Firmware System Design:
Development Environments, Toolchains, and Deployment

1. Integrated Development Environments (IDE)
2. Configuring Toolchains
3. Deploying Firmware

Firmware Development - Reflection

Firmware Systems Design will be used as an example.

Self-Reflection

What experience have you had with setting up a Development Environment?

After the last lecture, what do you already know about how code is built?

What methods have you already used for deploying firmware?

Systems Design - Firmware Development

1. Integrated Development Environments (IDE)
   - Text Editing - Code Specific Formatting
   - File / Project Manager
   - Integrated Macros for Code Building
2. Configuring Toolchains
   - AVR/GNU C Compiler
   - Linking in Libraries
   - Build Parameters
   - Selecting Optimisation Parameters
3. Deploying Firmware
   - Vendor-Specific Debuggers and Programmers
   - Bootloading
4. Firmware Version Management
   - Version Management Tools
   - Backup

Integrated Development Environments (IDE) Introduction

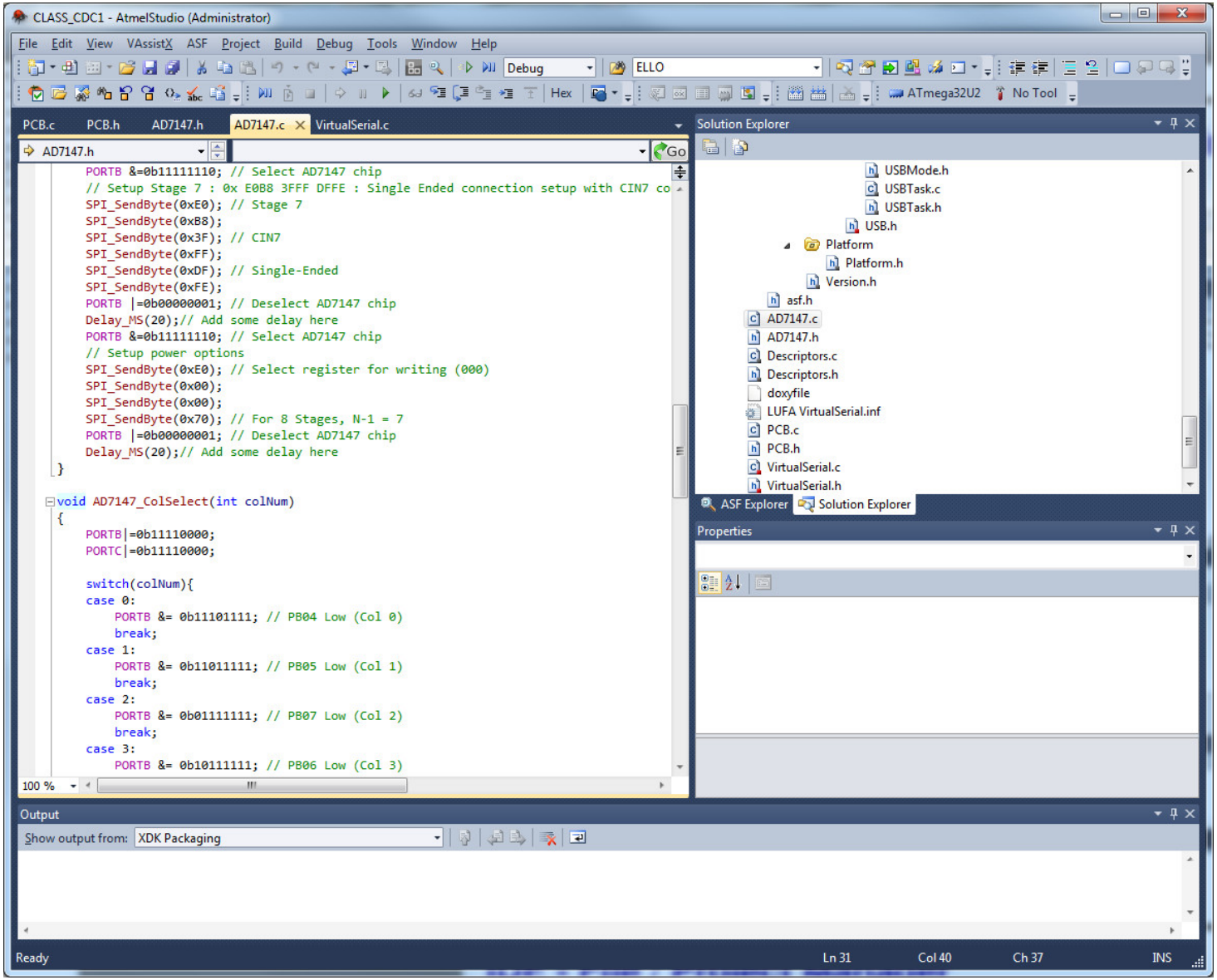
An Integrated Development Environment (IDE) has many built in features that make code writing, file management and building a comfortable process

Whether code is for a PC or a microcontroller, IDEs have similar functionality and workflow

Like Notepad++ and many other text editors, the ones used in IDEs typically have language-specific highlighting and formatting to make code easier to follow

Buttons and keyboard shortcuts facilitate seamless building and deployment of code

IDE - Text Editing - Layout and Code Specific Formatting



IDE - File / Project Manager

A coding project typically involves a large number of files that are arranged in a carefully organised folder hierarchy

A project file keeps track of the state of each of the code files and their relative locations. This functionality allows the compiler to work efficiently and only update files that have been changed since the last build

The AtmelStudio IDE (pictured on earlier slide) is based on the Visual Studio IDE for writing PC programs. Note that most IDEs have a similar layout and workflow

The file tree at the right-hand side facilitates quick look-up and the file tabs at the top of the editor window allow quick switching between files

IDE - Integrated Macros for Code Building

Depending on the IDE, or your development method, there are different ways to initiate program building. Eg. Notepad++ can be an effective code editor, which can have plugins installed for this functionality

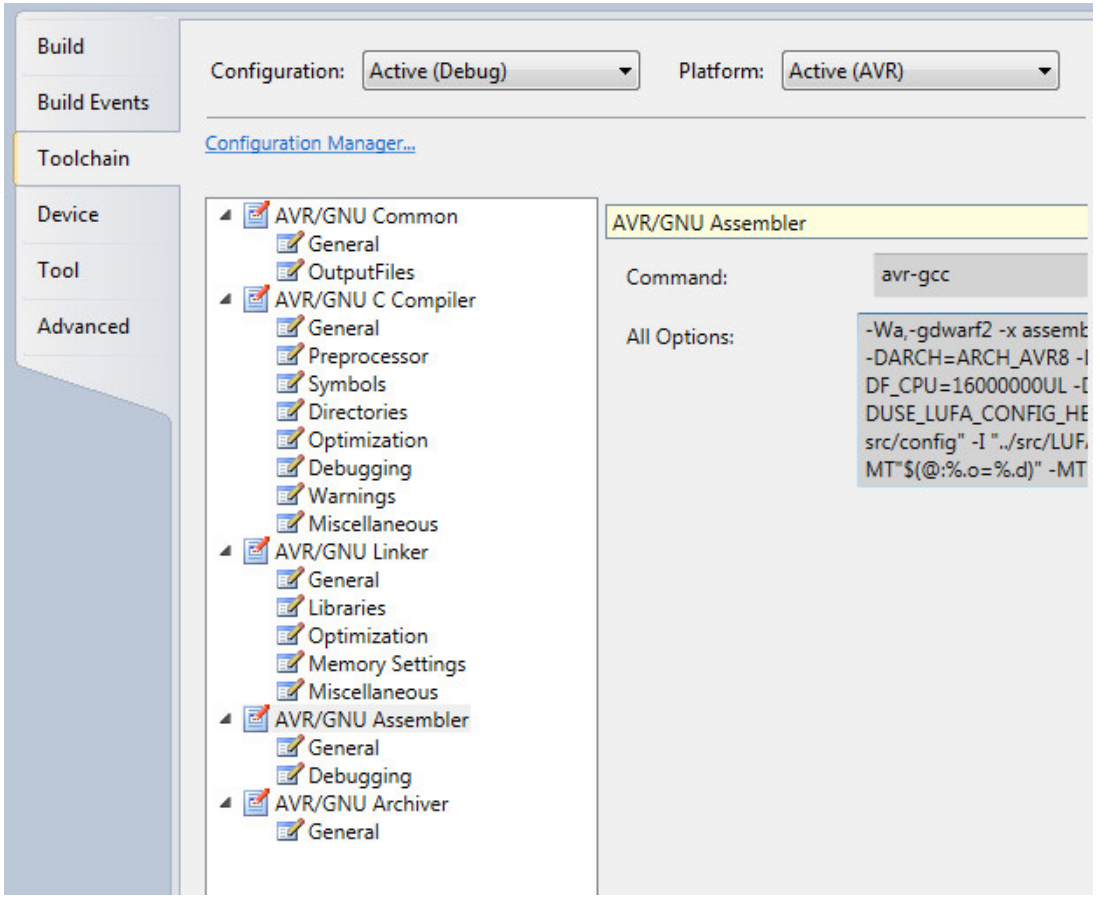
Typically the macros run internal, or external, programs to preprocess, compile, link, assemble, and even deploy the program to the microcontroller

Buttons and shortcuts initiate small snippets of code that are typically run in a hidden command line format

Macros can be written for "cleaning" the solution (clearing out all build artefacts), or performing custom functionality (Eg. Running your own program)

Introduction to Configuring Toolchains

The set of software tools for building the solution, including compilers, linkers, and assemblers are called a toolchain; they perform sequential functions (links) of the build from start to finish.



Configuring Toolchains - AVR / GNUC Example

Options in the AVR/GNU C Compiler include:

- Definition of signed or unsigned char variables
- Definition of Symbols for specific hardware configurations (Eg. F_CPU=8000000UL specifies that the CPU frequency is 8MHz)
- Configuring which directories to search for code files
- Configuring the level of optimisation (Eg. -O1 is for basic optimisation)
- Configuration of warnings and language variables (Eg. -std=gnu99 for C99 Standard compilation)

Configuring Toolchains - Linking in Libraries

Standard libraries for mathematics or string functinalities can be linked in. If the libraries are not in the already searched paths, they may have to be added.

Memory use can be altered. Recall the different architectures: Harvard and Von Neumann (Princeton)

The AVR is a Harvard architecture CPU. This means that it separates instruction memory and data memory. The gcc was originally designed to support Von Neumann architectures which define a single storage structure to hold both instructions and data. This dichotomy is solved by a series of nifty tricks in the AVR port of gcc.

Configuring Toolchains - Assembler

Finally, when the code has been compiled and structured for the memory architecture, the assembly phase can be undertaken. This finalises the firmware, making it ready for deployment.

https://www.flickr.com/photos/visualpanic/2145968772

Deploying Firmware

Once firmware is written and built, what methods are available to have it flashed into the processor?

How does in circuit programming work?

How do different chip vendors achieve this?

What are the advantages and limitations of these techniques?

Deploying Firmware - Vendor-Specific Debuggers and Programmers

Some vendors have proprietary programmers and in-circuit debuggers that are designed to be able to halt the program at different points and allow for feedback from the device.

MPLAB ICD3

http://sigma.octopart.com/21262504/image/Microchip-DV164035.jpg

ATMEL JTAG REAL ICE

http://www.ecvv.com/product/2534819.html

Deploying Firmware - JTAG Interface

The JTAG interface is more than a hardware footprint: it is a 4-wire interface that can be used to test hardware connections and program microcontrollers

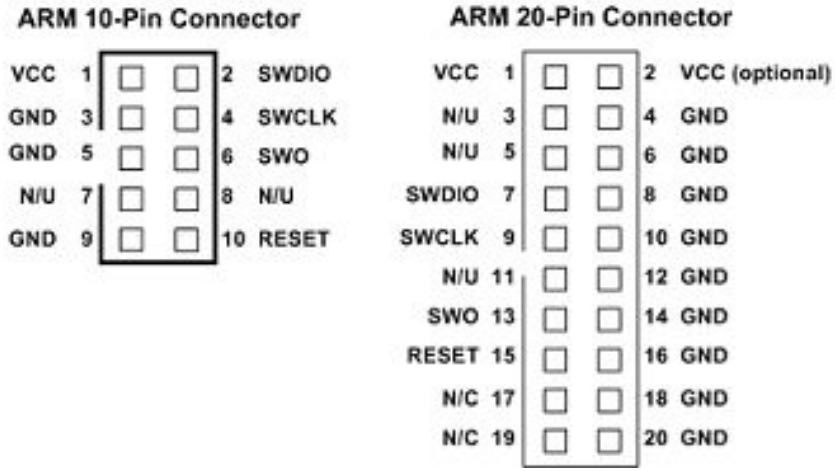
Hardware test features include reconciliation that connections reflect a schematic

Testing physical connections for chips that have inacessible pads (Eg. Ball grid array format.)

Microcontrollers that are not responding at a code level of fail to boot may be recovered through this interface

Standardised footprint and communication interface make the technique portable across different chip vendors

Deploying Firmware - JTAG Interface



http://www.keil.com/support/man/docs/ulink2ulink2_connector_20_10pin_swm.png

This layout method allows for a single programmer to be used for many applications

Further reading:

http://www.xjtag.com/support-jtag/what-is-jtag.php

Deploying Firmware - Bootloading

Bootloading is the act of deploying code through a standard communication interface

The bootloader code occupies a small amount of memory, and when initiated listens for communications on a particular interface. The bootloader overwrites the main program in the Program Memory

Bootloading has the advantage that no hardware tool is required to flash firmware into the device. This means that firmware can be updated by lay people in the field. The main disadvantage of bootloaders is that they occupy a small portion of program memory

An example bootloading tool for Atmel ATMEGA chips is called FLIP, which loads code over USB.

Firmware Version Management

Version management is important when making modifications to large code-bases, or working with more than one person on the project at a time

Specialist tools are designed to keep track of changes, and allow different versions to be developed and tested individually

There are different models: Client/Server, or distributed, which each have advantages and disadvantages.

Open Source examples of these types include: Subversion(client/server) - Git,Mercurial (distributed)

Further reading:
`https://en.wikipedia.org/wiki/List_of_revision_control_software`

Firmware Version Management

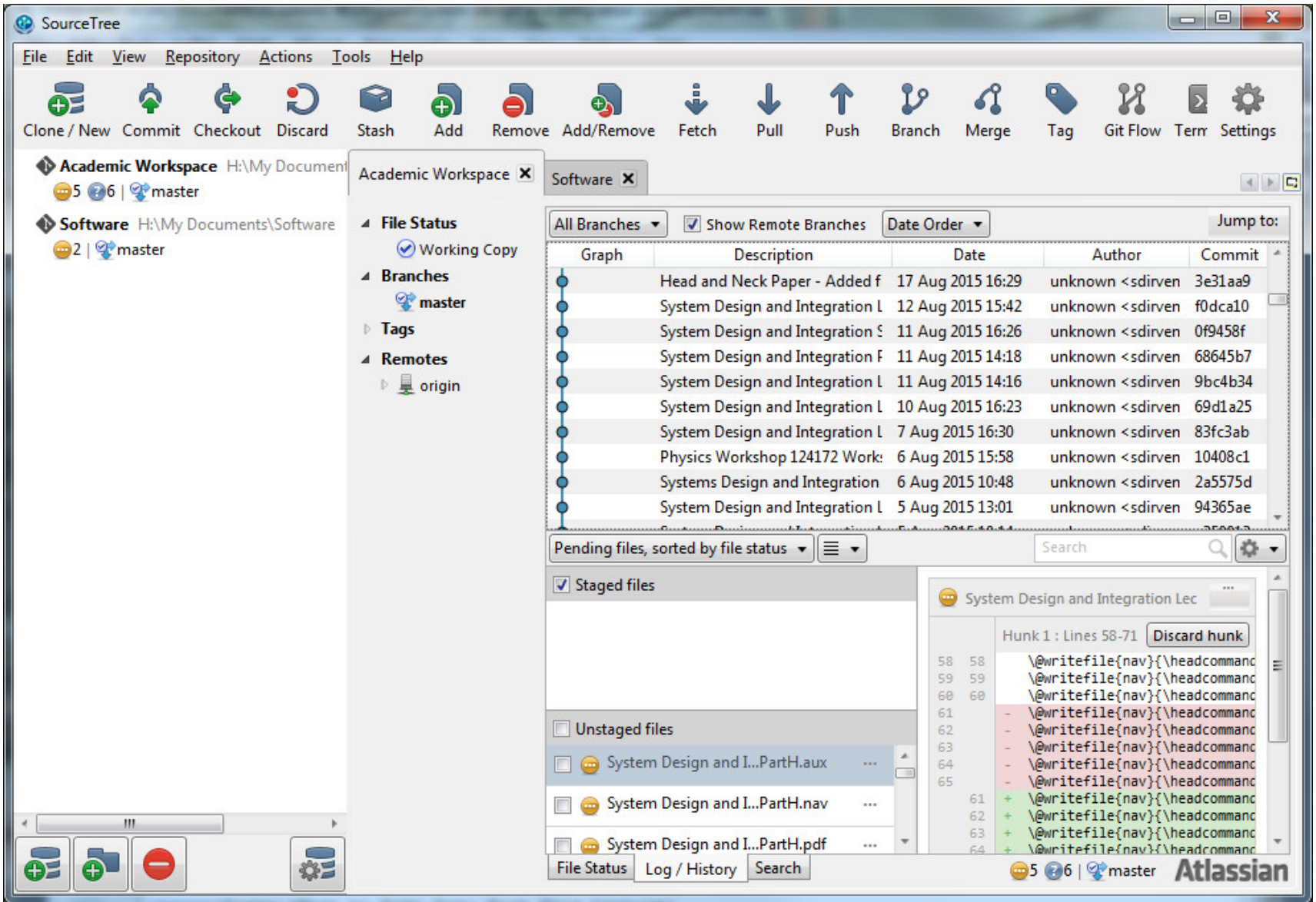
Version management is a good method of backing up software, and keeping track of changes

These softwares typically have an online interface, as well as downloadable user interfaces. (Eg. Sourcetree) Repositories keep a snapshot of all changes to files at each "commit", such that you can go back to a previous version

The Trunk is where the latest version of good software is stored

Branches are where alterations or changes are being made that are incomplete

Firmware Version Management



What to Take Away From This Lecture

The Main Points

- Integrated Development Environments (IDE) present software projects in a user-friendly manner, manage the file structure, facilitate building and deployment of solutions
- A toolchain consists of a Compiler, a Linker, and Assembler that work in harmony to generate the final machine code file (1's and 0's)
- Firmware can be deployed by vendor-specific programmers, JTAG, or bootloaders, each which have their own advantages and limitations
- Code backup and version management are important considerations when working with multiple files, or developers