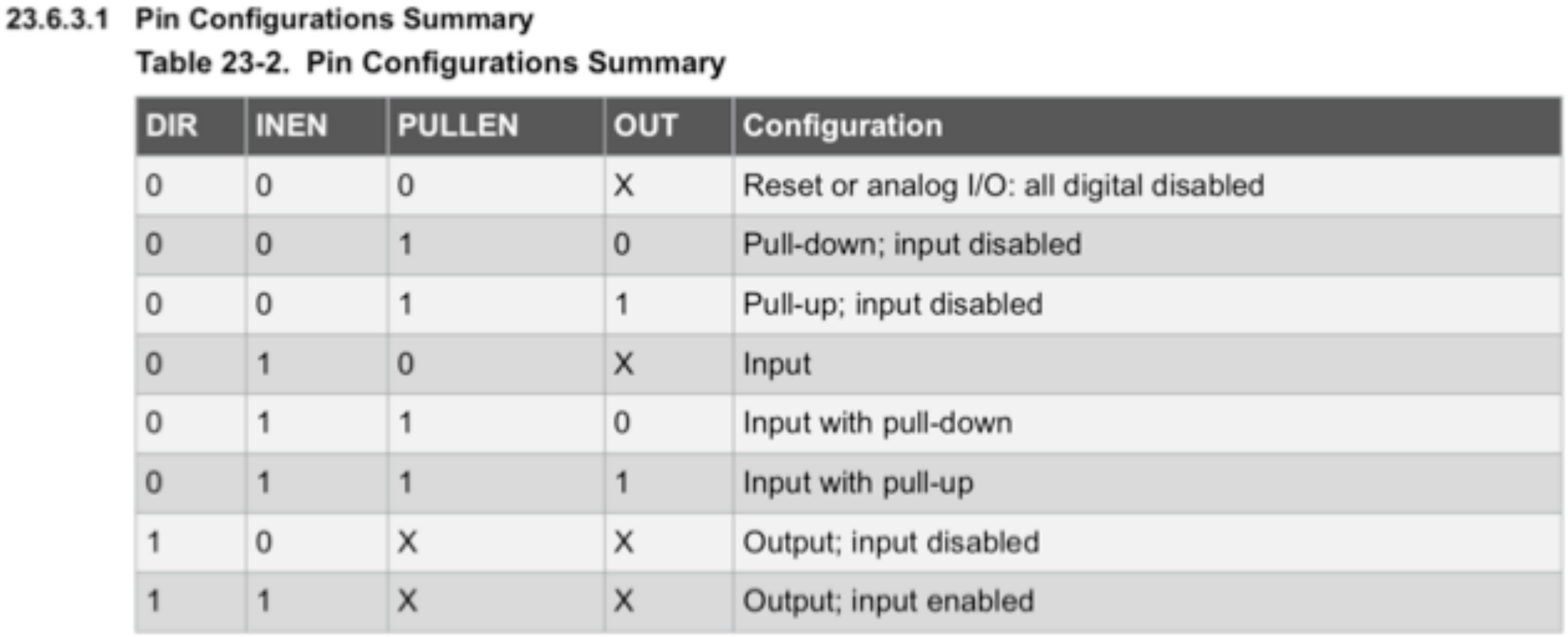
Topology of a GPIO pin

Functional Description

Figure 23-2. Overview of the PORT



GPIO Configurations

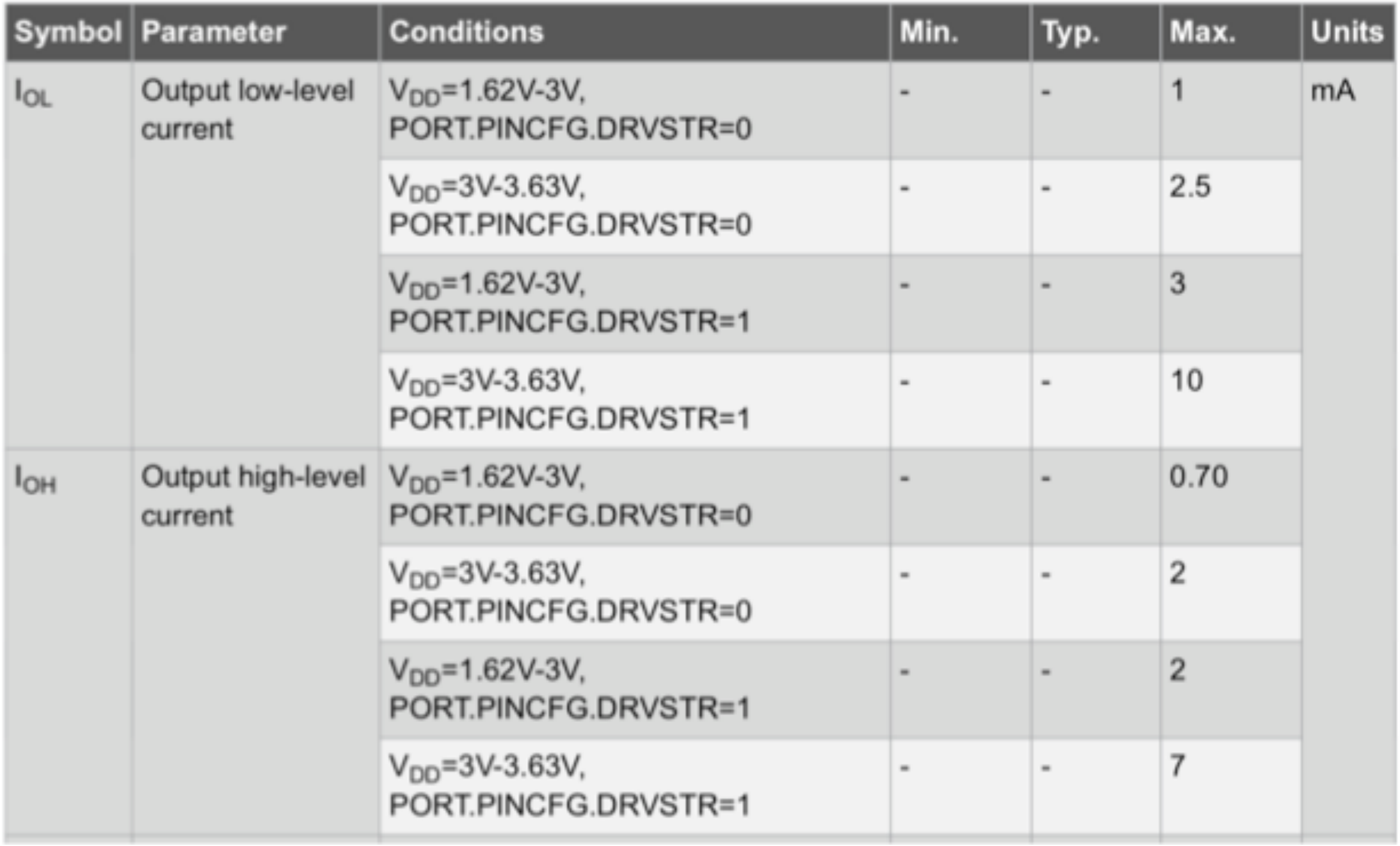


23.6.3.1 **Pin Configurations Summary**

Table 23-2. Pin Configurations Summary

DIR	INEN	PULLEN	OUT	Configuration
0	0	0	X	Reset or analog I/O: all digital disabled
0	0	1	0	Pull-down; input disabled
0	0	1	1	Pull-up; input disabled
0	1	0	X	Input
0	1	1	0	Input with pull-down
0	1	1	1	Input with pull-up
1	0	X	X	Output; input disabled
1	1	X	X	Output; input enabled

A fun extra feature: Drive Strength



Symbol	Parameter	Conditions	Min.	Typ.	Max.	Units
I_{OL}	Output low-level current	V_{DD}=1.62V-3V, PORT.PINCFG.DRVSTR=0	-	-	1	mA
		V_{DD}=3V-3.63V, PORT.PINCFG.DRVSTR=0	-	-	2.5	
		V_{DD}=1.62V-3V, PORT.PINCFG.DRVSTR=1	-	-	3	
		V_{DD}=3V-3.63V, PORT.PINCFG.DRVSTR=1	-	-	10	
I_{OH}	Output high-level current	V_{DD}=1.62V-3V, PORT.PINCFG.DRVSTR=0	-	-	0.70	
		V_{DD}=3V-3.63V, PORT.PINCFG.DRVSTR=0	-	-	2	
		V_{DD}=1.62V-3V, PORT.PINCFG.DRVSTR=1	-	-	2	
		V_{DD}=3V-3.63V, PORT.PINCFG.DRVSTR=1	-	-	7	