Föreläsning 5: Programmering av inbyggda system

Datum: 2024-02-26

Föreläsare: Mert Demirsü

Dagens genomgång (Teoretisk)

FPGA

GPU

ASICs & Microcontrollers

CPU & Architectures

System on Chip

Hardware Acceleration

Arduino UNO Intro

Feedback

Övningar läggs upp efter Lunch

Excel för bokning för redovisning av Labb 1 Måndag/Torsdag

https://docs.google.com/spreadsheets/d/19BM1IzFjMTFLiwcJyiRyOaJF1ixKhvMGb6kafT0jFNE/edit?usp=sharing

Några Frågor/Feedback ?

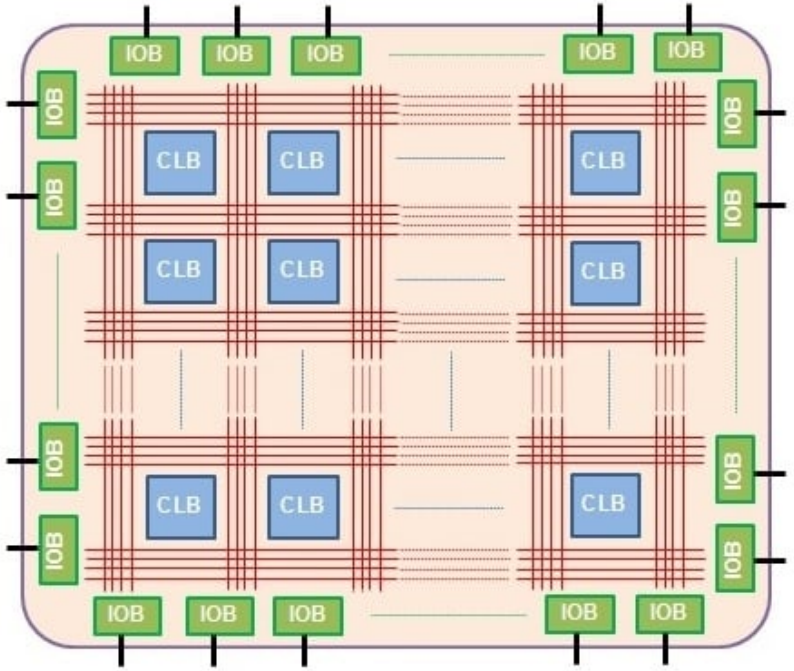
Short Summary table

	CPU	FPGA	GPU	ASIC
Overview	Traditional sequential processor for general-purpose applications	Flexible collection of logic elements and IP blocks that can be configured and changed in the field	Originally designed for graphics; now used in a wide range of computationally intensive applications	Custom integrated circuit optimized for the end application
Processing	Single- and multi-core MCUs and MPUs, plus specialized blocks: FPU, etc.	Configured for application; SoCs include hard or soft IP cores (e.g., Arm)	Thousands of identical processor cores	Application-specific: may include third-party IP cores
Programming	OSes, APIs run huge range of high-level languages; assembly language	Traditionally HDL (Verilog, VHDL); newer systems include C/C++ via openCL & SDAccel	OpenCL & Nvidia's CUDA API allow general-purpose programming (e.g., C, C++, Python, Java, Fortran)	Application-specific: TensorFlow open-source framework for Google's TPU; CPU manufacturers (e.g., Intel) include tools with new ASIC releases
Peripherals	Wide choice of analog and digital peripherals in MCUs; MPUs include digital bus interfaces	SoCs include many transceiver blocks, configurable I/O banks	Very limited; e.g., only cache memory	Tailored to application: may include industry-standard functions (USB, Ethernet, etc.)
Strengths	Versatility, multitasking, ease of programming	Configurable for specific application; configuration can be changed after installation; high performance per watt; accommodates massively parallel operation; wide choice of features: DSPs, CPUs	Massive processing power for target applications—video processing, image analysis, signal processing	Custom-designed for application with optimum combination of performance and power consumption
Weaknesses	OS capability adds high overhead; optimized for sequential processing with limited parallelism	Relatively difficult to program; second-longest development time; poor performance for sequential operations; not good for floating-point operations	High power consumption, not suited to some algorithms; problems must be reformulated to take advantage of parallelism, but API frameworks provide abstraction	Longest development time; high cost; cannot be changed without redesigning the silicon

It's also worth considering how these choices stack up in some common applications. As shown in the table, designers can often use any or all of the options either alone or, more likely, in combination.

Applications	CPU	FPGA	GPU	ASIC	Comments
Vision & image processing		✓	✓	✓	FPGA may give way to ASIC in high-volume applications
AI training			✓		GPU parallelism well-suited for processing terabyte data sets in reasonable time
AI inference	✓	✓	✓	✓	Everyone wants in! FPGAs perhaps leading; high-end CPUs (e.g., Intel's Xeon) and GPUs (e.g., Nvidia's T4) address this market
High-speed Search	✓	✓	✓	✓	Microsoft's Bing uses FPGAs; Google uses TPU ASIC; CPU needed for coordination & control
Industrial motor control	(✓)	✓		✓	Many motor-control MCUs and ASICs available; FPGAs offer a quick-turn ASIC alternative
Supercomputer HPC	✓		✓		Majority of TOP500 supercomputers uses some combination of CPUs and GPUs
General-purpose computing	✓		(✓)		CPU most versatile, flexible option; GPUs beginning to perform some tasks
Embedded control	✓	✓		✓	CPUs (→ MCU) dominant in low-cost, space-constrained, low-power, mobile applications
Prototyping, low-volume		✓			FPGAs best choice for low-volume, high-end applications; also pre-silicon validation, post-silicon validation and firmware development

FPGA - Field Programmable Gate Array



Each "CLB" Block is "specialized" and works in Parallel for a single task, this often results in effective hardware accelerated performance. Each Block can also be reprogrammed to be used for other tasks.

Notera! ASICs Kan anses som FPGA men "Icke-reprogrammerbara", I praktiken är det djupare en så men detta bör inte betraktas i denna kursen.

Specific applications using an FPGA include digital signal processing, biomedical instrumentation, device controllers, software-defined radio, random logic, medical imaging, computer hardware emulation, voice recognition, cryptography, filtering and communication encoding, and more. Today mostly replaced by microcontrollers except in the most basic cases.

FPGA - Field Programmable Gate Array

Benefits

Customizability: FPGAs can be programmed and reprogrammed to perform a wide range of functions, offering flexibility across different applications and stages of product development.

Efficiency for Specific Tasks: For certain applications, especially those requiring specific, custom logic or real-time processing, FPGAs can be more efficient than CPUs and GPUs.

Low Latency: FPGAs excel in environments where low latency is critical, as they can process data with minimal delay.

Parallel Processing: Like GPUs, FPGAs can handle parallel processing, making them suitable for tasks that can be broken down into simultaneous operations.

Power Consumption: For some applications, FPGAs may offer better power efficiency compared to general-purpose processors, especially when tailored for specific tasks.

FPGA - Field Programmable Gate Array

Cons

Complex Development Process: Programming FPGAs requires specialized knowledge in hardware description languages (HDLs) like VHDL or Verilog, making the development process more complex and time-consuming.

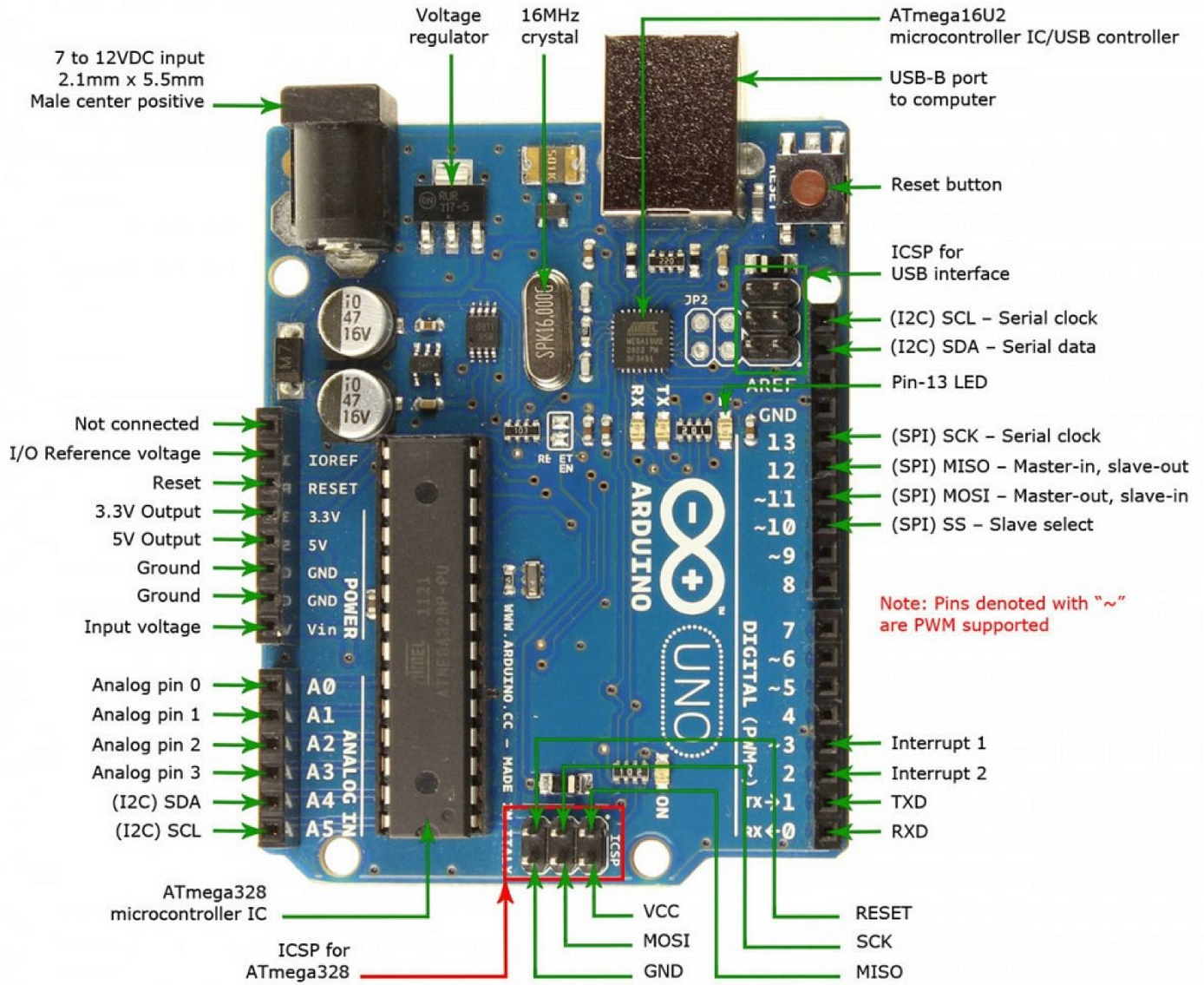
Cost: While the unit cost of FPGAs can be high, especially for low volumes, the total cost of ownership might also include development tools and time.

Flexibility vs. Optimization Trade-off: While FPGAs offer great flexibility, achieving optimal performance for a specific task requires deep technical expertise and can be labor-intensive.

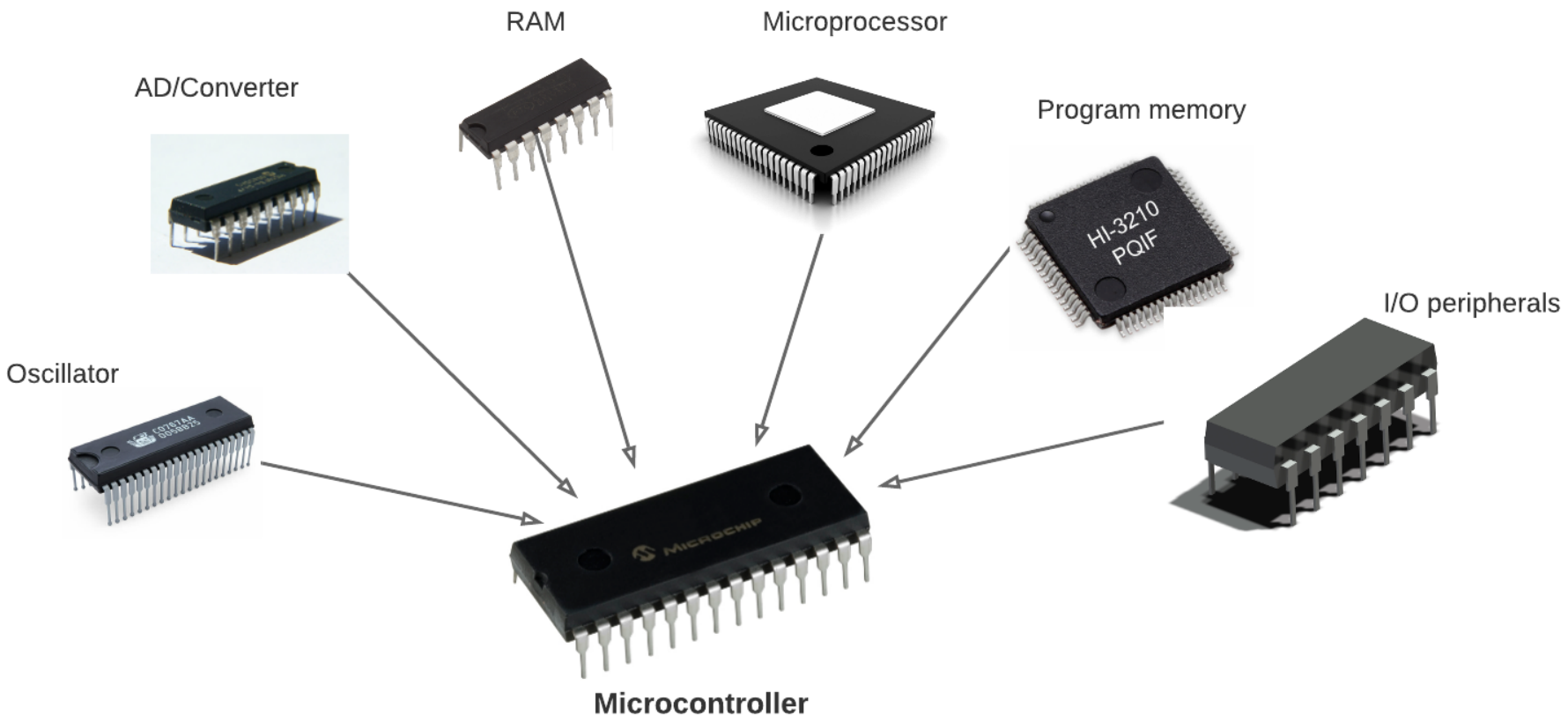
Initial Development Time: Due to their complexity and the need for custom programming, FPGAs can have longer initial development times compared to using off-the-shelf CPUs or GPUs.

Power Efficiency: Although FPGAs can be more power-efficient for certain applications, their power consumption can still be significant, especially when not optimally programmed.

Microcontrollers



Microcontrollers



Microcontrollers

Often defined as compressed "microprocessor"

Used in "compact devices" such as Arduino UNO

Contains several "parts" such as RAM, a compact CPU, I/O Peripheral input, Analog to digital converter and oscillator (timer). Thus could be considered a cut down "Mini-PC".

Microcontrollers

Strengths:

Cost-Effective: Affordable for various applications.

Compact and Power Efficient: Ideal for portable and space-constrained devices.

Versatile: Supports numerous communication protocols and peripherals.

Accessible Development: Benefited by extensive tools, libraries, and community support.

Microcontrollers

Weaknesses:

Limited Resources: Constrained processing power and memory.

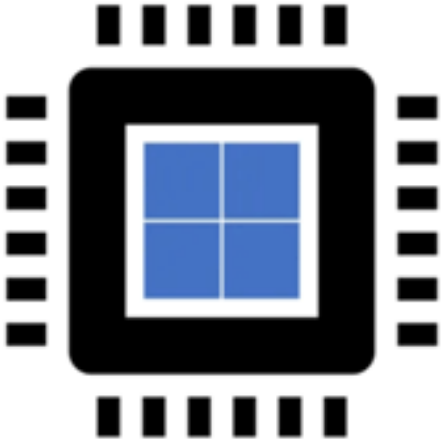
Scalability Issues: Difficult to upgrade or enhance functionality without redesign.

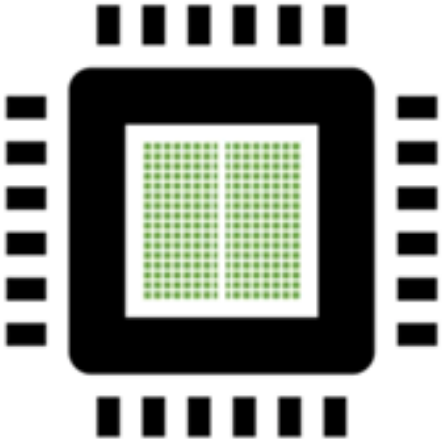
Specialized Programming: Requires knowledge of embedded C or assembly language.

Debugging Challenges: Hardware interactions and real-time constraints complicate debugging.

GPU - Graphical Processing Unit

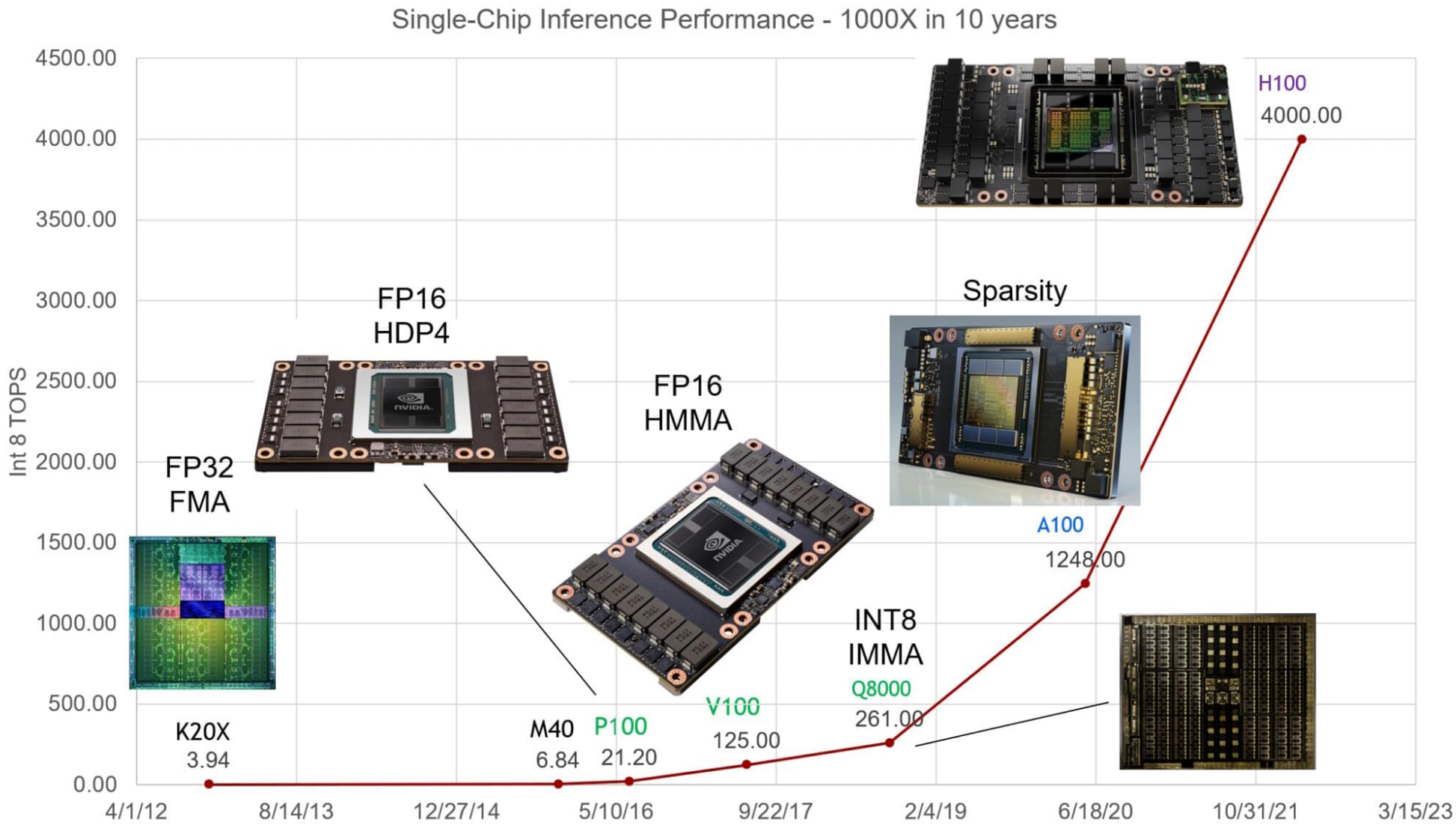
GPU - Graphical Processing Unit





CPU	GPU
Central Processing Unit	Graphics Processing Unit
4-8 Cores	100s or 1000s of Cores
Low Latency	High Throughput
Good for Serial Processing	Good for Parallel Processing
Quickly Process Tasks That Require Interactivity	Breaks Jobs Into Separate Tasks To Process Simultaneously
Traditional Programming Are Written For CPU Sequential Execution	Requires Additional Software To Convert CPU Functions to GPU Functions for Parallel Execution

GPU - Graphical Processing Unit



GPU - Graphical Processing Unit

Strengths:

High Parallel Processing Capability: Efficient at handling multiple operations simultaneously, ideal for machine learning, scientific simulations, and complex calculations.

Speed: Can significantly improve processing times for suitable tasks by offloading them from the CPU.

Efficiency in AI: Key for rapid training of deep learning and AI models.

Versatility: Used across various fields beyond gaming, including scientific research and data analysis.

Energy Efficiency: More computations per watt compared to CPUs for specific tasks.

GPU - Graphical Processing Unit

Weaknesses:

Cost: High-end models are expensive, affecting both initial investment and software development.

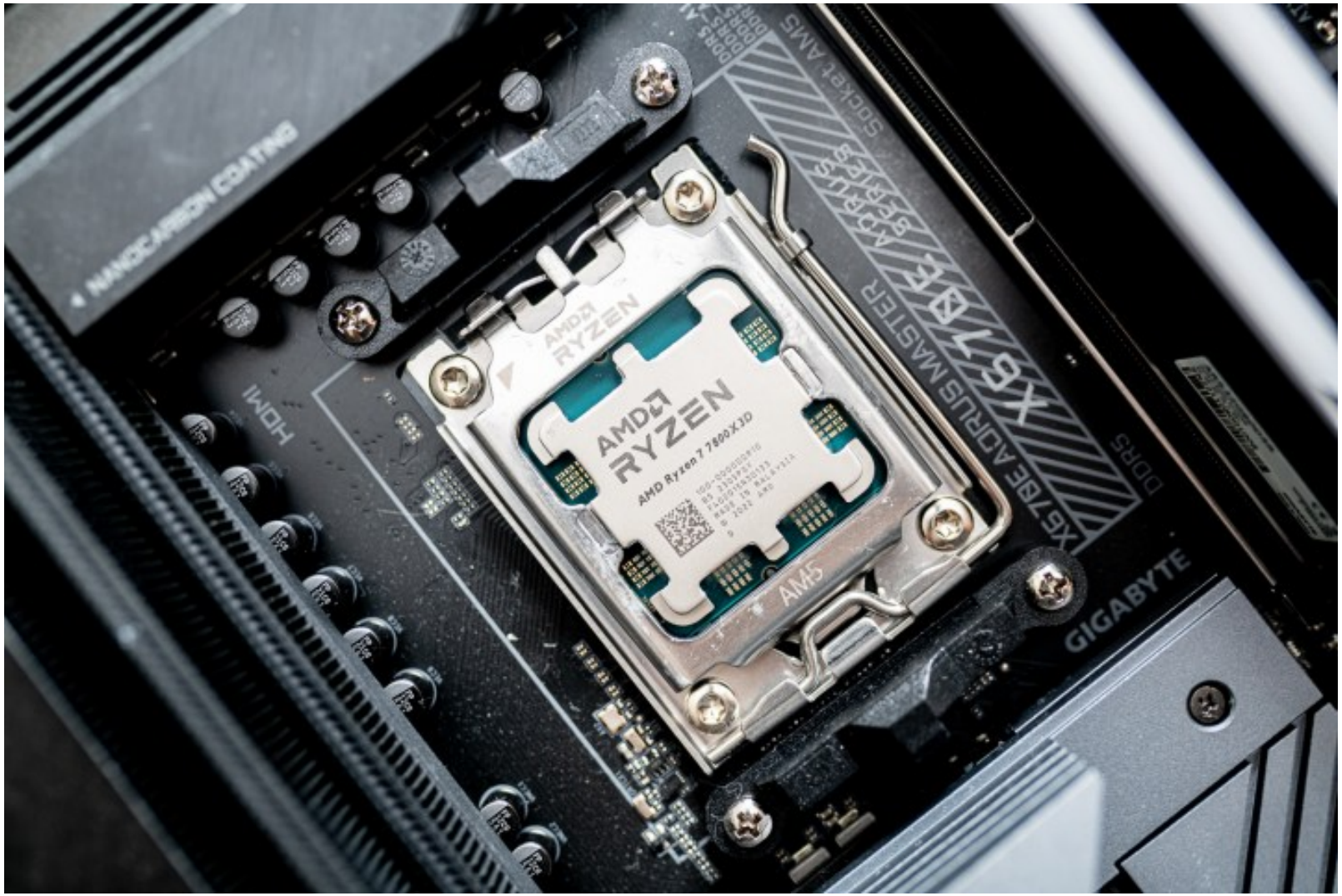
Power Consumption: Can be substantial, leading to higher operational costs and cooling requirements.

Complex Programming: Requires specialized skills to optimize software for parallel processing.

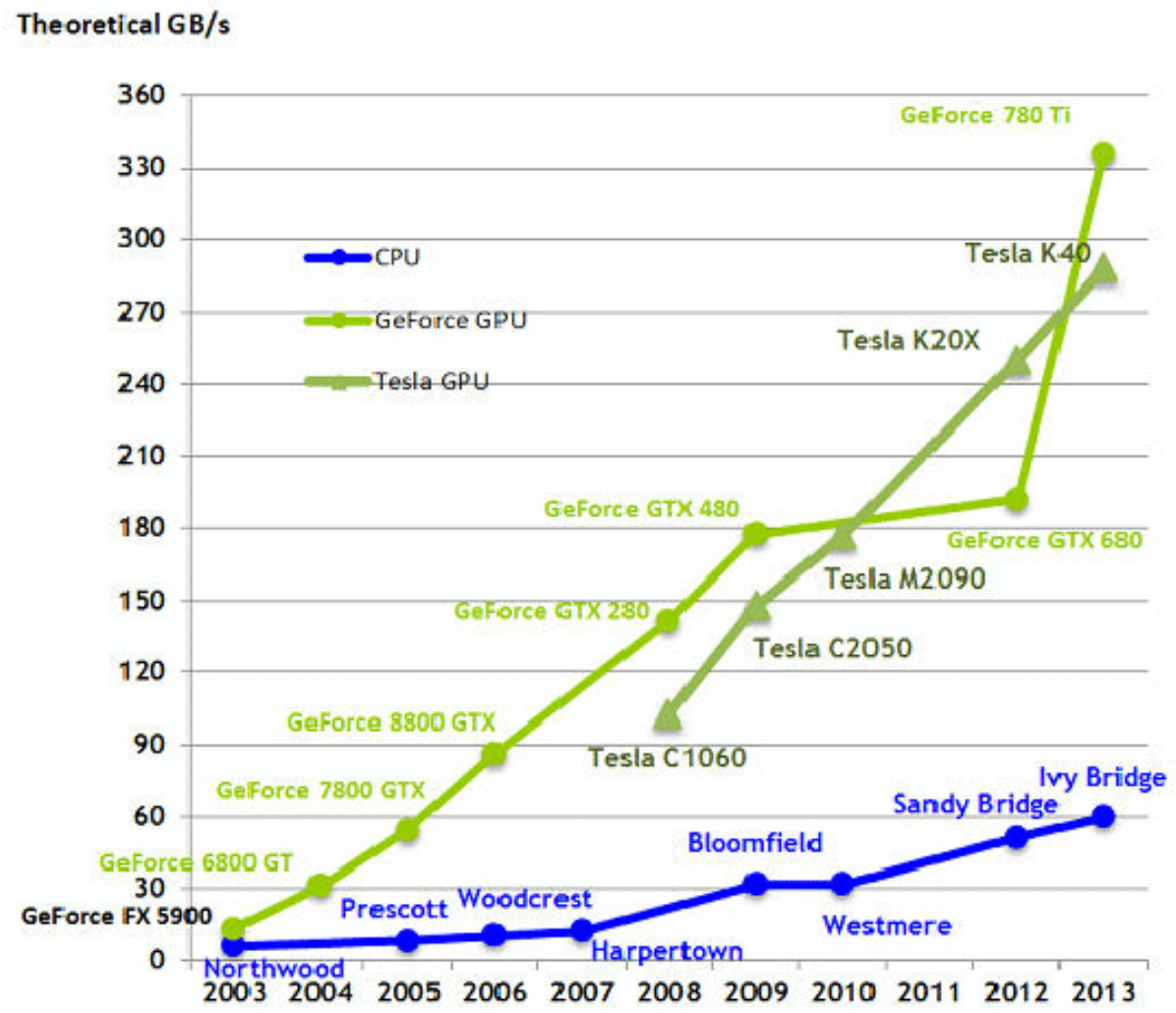
Task Suitability: Benefits primarily parallelizable tasks; not all operations see performance improvements.

Hardware Compatibility: Integration can be challenging due to physical, power, and driver compatibility issues.

CPU - Central Processing Unit



CPU - Central Processing Unit



CPU - Central Processing Unit

Strengths:

Versatility: CPUs are designed to handle a wide variety of tasks, from basic computing to complex data processing, making them highly versatile and suitable for general-purpose computing.

Ease of Programming: Compared to GPUs and FPGAs, CPUs are easier to program for, with a vast ecosystem of development tools and languages that support a wide range of applications.

Low Latency for Sequential Tasks: CPUs excel at handling sequential tasks and operations that require immediate attention, offering low latency in many cases.

Ubiquity and Compatibility: Being the standard computing resource in most devices, CPUs benefit from widespread compatibility with software and hardware, ensuring broad support.

CPU - Central Processing Unit

Weaknesses:

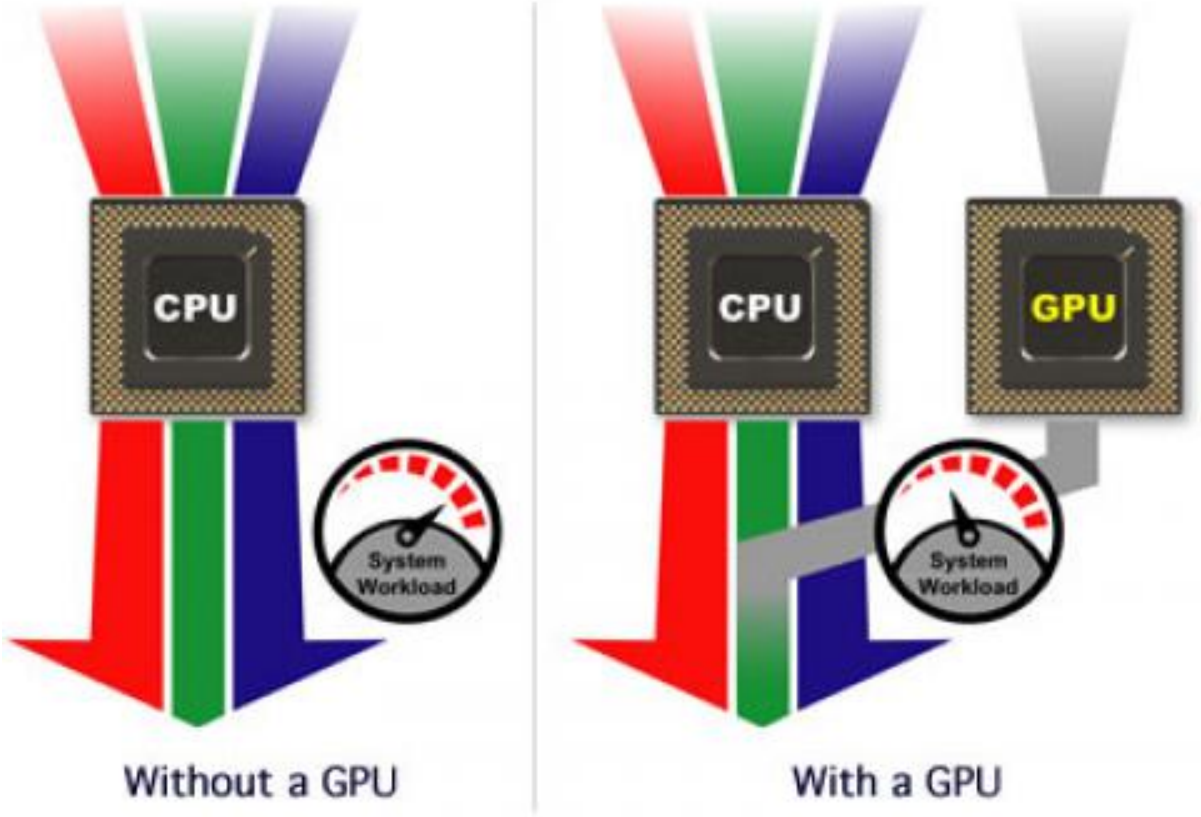
Limited Parallel Processing: While modern CPUs include multiple cores to handle several threads simultaneously, they are generally less efficient at parallel processing compared to GPUs and FPGAs for tasks that can be highly parallelized.

Energy Efficiency: When it comes to high-performance tasks, CPUs may not be as energy-efficient as specialized hardware designed for those purposes, leading to higher power consumption.

Performance Ceiling: For highly specialized tasks, such as deep learning or real-time video processing, CPUs may not offer the same level of performance as GPUs or FPGAs optimized for those operations.

Slow development time: Recent hardlimits in technological development due to die shrinkage and power constraints has slowed down the development of CPUs compared to other areas such as GPUs.

Hardware Acceleration



Hardware Acceleration

Hardware acceleration involves using specialized computing units like FPGAs, ASICs, GPUs, and CPUs to perform certain tasks more efficiently than a general-purpose CPU could on its own. This approach leverages the strengths of each type of hardware to optimize performance, power consumption, and functionality for specific applications.

Hardware Acceleration

Strengths:

Increased Efficiency: By offloading tasks to specialized hardware, systems can achieve greater computational efficiency and speed, reducing the workload on the CPU.

Optimized Power Consumption: Specialized hardware can often perform tasks more power-efficiently than a general-purpose CPU, leading to lower energy costs and heat generation.

Enhanced Performance for Specific Tasks: Each type of hardware can be optimized for specific tasks, such as data processing, graphics rendering, or complex calculations, providing superior performance where it matters most.

Flexibility and Scalability: Combining different types of hardware accelerators allows for a highly flexible and scalable system that can be tailored to the needs of various applications.

Hardware Acceleration

Weaknesses:

Complexity in Integration: Integrating various types of hardware accelerators into a single system can be complex, requiring specialized knowledge and software support.

Increased System Cost: The use of specialized hardware components can increase the overall cost of a system, both in terms of the hardware itself and the development effort required.

Software Compatibility and Development: Optimizing software to leverage hardware acceleration can be challenging, requiring specific programming skills and sometimes extensive modification of existing code.

Limited by Specific Use Cases: While hardware acceleration can offer significant benefits for certain tasks, not all applications will see performance improvements, and some may be better served by general-purpose computing resources.

Hardware Acceleration Examples

1. Video Processing and Streaming

- **Hardware**: GPUs, ASICs
- **Application**: Efficient real-time video encoding/decoding, adding video effects, and live streaming. GPUs excel in parallel processing of video data, while ASICs are used for dedicated video processing tasks.

2. Data Encryption and Security

- **Hardware**: FPGAs, ASICs
- **Application**: Accelerating encryption algorithms for securing data in transit and at rest. FPGAs offer flexibility for evolving encryption standards, whereas ASICs deliver optimized performance for specific algorithms.

Hardware Acceleration Examples

3. Machine Learning and AI

- **Hardware**: GPUs, FPGAs, ASICs (e.g., Google's TPUs)
- **Application**: Training and inference phases of machine learning models. GPUs are preferred for training due to parallel processing capabilities. FPGAs and ASICs are chosen for inference, providing lower latency and reduced power consumption.

4. High-Frequency Trading (HFT)

- **Hardware**: FPGAs
- **Application**: Executing financial transactions with ultra-low latency. FPGAs enable microsecond-level market data analysis and trading actions, crucial for HFT.

Hardware Acceleration Examples

5. Scientific Research and Simulations

- **Hardware**: GPUs, High-Performance CPUs
- **Application**: Conducting complex simulations in fields like physics, chemistry, and biology. GPUs support extensive mathematical model computations, while high-performance CPUs are used for sequential processing tasks.

6. Blockchain and Cryptocurrency Mining

- **Hardware**: ASICs, GPUs
- **Application**: Performing blockchain computations and mining cryptocurrencies efficiently. ASICs provide top efficiency for specific algorithms (Bitcoin mining), and GPUs offer versatility for mining a variety of cryptocurrencies.

Hardware Acceleration Examples

7. Network Packet Processing

- **Hardware**: FPGAs, ASICs
- **Application**: Enhancing network packet inspection, routing, and switching. FPGAs offer protocol flexibility, and ASICs are implemented in high-end routers for maximal throughput.

8. Automotive Advanced Driver-Assistance Systems (ADAS)

- **Hardware**: GPUs, ASICs
- **Application**: Processing sensor inputs and making real-time decisions in autonomous and semi-autonomous vehicles. GPUs are used for sensor data parallel processing, and ASICs handle specific functions like image recognition.

Del 2 - Arduino Installation & Example run

Go to https://www.arduino.cc/en/software

For documentation of reprogramming the Arduino, google the ATMega328P Datasheet or go to https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

Arduino

Vad sker?

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

Arduino

Vad sker...nu?

```
void setup() {
  // Set pin 13 as an output by setting the 5th bit of DDRB
  // Pin 13 is mapped to bit 5 of the DDRB register on the ATmega328P
  DDRB |= (1 << 5);
}

void loop() {
  // Set pin 13 high (turn on LED) by setting the 5th bit of PORTB
  PORTB |= (1 << 5);
  delay(1000); // Wait for a second

  // Set pin 13 low (turn off LED) by clearing the 5th bit of PORTB
  PORTB &= ~(1 << 5);
  delay(1000); // Wait for a second
}
```

Vad sker.....nu?

```
// Define the memory addresses for DDRB and PORTB registers
volatile uint8_t* ddrb  = (volatile uint8_t*)0x24;
volatile uint8_t* portb = (volatile uint8_t*)0x25;

void setup() {
  // Set pin 13 (PB5) as an output
  *ddrb |= (1 << 5);
}

void loop() {
  // Set pin 13 high (turn on the LED)
  *portb |= (1 << 5);
  delay(1000); // Wait for a second

  // Set pin 13 low (turn off the LED)
  *portb &= ~(1 << 5);
  delay(1000); // Wait for a second
}
```