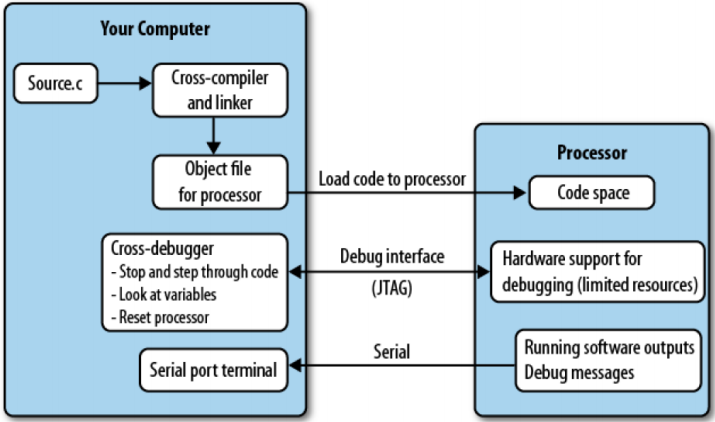
**Embedded system:** is a computerized system that is purpose-built for its application. Computerized: computation is core Purpose-built: software and hardware components. Application: functionality, requirement, specification

**Designed** with specific quality requirements-- **Deterministically**: exactly the same each time --**Real time**: always reacting to an event fast enough− **Fault tolerant**: tolerable quality degradation in the face of errors ex: Unmanned aviation vehicle (UAV) − **Agile:** instant change of operations on a sudden signal ex: Uninterruptible power supply (UPS)

**Design case** − Application: obtain power consumption data in computer − Embedded system: power sensor in computer. Keep development time as low as possible (well established libraries, documents, tools). Deploy, update and maintenance: Remote access only for update and maintenance − Need MCU that supports remote control and update − Need software including remote control and update.

**Challenges:** Resource constraints - Memory and code space, Processor cycles and speed, Power consumption, Peripherals. Compromises: tradeable resources: code space vs processor cycles. Processor speed vs power consumption. Can have buggy hardware. Cost for hardware: design, PCB manufacturer, board assembly, parts cost, shipping, custom and tax. Cost of Software: licenses, software development, testing. Need to be able to support updating.

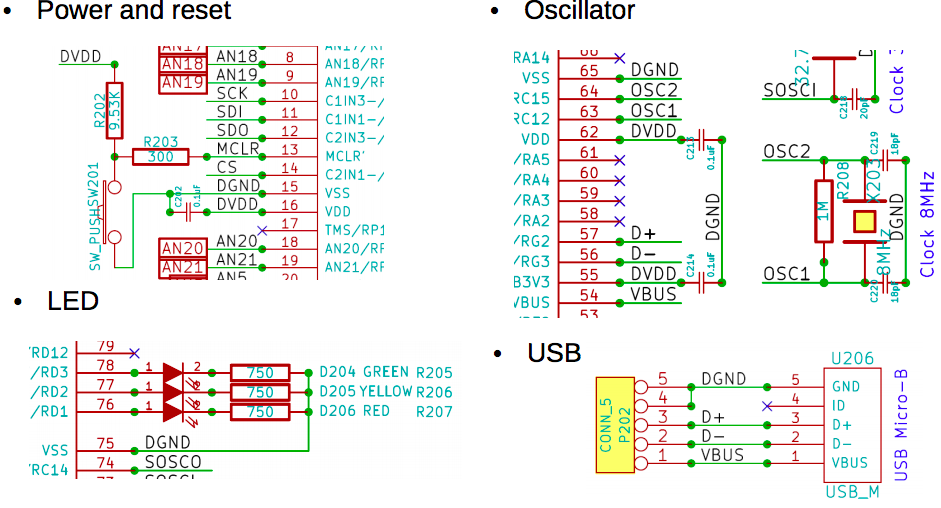
**System Stack:** Firmware (run in device)- libraries include standard libraries and system-specific libraries. CS: App, Lib, OS

CE: Lib, OS, Driver EE: Driver, Hardware

**Application development**:- Development on top of existing drivers and OS with given APIs - No direct programming interface to hardware – Develop with a well established software development kits (SDK) -Android, iPhone - Arduino, Raspberry Pi

**System Development:** Need to develop OS, system-specific libraries, and applications. Customized board. Develop on bare metal

**Cross-Compiler:** can convert instructions into machine code or low level code for a computer other than that on which it is run

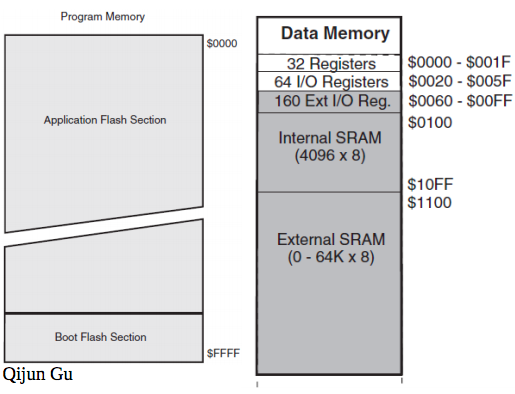
**Cross-Debugger:** The debugger sits on your computer and communicates with the target processor through the special processor interface. The interface is dedicated to letting someone else eavesdrop on the processor as it works. The processor must expend some of its resources to support the debug interface, allowing the debugger to halt it as it runs and providing the normal sorts of debug information. Often called JTAG

**Compiler:** Source code -> executable – Preprocess -> source code – Compile -> object code – Link -> executable code ● Host : where a program is compiled ● Target : where a program is executed ● Host = Target in regular computer ● In embedded system – Host is a regular computer – Target is the embedded system – Cross-compilation : avr-gcc or xc16-gcc

**Command line:** xxx-gcc sourcefiles options -o outfile : file to generate -c : compile only

-Wall : show all warnings -g : include debugging information -O[0-3s] : optimization -mmcu=tgt or -mcpu=tgt : target processor -Dmacro : additional macro -Idir : additional include directory -Ldir : additional library directory -llib : additional library -Wl,[lflags] : linking flags

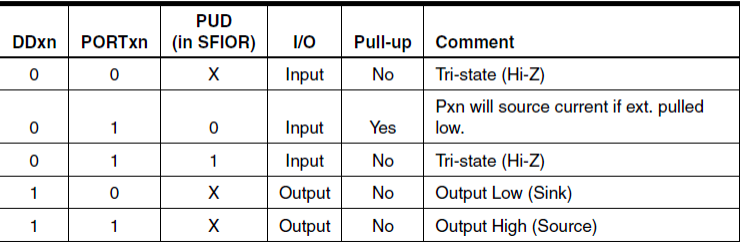
**Software Team:** After components are selected, − Processor is decided. − Tool chain: compiler, debugger, loader. Ideally in Integrated Development Environment (IDE) Licensed: most chip makers provide dev kits. Free: gcc-based open source tool chains. After schematic is completed, − Read schematic and datasheets − Read documentations − Develop software in dev kits and prototypes.

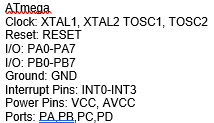
**System Architecture: 1 von Neumann** − A processing unit − A memory to hold both instructions and data − CPU can either read an instruction or access data from the memory, but not at the same time

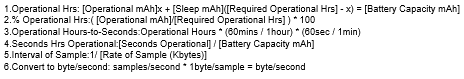
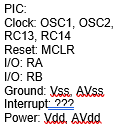
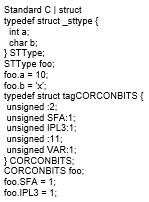
**2 Harvard** − A processing unit − Two memories to hold instructions and data separately − CPU can both read an instruction and access data at the same time

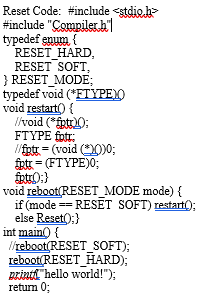
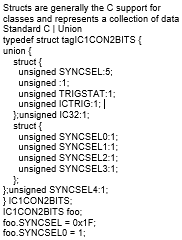
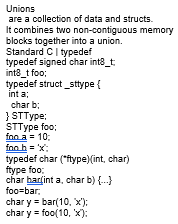
**IO architecture: 1 Port IO** − IO address is not memory address − IO data path is separated from memory data path − Special IO instructions − Simultaneous access to IO and memory

**2 Memory mapped IO** − IO address is a part of memory address − IO and memory share the same data path − No special IO instructions − Exclusive access to IO or memory



Output for Atmega LEDs − From schematic: Port D0..2 − Three register bits of Port D. DDRD : direction control − 0 : input, 1 : output PORTD : data output − 0 : off, 1: on PIND : data input





Typedef is used to assign an alias to

primative types. You can use

typedef

to

give a name to your user defined data

types as well.

Standard C | enum

typedef enum {

ST\_OFF = 0,

ST\_ON = 1,

ST\_SAMPLE = 2,

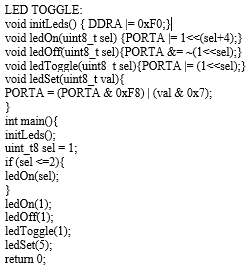
ST\_PROCESS = 3,

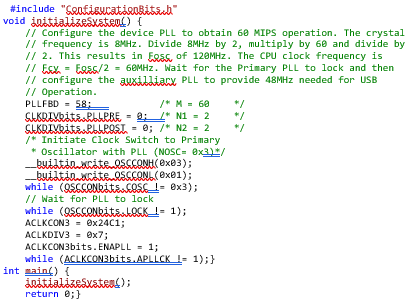
ST\_PAUSE = 4,

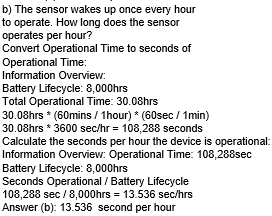
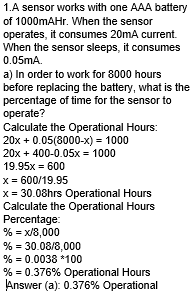
} STATUS;

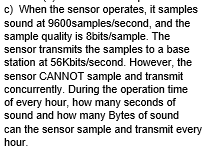
STATUS foo;

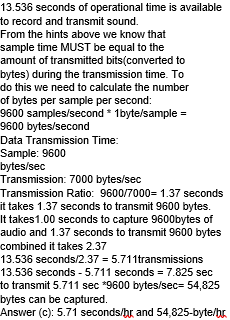
foo = ST\_OFF











Assign by bit:

struct { unsigned SYNCSEL0:1;

unsigned SYNCSEL1:1;

unsigned SYNCSEL2:1;

unsigned SYNCSEL3:1;

unsigned SYNCSEL4:1; };

Extern -