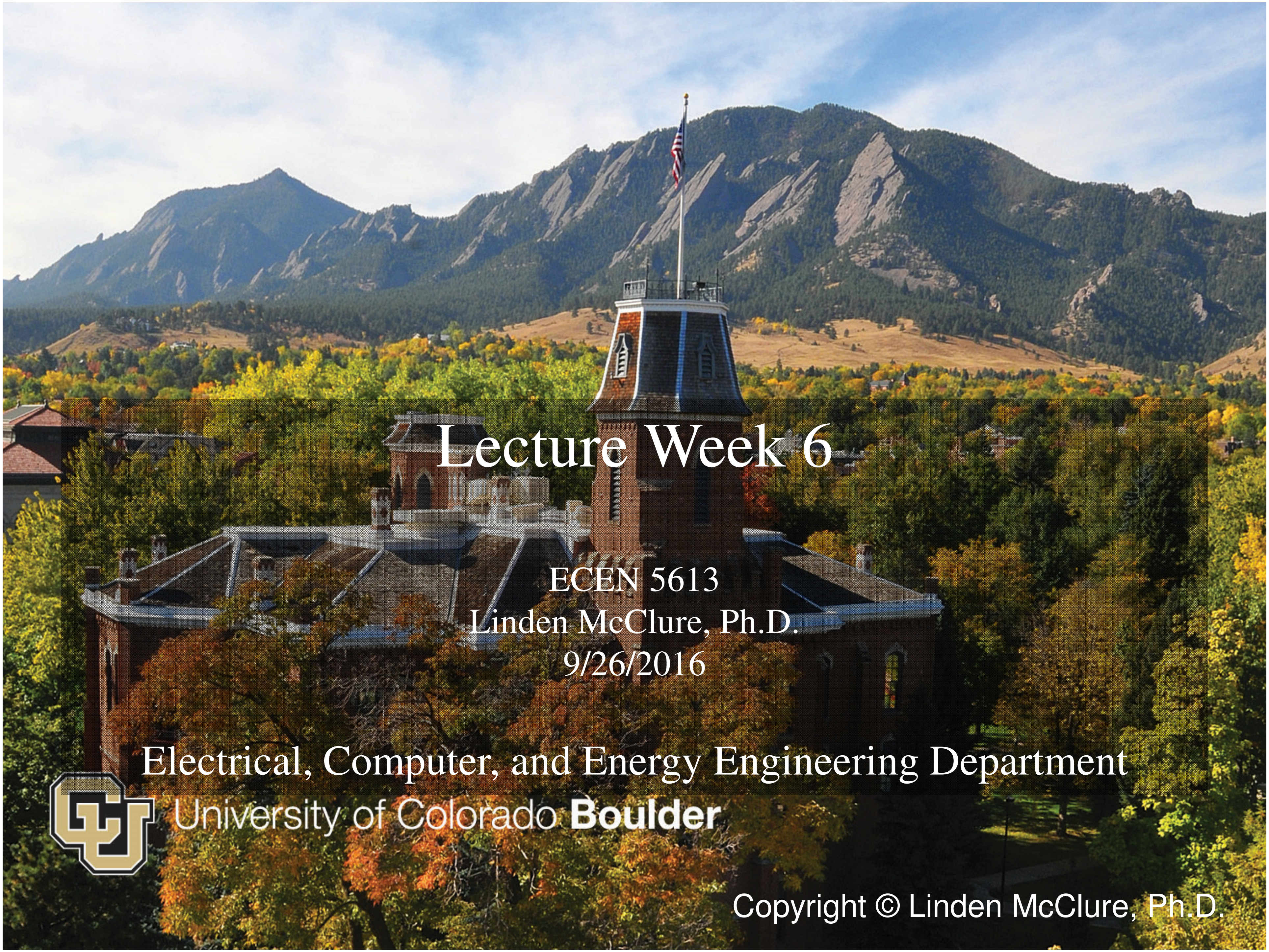


Lecture Week 6

ECEN 5613

Linden McClure, Ph.D.

9/26/2016

Electrical, Computer, and Energy Engineering Department
University of Colorado Boulder

Outline

- CTP Topic Assignments
- Lab #3 Topics
  - Overview of Software Development
  - Paulmon2
  - Introduction to Programming with SDCC
  - Make
  - Code::Blocks
  - Subversion
- Final Project Ideas

Announcements

- Should have RS-232, Atmel CPU, FLIP, Paulmon, SDCC by Friday, 10/07
  - Start with Spring 2016 Lab #3 up through first SDCC exercise (item 16)
  - An updated Lab #3 will be posted by around 10/05
- Clean up lab benches, turn off soldering irons
- Try to get labs signed off during office hours before the due dates
- Student questions:
  - /RD signal and SPLD pin-keeper
  - LogicPort threshold setting

Upcoming Dates of Interest

- 9/26: CTP topic assignments.
- 9/26: lecture in ECCS 1B14 – last lecture for Lab #3
- 9/28: signature due date for Lab #2 Supplemental Elements
- 9/29: submission due date for Lab #2 - submit electronically via Desire2Learn
- **9/30: deadline for completing Honor Code Quiz on Desire2Learn**
- 10/03: TBD – no lecture? Student/Instructor meetings regarding CTP/projects?
- 10/10: TBD – no lecture? Debug session for Lab #3
- 10/14: signature due date for Lab #3 Required Elements
- 10/17: Week 8 lecture in ECCS 1B14 – only lecture for Lab #4
- 10/19: signature due date for Lab #3 Supplemental Elements
- 10/23: PowerPoint submission for Final Project PDR
- 10/24: Week 10 PDR during lecture in ECCS 1B14
- 10/31: CTP submissions, lecture in ECCS 1B14

Freescale's Tiny Kinetis KL03 MCU

2/25/2014

Freescale (NXP) has announced the Kinetis KL03 MCU, which it calls the "world's smallest" 32-bit MCU based on ARM technology. The Kinetis KL03 comes in a chip-scale package that measures 1.6 x 2.0 mm -- smaller than a dimple on a golf ball.

Based on the ARM Cortex-M0+ architecture -- the smallest, lowest power ARM core -- the Kinetis KL03 provides the 32-bit processing power required to support complex algorithms, connectivity stacks, and sophisticated human-machine interfaces. Pricing is $0.52-0.85 (USD) in 10,000 unit quantities.

Source: http://www.eetimes.com/document.asp?doc_id=1321173

Current Topics Presentation – CTP

Technology Topics

Possible Technology Topics

1. ARM topics
2. Battery technologies
3. Battery management - charging and fuel gauge functions
4. Supercapacitors, ultracapacitors
5. Sensors: Sensor Hub, eCompass, Accelerometer, Gyroscope, Ambient Light Sensor, thermal, SAR, Location/position (GNSS/GPS/GLONASS)
6. Wifi – 802.11a/b/g/n, 802.11ac, 802.11ad, antenna diversity, 1x1, 2x2
7. Touch screens, touch controllers, pen technology
8. USB BC 1.2 SDP, CDP, OTG
9. Storage - eMMC, UFS
10. Power Management - PMIC technology
11. Switching regulators, shoot through current on FETs
12. Security, SW/HW TPM, authentication chips
13. Specific motors and motor drivers
14. ESD protection circuits
15. Camera sensors and interfaces
16. MIPI protocol – MIPI-DSI, MIPI-CSI
17. SPI
18. SWI, one wire interface
19. RFID protocol and technology
20. Zigbee protocol
21. BLE – Bluetooth Low Energy protocol and devices
22. Rad Hard – radiation hardened technology and devices
23. Automotive embedded systems – technology requirements and implementation considerations
24. Other ______________________

CTP Topic Assignments

	Name	Preferences	Assignment
1	Dubey, Ravi	23, 2, 5	23. Automotive embedded systems – technology requirements and implementation considerations
2	Dutta, Subhradeep (Subhz)	24, 15, 5	15. Camera sensors and interfaces
3	Gandhi, Gaurav	23, 5, 19	5. Sensors: Sensor Hub, eCompass, Accelerometer, Gyroscope, Ambient Light Sensor, thermal, SAR, Location/position (GNSS/GPS/GLONASS)
4	Gnanasekaran, Praveen	21, 5, 7	21. BLE – Bluetooth Low Energy protocol and devices
5	Gundepally, Ashwath (Ash)	23, 12, 5	23. Automotive embedded systems – technology requirements and implementation considerations
6	Halambi, Akash	23, 5, 13	5. Sensors: Sensor Hub, eCompass, Accelerometer, Gyroscope, Ambient Light Sensor, thermal, SAR, Location/position (GNSS/GPS/GLONASS)
7	Kalyani Vijaya Kumar, Karthik (KV)	1, 6, 21	1. ARM topics
8	Khare, Srishti	23, 5, 7	7. Touch screens, touch controllers, pen technology
9	Lobo, Nestor	20, 5, 9	20. Zigbee protocol
10	Noronha, Richard	23, 7, 10	10. Power Management - PMIC technology
11	Pingali, Venkateswara (Kalyan)	19, 20, 23	20. Zigbee protocol
12	Sabharwal, Saksham	23, 21, 5	21. BLE – Bluetooth Low Energy protocol and devices
13	Sali, Rohith	23, 5, 7	7. Touch screens, touch controllers, pen technology
14	Sampath Kumar, Divya	16, 19, 22	19. RFID protocol and technology
15	Shah, Akshit	23, 5, 19	5. Sensors: Sensor Hub, eCompass, Accelerometer, Gyroscope, Ambient Light Sensor, thermal, SAR, Location/position (GNSS/GPS/GLONASS)
16	Singh, Tarun	17, 19, 20	19. RFID protocol and technology
17	Srivastav, Vishal	1, 20, 21	1. ARM topics
18	Sunil Kumar, Vijoy	10, 23, 5	10. Power Management - PMIC technology
19	Venkatesan-Kulathu-Sundaram, Bhallaji	16, 15, 18	15. Camera sensors and interfaces
20	Weiss, Alec	10, 21, 6	6. Wifi – 802.11a/b/g/n, 802.11ac, 802.11ad, antenna diversity, 1x1, 2x2
21	White, Timothy	24, 21, 22	24. LiFi – Optical Wireless Communication
22	Burin, Micheal	17, 1, 7	1. ARM topics
23	Jacobus, Joey	6, 7, 23	6. Wifi – 802.11a/b/g/n, 802.11ac, 802.11ad, antenna diversity, 1x1, 2x2

CTP Topic Assignments

Some reference presentations will be posted on D2L over the coming week.

CTP presentations will be done in class on 10/31, 11/07, 11/14, and 11/28.
I would like 2-3 presentations ready by 10/30.

Overview of Software Development

Compilers, Operating Systems, and Development Tools

Embedded Cross Compilers

Why use an embedded cross compiler?

- High-level languages are based on human readable logic expressions vs. assembly, which is designed to be machine interpretable
- Processor-specific knowledge requirements are lower than with assembly
- Compiled languages allow reuse of software from different architectures
- These attributes reduce development time improve coding accuracy, maximize code reuse and optimize maintainability

Source: Scott Hoot

Embedded Cross Compilers
Why not always use a cross compiler?

- Execution speed and code size often improve with "hand-coded" assembly routines
  - Assembly is limited only by the capabilities of the processor architecture and skill of the developer
  - Compilers must be accurate to the language
- Some operations are not available in language standards
  - Overcome with compiler "extensions" and/or inline assembly
  - Breaks code portability

Embedded Cross Compilers
How to select a cross compiler?

- Target support
- Host requirement
- Features
- Optimizations
- Cost
- Experience
- Source portability and maintainability

Embedded Cross Compilers
8051 specific tips for using SDCC

- Deviations from ANSI C
  - '_interrupt' keyword
  - '_bit' and '_sfr' variables
  - '_data', '_xdata', '_idata' and '_pdata' data locations
  - '_code' storage location
  - '_asm' and '_endasm' inline assembly keywords
  - Interfacing with assembly code
- Compile and Link Options
  - 'small', 'medium' and 'large' memory models
  - 'code_loc' and 'xram_loc' linker directives

Embedded Cross Compilers
8051 specific tips for using SDCC (cont.)

- Headers and Libraries
  - 8051 Registers (8052.h or reg51.h)
  - Standard C Lib (stdio.h)
  - Memory allocation (malloc.h)
  - Serial port (ser.h and serial.h)
  - Floating point math (math.h and float.h)
- Optimizations
  - ‘naked’ functions
  - ‘using’ keyword
  - 16- and 32-bit math

Atmel Include File for SDCC

```
/****************************************************************************
* NAME:          at89c51RC2.h
*----------------------------------------------------------------------------
* PURPOSE:
*   This file defines Sfr registers and BIT Registers for AT89C51RC2
```

> This Atmel file doesn't work.
> Instead, use the standard SDCC header file:
> C:\Program Files\SDCC\include\mcs51\at89c51ed2.h
> Note the at89c51ed2.h file has an error with the definition of RBCK.

```
/*----------------------------------------*/
/* Include file for 8051 SFR Definitions  */
/*----------------------------------------*/

/*  BYTE Register  */
Sfr (P0 , 0x80);

Sbit (P0_7 , 0x80, 7);
Sbit (P0_6 , 0x80, 6);
Sbit (P0_5 , 0x80, 5);
Sbit (P0_4 , 0x80, 4);
Sbit (P0_3 , 0x80, 3);
```

<continued>

Operating Systems
Why use an Embedded Operating System?

System Management

- CPU Scheduling

  Provides infrastructure to enable preemptive multi-tasking of the CPU

- Memory Management

  Heap, stack and register management are further abstracted

- IO Abstraction

  A consistent interface for IO services is provided through the use of drivers

Operating Systems
What is the downside of using an OS?

- OS Activities
  - Task switching
  - CPU scheduling
  - Delayed interrupt procedures
  - Memory management
  - I/O abstraction
- System Results
  - Increased program and data memory usage
  - Increased execution latencies and overhead

Operating Systems
How to select an Operating System?

- Experience
- Cost
- Features
- Performance
- Size
- Drivers
- API
- Support

Operating Systems
When is a Real-Time OS essential?

- When execution time is equally important to execution correctness.
  - Hard Real-Time
    - Applies mostly to Acquisition and Control Systems
    - Execution deadlines are NEVER missed
  - Soft Real-Time
    - Applies to any system where value of computation decreases over time
    - Examples include communication, audio and video
    - Execution deadlines are MOSTLY met

8051 Example: http://www.freertos.org/portcygn.html

Software Development Tools
Software Management Tools

- Source Code Control
  - Provides source code repository and version control capabilities
- Build Management
  - Interfaces with source code control for organization level build and test management
- Test Suites
  - Commonly runs in conjunction with Build Management to test correctness of developed code
  - Unit Tests – Validates at the ‘function and file’ level
  - System Tests – Validates at the system level

Software Development Tools
Integrated Development Environment

Assists programmer in developing software

- Normally includes editor and build automation
- Sometimes includes source code control and debugger interfaces
- Based on a graphical user interface

Read more about IDE's:

http://www.codeblocks.org/home

http://www.eclipse.org/

http://www.coocox.org/software/coide.php

Software Development Tools
Simulators and Emulators

- Simulators run on a ‘host’ system and attempt to provide functionally accurate execution
  - Example: Emily52
- Emulators run in a ‘target’ system as attempt to provide functional and time accurate execution
  - Example Hardware Emulator: Nohau EMUL51-PC
  - Example In-Circuit Emulator: Ceibo DS-51

Software Development Tools

	Dunfield MICRO-C	SDCC
Preprocessor	MCP	SDCPP
Compiler	MCC51	SDCC
Linker (source or object)	SLINK	ASLINK
Optimizer	MCO51	SDCC (peephole)
Assembler	ASM51	ASX8051
Debugger/Monitor	MON51 or PAULMON2	PAULMON2 or SDCDB[1]
Simulator	Emily52	Emily52 or s51 (ucSim)[1]
Emulator	N/A	N/A
Profiler	N/A	N/A
Disassembler	PAULMON2	PAULMON2
Make	GNU Make or Dunfield Make	GNU Make (with Eclipse)

[1] SDCDB and s51 are for Linux.

Related Documentation and Tools

c-refcard.pdf	C Reference Card, 2 pages	http://refcards.com/subject/all/
c-faq	C-FAQ-list	http://www.eskimo.com/~scs/C-faq/top.html
ISO/IEC 9899:TC2	”C-Standard”	http://www.open-std.org/jtc1/sc22/wg14/www/standards.html#9899
ISO/IEC DTR 18037	”Extensions for Embedded C”	http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1021.pdf
The C Book	Free book on C	http://publications.gbdirect.co.uk/c_book/
EssentialC.pdf	Summary of the C language	http://cslibrary.stanford.edu/ and http://cslibrary.stanford.edu/101/

indent	Formats C source - Master of the white spaces	http://directory.fsf.org/GNU/indent.html
srecord	Object file conversion, checksumming, …	http://sourceforge.net/projects/srecord
cmon51	8051 monitor (hex up-/download, single step, disassemble)	http://sourceforge.net/projects/cmon51
doxygen	Source code documentation system	http://www.doxygen.org
paulmon	8051 monitor (hex up-/download, single step, disassemble)	http://www.pjrc.com/tech/8051/paulmon2.html
splint	Statically checks c sources (see 3.2.8)	http://www.splint.org

Paulmon 2

Paulmon2 Configuration

```
; These two parameters control where PAULMON2 will be assembled,
; and where it will attempt to LJMP at the interrupt vector locations.

.equ    base, 0x0000            ;location for PAULMON2
.equ    vector, 0x2000          ;location to LJMP interrupt vectors

; These three parameters tell PAULMON2 where the user's memory is
; installed.  "bmem" and "emem" define the space that will be searched
; for program headers, user installed commands, start-up programs, etc.
; "bmem" and "emem" should be use so they exclude memory areas where
; peripheral devices may be mapped, as reading memory from an io chip
; may reconfigure it unexpectedly.  If flash rom is used, "bmem" and "emem"
; should also include the space where the flash rom is mapped.

.equ    pgm, 0x2000             ;default location for the user program
.equ    bmem, 0x1000            ;where is the beginning of memory
.equ    emem, 0xFFFF            ;end of the memory
```

Look at Lab #3 details for suggested values

http://www.pjrc.com/tech/8051/paulmon2.html

Demonstration

- Create an SDCC/Eclipse project
  - Configure SDCC/Eclipse build environment
- Add C and header files
- Build project

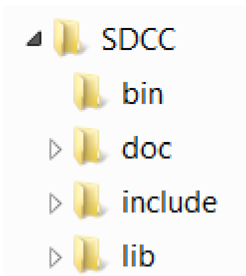
...

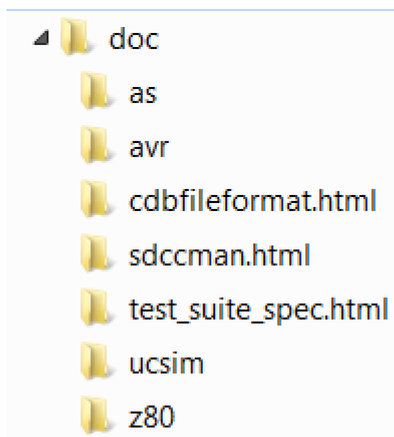
- Download and execute project
  - Demonstrate PAULMON2 features
  - Show handoff from PAULMON2 to application

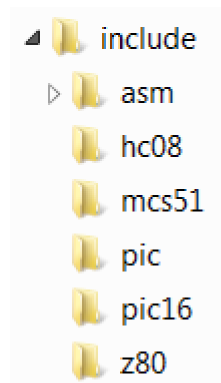
Introduction to Programming with SDCC

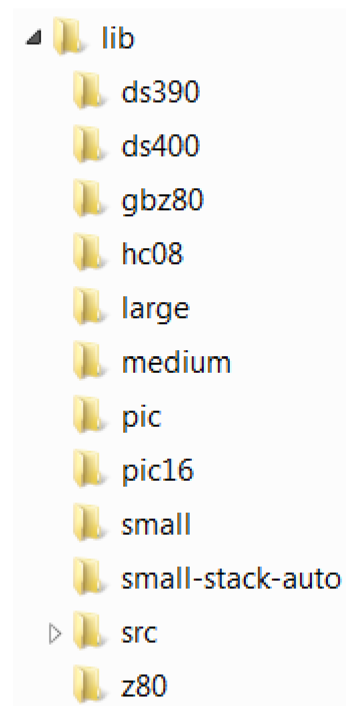
SDCC Installation

- Installation executable and user manual available on course web site









SDCC Source Code is Available

See the src folder for all the library source code

- src
  - ds390
  - ds400
  - gbz80
  - hc08
  - large
  - mcs51
  - medium
  - pic
  - pic16
  - small
  - z80
- z80

Startup code for 8051 is in the mcs51 folder

Name	Date modified	Type	Size
crtbank.asm	7/28/2006 1:59 AM	ASM File	3 KB
crtclear.asm	7/28/2006 1:59 AM	ASM File	2 KB
crtpagesfr.asm	7/28/2006 1:59 AM	ASM File	2 KB
crtstart.asm	7/28/2006 1:59 AM	ASM File	2 KB
crtxclear.asm	7/28/2006 1:59 AM	ASM File	2 KB
crtxinit.asm	7/28/2006 1:59 AM	ASM File	2 KB
crtxstack.asm	7/28/2006 1:59 AM	ASM File	2 KB

SDCC and C Documentation

- sdccman – the SDCC manual. Use the manual for SDCC 2.6.0, which is posted on the course web site and is also in the doc folder under Program Files\SDCC.
- Notes on SDCC
- SDCC Programming Tips for the Atmel AT89C51RC2
- sdcc_syntax_examples.c   and sdcc_syntax_examples.rst

- Search for “free C programming book” or “free C programming tutorial” on the web.

SDCC Tools

- sdcc - The compiler.
- sdcpp - The C preprocessor.
- asx8051 - The assembler for 8051 type processors.
- aslink -The linker for 8051 type processors.
- s51 - The ucSim 8051 simulator. Not available on Win32 platforms.
- sdcdb - The source debugger. Not available on Win32 platforms.
- packihx - A tool to pack (compress) Intel hex files. Not necessary.

SDCC Files

- sourcefile.c - C source file
- sourcefile.asm - Assembler source file created by the compiler
- sourcefile.lst - Assembler listing file created by the Assembler
- sourcefile.rst - Assembler listing file updated with linkedit information, created by linkage editor
- sourcefile.sym - symbol listing for the source file, created by the assembler
- sourcefile.rel or sourcefile.o - Object file created by the assembler, input to Linkage editor
- sourcefile.map - The memory map for the load module, created by the Linker
- sourcefile.mem - A file with a summary of the memory usage
- sourcefile.ihx - The load module in Intel hex format (you can select the Motorola S19 format with --out-fmts19.

MCS51 Storage Classes

SDCC supports standard ANSI storage classes (e.g. int, char, etc.).

In addition to the ANSI storage classes SDCC allows the following MCS51 specific storage classes:

MCS51 Storage Class Language Extensions

3.4.1.1 data / near

- This is the **default** storage class for the Small Memory model (*data* and *near* or the more ANSI-C compliant forms __*data* and __*near* can be used synonymously).

- Variables declared with this storage class will be allocated in the directly addressable portion of the internal RAM of a 8051, e.g.:

```
__data  unsigned  char  test_data;
```

- Writing 0x01 to this variable generates the assembly code:

```
75*00 01      mov    _test_data,#0x01
```

MCS51 Storage Class Language Extensions

3.4.1.2 xdata / far

- Variables declared with this storage class will be placed in the external RAM. This is the **default** storage class for the Large Memory model, e.g.:

```
__xdata  unsigned  char  test_xdata;
```

- Writing 0x01 to this variable generates the assembly code:

```
90s00r00      mov    dptr,#_test_xdata
74 01         mov    a,#0x01
F0            movx   @dptr,a
```

MCS51 Storage Class Language Extensions

3.4.1.3 idata

- Variables declared with this storage class will be allocated into the indirectly addressable portion of the internal RAM of an 8051, e.g.:

```
__idata  unsigned  char  test_idata;
```

- Writing 0x01 to this variable generates the assembly code:

```
78r00        mov    r0,#_test_idata
76 01        mov    @r0,#0x01
```

- Please note, the first 128 bytes of idata physically access the same RAM as the data memory. The original 8051 had 128 byte idata memory, nowadays most devices have 256 byte idata memory. The stack is located in idata memory.

Not used in ECEN 5613

MCS51 Storage Class Language Extensions

3.4.1.4 pdata

- Paged xdata access is just as straightforward as using the other addressing modes of a 8051. It is typically located at the start of xdata and has a maximum size of 256 bytes. The following example writes 0x01 to the pdata variable.
- Please note, pdata access physically accesses xdata memory. The high byte of the address is determined by port P2 (or in case of some 8051 variants by a separate Special Function Register, see section 4.1). This is the **default** storage class for the Medium Memory model, e.g.:

```
__pdata  unsigned  char  test_pdata;
```

- Writing 0x01 to this variable generates the assembly code:

```
78r00   mov    r0,#_test_pdata
74 01   mov    a,#0x01
F2      movx   @r0,a
```

- If the --xstack option is used the pdata memory area is followed by the xstack memory area and the sum of their sizes is limited to 256 bytes.

MCS51 Storage Class Language Extensions

3.4.1.5 code

- ’Variables’ declared with this storage class will be placed in the code memory:

```
__code unsigned char test_code;
```

- Read access to this variable generates the assembly code:

```
90s00r6F    mov    dptr,#_test_code
E4          clr    a
93          movc   a,@a+dptr
```

- char indexed arrays of characters in code memory can be accessed efficiently:

```
__code char test_array[] = {’c’,’h’,’e’,’a’,’p’};
```

- Read access to this array using an 8-bit unsigned index generates the assembly code:

```
E5*00       mov    a,_index
90s00r41    mov    dptr,#_test_array
93          movc   a,@a+dptr
```

MCS51 Storage Class Language Extensions

3.4.1.6 bit

- This is a data-type and a storage class specifier. When a variable is declared as a bit, it is allocated into the bit addressable memory of 8051, e.g.:

```
__bit     test_bit;
```

- Writing 1 to this variable generates the assembly code:

```
D2*00    setb     _test_bit
```

- The bit addressable memory consists of 128 bits which are located from 0x20 to 0x2f in data memory.
- Apart from this 8051 specific storage class most architectures support ANSI-C bitfields1. In accordance with ISO/IEC 9899 bits and bitfields without an explicit signed modifier are implemented as unsigned.

MCS51 Storage Class Language Extensions

3.4.1.7 sfr / sfr16 / sfr32 / sbit

- Like the bit keyword, *sfr / sfr16 / sfr32 / sbit* signify both a data-type and storage class, they are used to describe the *s*pecial *f* unction *r*egisters and *s*pecial *bit* variables of a 8051, eg:

```
__sfr  __at (0x80)   P0; /* special function register P0 at location 0x80 */
/* 16 bit special function register combination for timer 0 */
/* with the high byte at location 0x8C and the low byte at location 0x8A */
__sfr16  __at (0x8C8A)   TMR0;
__sbit  __at (0xd7)   CY; /* CY (Carry Flag) */
```

- Special function registers which are located on an address dividable by 8 are bit-addressable, an *sbit* addresses a specific bit within these sfr.
- 16 Bit and 32 bit special function register combinations which require a certain access order are better not declared using *sfr16* or *sfr32.* Although SDCC usually accesses them Least Significant Byte (LSB) first, this is not guaranteed.

Pointers

3.4.1.8 Pointers to MCS51/DS390 specific memory spaces

- SDCC allows (via language extensions) pointers to explicitly point to any of the memory spaces of the 8051. In addition to the explicit pointers, the compiler uses (by default) generic pointers which can be used to point to any of the memory spaces.

  Pointer declaration examples:

  ```
  /* pointer physically in internal ram pointing to object in external ram */
  __xdata  unsigned  char * __data  p;
  /* pointer physically in external ram pointing to object in internal ram */
  __data unsigned char * __xdata  p;
  /* pointer physically in code rom pointing to data in xdata space */
  __xdata unsigned char * __code  p;
  /* pointer physically in code space pointing to data in code space */
  __code unsigned char * __code  p;
  /* the following is a generic pointer physically located in xdata space */
  char * __xdata  p;
  /* the following is a function pointer physically located in data space */
  char (* __data fp)(void);
  ```

- All unqualified pointers are treated as 3-byte (4-byte for the ds390) *generic* pointers.
- The highest order byte of the *generic* pointers contains the data space information. Assembler support routines are called whenever data is stored or retrieved using *generic* pointers. These are useful for developing reusable library routines. Explicitly specifying the pointer type will generate the most efficient code.

Volatile

- In case of memory mapped I/O devices the keyword *volatile* has to be used to tell the compiler that accesses might not be removed:

```
volatile   __xdata   __at (0x8000)  unsigned char  PORTA_8255;
```

- It tells the compiler that the object is subject to sudden change for reasons which cannot be predicted from a study of the program itself, and forces every reference to such an object to be a genuine reference. For example, this prevents an optimizing compiler from removing repetitive reads to the same I/O register, like those used to poll a device. [See section 8.4.2 of "The C Book"]

Interrupts

3.8 Interrupt Service Routines

3.8.1 General Information

- SDCC allows interrupt service routines (ISRs) to be coded in C, with some extended keywords.

```
void  timer_isr (void)  __interrupt (1)  __using (1)
{
...
}
```

- The optional number following the *interrupt* keyword is the interrupt number this routine will service. When present, the compiler will insert a call to this routine in the interrupt vector table for the interrupt number specified. The optional *using* keyword can be used to tell the compiler to use the specified register bank (8051 specific) when generating code for this function.

Interrupts

- If you have multiple source files in your project, interrupt service routines can be present in any of them, but a prototype of the isr MUST be present or included in the file that contains the function *main*.
- If an interrupt service routine changes variables which are accessed by other functions these variables have to be declared *volatile*.
- If the access to these variables is not *atomic* (i.e. the processor needs more than one instruction for the access and could be interrupted while accessing the variable) the interrupt must be disabled during the access to avoid inconsistent data. Access to 16 or 32 bit variables is obviously not atomic on 8 bit CPUs and should be protected by disabling interrupts.
- Functions that are called from an interrupt service routine should be preceded by a #pragma nooverlay if they are not reentrant.

Reentrant

3.6 Parameters & Local Variables

- A reentrant function does not hold static data over successive calls, does not return a pointer to static data, and does not call non-reentrant functions. If each call to the function operates on its own copy of variables on the stack, then the function may be reentrant.
- Automatic (local) variables and parameters to functions can either be placed on the stack or in data-space. The default action of the compiler is to place these variables in the internal RAM (for small model) or external RAM (for large model). This in fact makes them similar to *static* so by default functions are non-reentrant.
- They can be placed on the stack by using the *--stack-auto* option, by using *#pragma stackauto* or by using the *reentrant* keyword in the function declaration, e.g.:

```
unsigned char foo(char i) __reentrant
{
...
}
```

- Since stack space on 8051 is limited, the *reentrant* keyword or the *--stack-auto* option should be used sparingly.

Critical

3.9.1 Critical Functions and Critical Statements

- A special keyword may be associated with a block or a function declaring it as *critical*. SDCC will generate code to disable all interrupts upon entry to a critical function and restore the interrupt enable to the previous state before returning. Nesting critical functions will need one additional byte on the stack for each call.

```
int foo ()  __critical
{
...
...
}
```

- The critical attribute may be used with other attributes like *reentrant.*
- The keyword *critical* may also be used to disable interrupts more locally:

```
__critical{ i++; }
```

- More than one statement could have been included in the block.

LECTURE BREAK

~10 Minutes

Startup Code

3.11.1 MCS51/DS390 Startup Code

- The compiler inserts a call to the C routine *_sdcc_external_startup()* at the start of the CODE area. This routine is in the runtime library. By default this routine returns 0, if this routine returns a non-zero value, the static & global variable initialization will be skipped and the function main will be invoked. Otherwise static & global variables will be initialized before the function main is invoked. You could add a *_sdcc_external_startup()* routine to your program to override the default if you need to setup hardware or perform some other critical operation prior to static & global variable initialization. On some mcs51 variants xdata memory has to be explicitly enabled before it can be accessed or if the watchdog needs to be disabled, this is the place to do it. The startup code clears all internal data memory, 256 bytes by default, but from 0 to n-1 if *--iram-sizen* is used. See also the compiler option *--no-xinit-opt* and section 4.1 about MCS51-variants.

```
// All processor XRAM should be enabled before the call to main().
_sdcc_external_startup()
{
    // initialize XRAM here
    return 0;
}
```

Special Function Register SCON (Address 98_H) Reset Value : 00_H
Special Function Register SBUF (Address 99_H) Reset Value : XX_H

Bit No.	MSB							LSB	
	$9F_H$	$9E_H$	$9D_H$	$9C_H$	$9B_H$	$9A_H$	99_H	98_H	
98_H	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	SCON
	7	6	5	4	3	2	1	0	
99_H	Serial Interface Buffer Register								SBUF

Serial I/O

```
// Remember that you need to initialize
// the serial port hardware

void putchar (char c)
{
    while (!TI);       // compare asm code generated for these three lines
    while (TI == 0);
    while ((SCON & 0x02) == 0);    // wait for TX ready, spin on TI
    SBUF = c;          // load serial port with transmit value
    TI = 0;            // clear TI flag
}

char getchar ()
{
//  char cc;
    while (!RI);       // compare asm code generated for these three lines
    while ((SCON & 0x01) == 0);  // wait for character to be received, spin on RI
    while (RI == 0);
    RI = 0;            // clear RI flag
    return SBUF;   // return character from SBUF
}
```

Redundant, choose one

If interrupts are enabled, then the order of these instructions needs to be evaluated

Advice

- For Lab #3, students should remember to program the AT89C51RC2 AUXR register for the XRAM in their C code. SDCC will have errors otherwise. Write/use the _sdcc_external_startup() function.
- Students should include at89c51ed2.h instead of at89c51rc2.h. Watch for errata.
- Remember to use the volatile keyword appropriately in your code
- Remember, you can use type casts to force the compiler to treat variables as a different type than the type they were declared
- If you use malloc() to allocate space, remember to use free() to release that space
- Use the .rst files to help you debug your code

SDCC Notes

Accessing external memory in C
syntax below (courtesy of http://www.pjrc.com/).
Notice with this technique, one simply accesses the variables lcd_command_wr, rd, etc to right or read from the LCD respectively. This syntax can also be found in section 3.6 of the SDCC manual.

Assembly Language (AS31)	**C Language (SDCC)**
.equ lcd_command_wr, 0xFE00 .equ lcd_status_rd, 0xFE01 .equ lcd_data_wr, 0xFE02 .equ lcd_data_rd, 0xFE03	volatile xdata at 0xFE00 unsigned char lcd_command_wr; volatile xdata at 0xFE01 unsigned char lcd_status_rd; volatile xdata at 0xFE02 unsigned char lcd_data_wr; volatile xdata at 0xFE03 unsigned char lcd_data_rd;

SDCC Notes

Table 19. AUXR Register

AUXR - Auxiliary Register (8Eh)

7	6	5	4	3	2	1	0
DPU	-	M0	-	XRS1	XRS0	EXTRAM	AO

Remember to enable all the XRAM

Possible Options:

Be careful about overwriting bits not associated with the specific operation being performed

```
AUXR = 0x0C;
AUXR = AUXR | BITMASK1 | BITMASK2;
AUXR |= 0x0C;
```

Remember to initialize the Serial Port bits correctly (e.g. TI)

Software Implementation Choices

Is there any significant difference between the functions below?

```
void putchar (char c)
{
    SBUF = c;          //write character to transmit buffer
    while(!TI);        //wait for transmitter to be ready
    TI = 0;            //clear the TI flag
}

void putchar (char c)
{
    while(!TI);        //wait for transmitter to be ready
    SBUF = c;          //write character to transmit buffer
    TI = 0;            //clear the TI flag
}
```

Software Implementation Choices

Compare the examples below.

```
c = getchar();
if (c == 0x13)
{
    while ((c=getchar()) != 0x11);
}
```

```
#define  XON  0x11
#define  XOFF  0x13
input = getchar();
if (input == XOFF)                      // if XOFF received, stop transmitting
{
    while ((input=getchar()) != XON);   // wait until XON received
}
```

Software Implementation Choices

Compare the examples below.

```
c = getchar();
if (c >= 0x30 && c <= 0x39)
{
    printf("%d", c-0x30);
}

input = getchar();
if (input >= '0' && input <= '9')                    // check if digit received
{
    printf("Number received = %d", input-'0');  // convert ASCII to decimal
}
```

Software Implementation Choices

Compare the examples below.

```
char * buffer;
gets(buffer);
```

```
char buffer[10];
gets(buffer);
```

```
char * buffer;
buffer = malloc (10);
gets(buffer);
```

```
char * buffer;
buffer = malloc (10);
If (buffer != NULL) {
    gets(buffer);
} else printf(“buffer not allocated”);
```

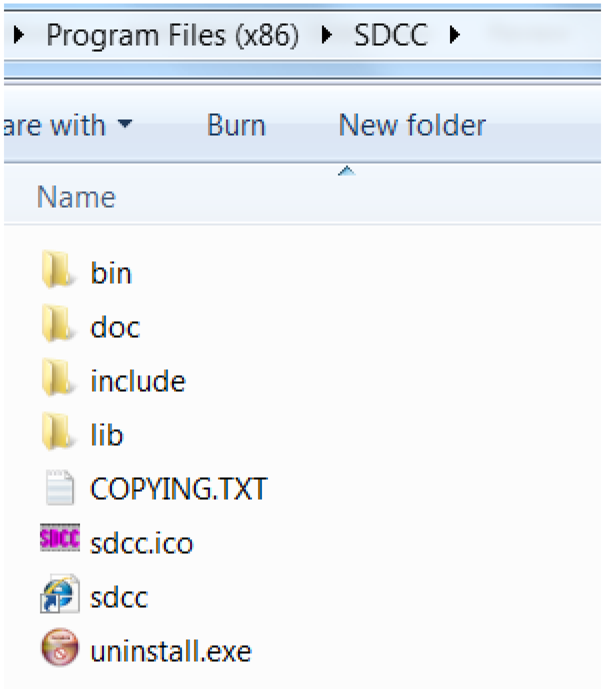
Resources

[SDCCman.pdf]

Review SDCC folder structure

  [bin/doc/include/lib]

  [lib/src]



[SDCC Syntax Examples.c]

[SDCC Syntax Examples.rst]

Make and Makefiles

GNU Make
http://www.gnu.org/software/make

Make Overview

- The make utility automatically determines which pieces of a program need to be recompiled, and issues commands to recompile them
- Make requires a makefile that describes the relationships between files in your program and provides commands for updating each file
- Once a suitable makefile exists, you only need to type ‘make’ or ‘make target’ on the command line in order to compile and link all files required for your program
- For more information, refer to the GNU make overview document (by Richard M. Stallman and Roland McGrath) available on the course web site. Parts of this presentation contain material from their GNU make overview document.
- Various make utilities are available for use in ECEN 5613. The free GNU make utility has many features, is widely used, and is standardized.

Makefile Structure

- A simple makefile consists of "rules" with the following structure:

    target ... : *prerequisites* ...

        command

        ...

- A **target** is usually the name of a file that is generated by a program; examples of targets are executable, object, or hex files. It can also be the name of an action to carry out, such as ‘clean'
- A **prerequisite** is a file that is used as input to create the target. A target often depends on several files.
- A **command** is an action that make carries out. A rule may have more than one command, each on its own line. **Note: for GNU make you need to put a tab character at the beginning of every command line!**
- Usually a command is in a rule with prerequisites and serves to create a target file if any of the prerequisites change. However, the rule that specifies commands for the target need not have prerequisites. For example, the rule containing the delete command associated with the target ‘clean' does not have prerequisites.
- A **rule**, then, explains how and when to remake certain files which are the targets of the particular rule. make carries out the commands on the prerequisites to create or update the target. A rule can also explain how and when to carry out an action.

Using Make

- GNU make processes only the specified/relevant targets in the makefile; if no target is specified, it processes the first target found in the makefile. You should place the default target first in your makefile.
- GNU make requires a tab character at the beginning of each command line.
- In the following examples, you would execute make as follows:
    - make
    - make all
    - make prog.hex

- Note: since the ‘all’ target is listed first in the makefile, it is the default target built when you type ‘make’ on the command line.

GNU Make Simple Example (SDCC)

```
##################################################################################
# GNU makefile example for code compiled with SDCC
# File Name: Makefile or makefile or gnumakefile
# Only the targets which apply will be processed by GNU make.
# To create the file 'prog.hex' using GNU make,
#     execute 'make prog.hex', 'make all', or just 'make'.
# Note: GNU make requires a special format for the makefile:
#     A tab character must be at the beginning of every command line!
##################################################################################

# Default target
all: prog.hex

# Compile phase
syntax.rel : syntax.c syntax.h
	sdcc -c -mmcs51 --std-sdcc99 --verbose --model-large syntax.c

# Link phase
prog.hex : syntax.rel
	sdcc --code-loc 0x0000 --code-size 0x8000 --xram-loc 0x0000 --xram-size
0x8000 --model-large --out-fmt-ihx syntax.rel

# Phony target
.PHONY: clean
clean:
	del *.rel *.lst *.rst *.hex *.mem *.map *.sym *.lnk *.ihx
```

GNU Make More Complex Example (SDCC)

```
###############################################################################
# GNU makefile example for code compiled with SDCC
# File Name: Makefile or makefile or gnumakefile
# Only the targets which apply will be processed by GNU make.
# To create the file 'prog.hex' using GNU make,
#            execute 'make prog.hex', 'make all', or just 'make'.
# Note: GNU make requires a special format for the makefile:
#     A tab character must be at the beginning of every command line!
###############################################################################

# Usually SDCC's large memory model is the best choice for ECEN 5613.
MEMORYMODEL = --model-large

# These settings control how the compiler will process the code
SDCCCFLAGS = -c -mmcs51 --std-sdcc99 --verbose $(MEMORYMODEL)

# These settings control where the linker will place the code
# and variables in memory.  The executable code will begin at 0000.
# External RAM usage for variables will begin at 0000.

ASLINKFLAGS = --code-loc 0x0000 --code-size 0x8000 --xram-loc 0x0000 --xram-size 0x8000 \
                   --out-fmt-ihx $(MEMORYMODEL)

# Default target
all: prog.hex

# Compile phase (the $< variable evaluates to the first prerequisite only)
syntax.rel: syntax.c syntax.h
             sdcc $(SDCCCFLAGS) $<

# Link phase (the $^ variable evaluates to all of the prerequisites for prog.hex)
prog.hex: syntax.rel
             sdcc $(ASLINKFLAGS) $^

# Phony target
.PHONY: clean
clean:
             del *.rel *.lst *.rst *.hex *.mem *.map *.sym *.lnk *.ihx
             del syntax.asm
```

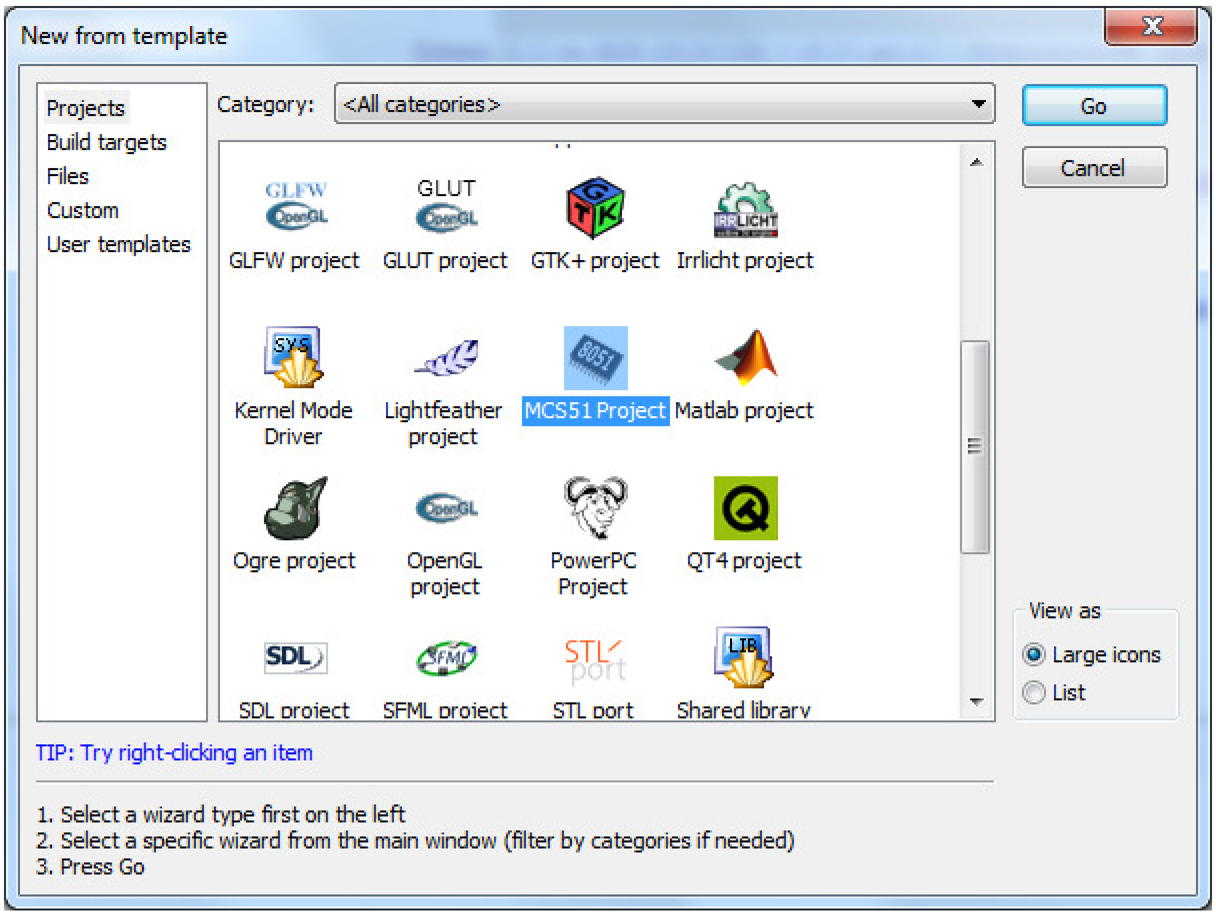
Code Blocks/SDCC Tutorial

Install Software

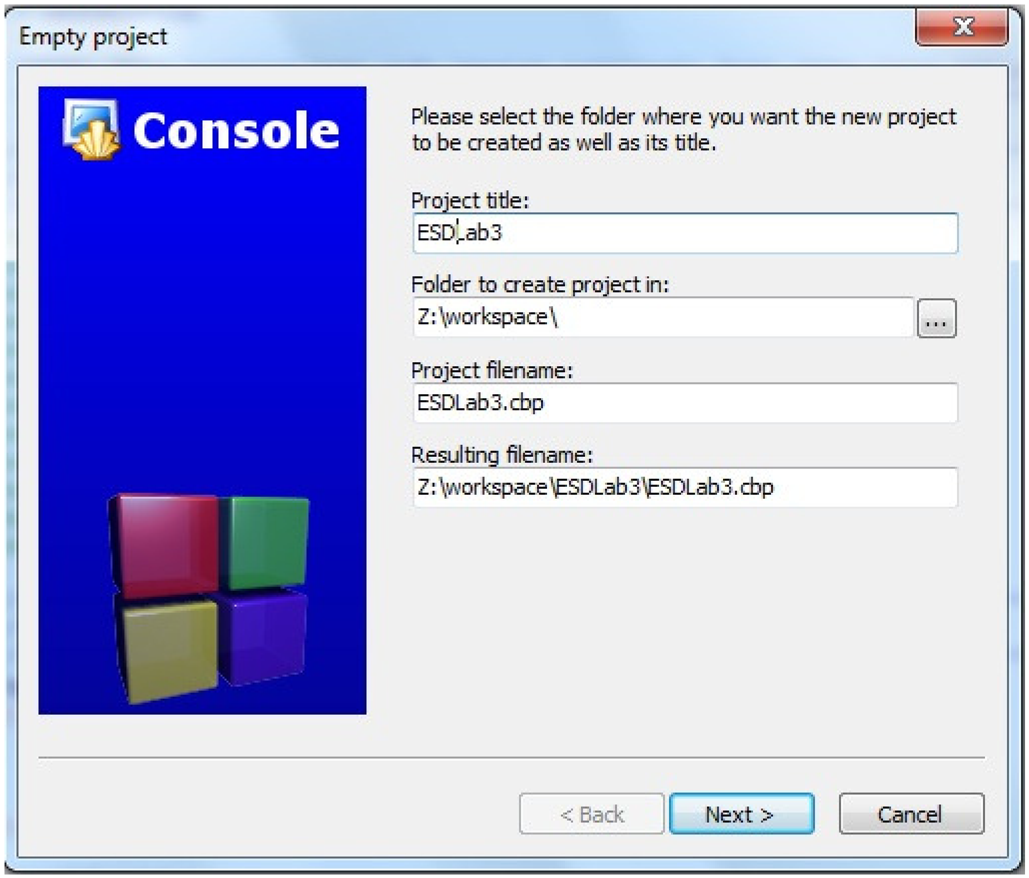
- Download and install SDCC 2.6.0
- Download and install Code Blocks
- Run Code Blocks

Create a project

- Go to ‘File->New->Project’ and select ‘MCS51 Project’.

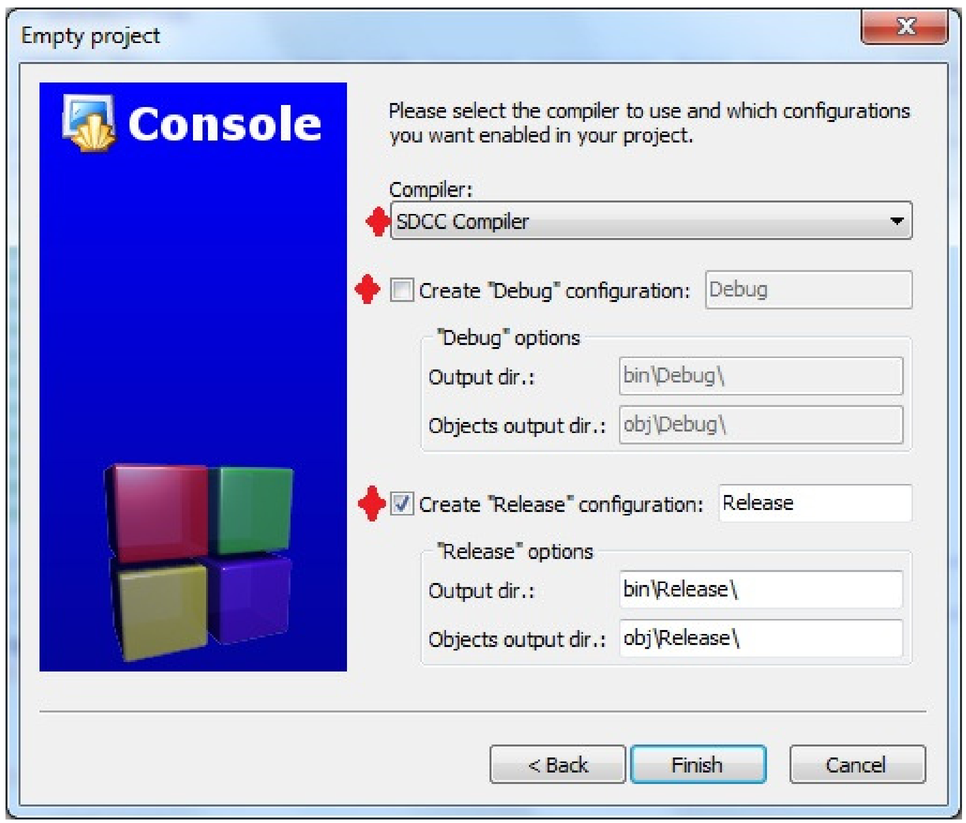


Create a project (cont.)



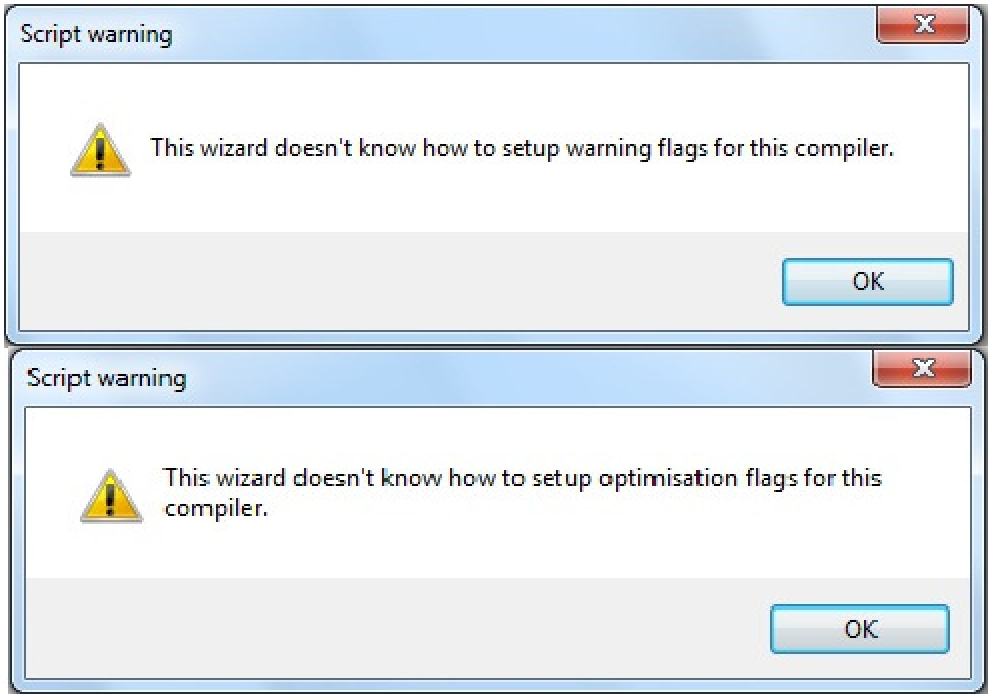
Create a project (cont.)

- Make sure SDCC is selected as your compiler, uncheck the ‘Create “Debug” configuration box and keep the “release” box checked. Click finish.



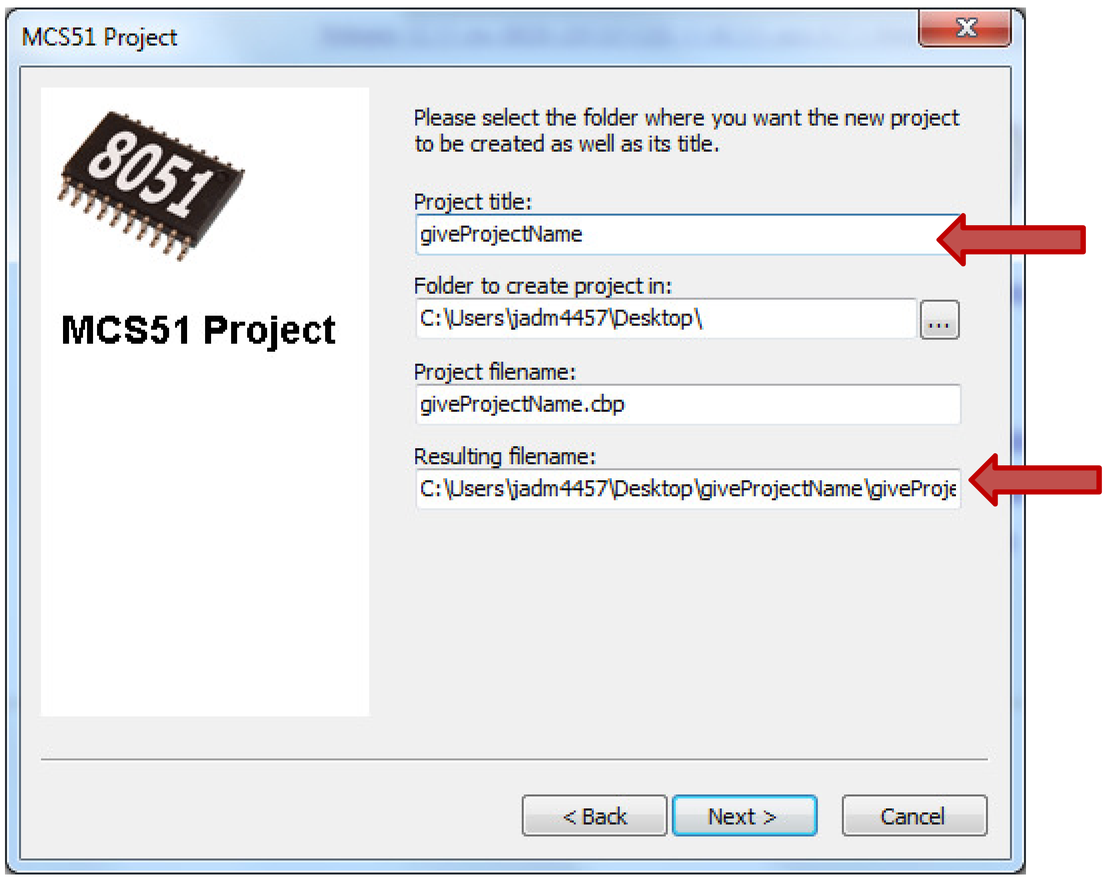
Create a project (Cont.)

- You might receive a few warnings; just ignore them!



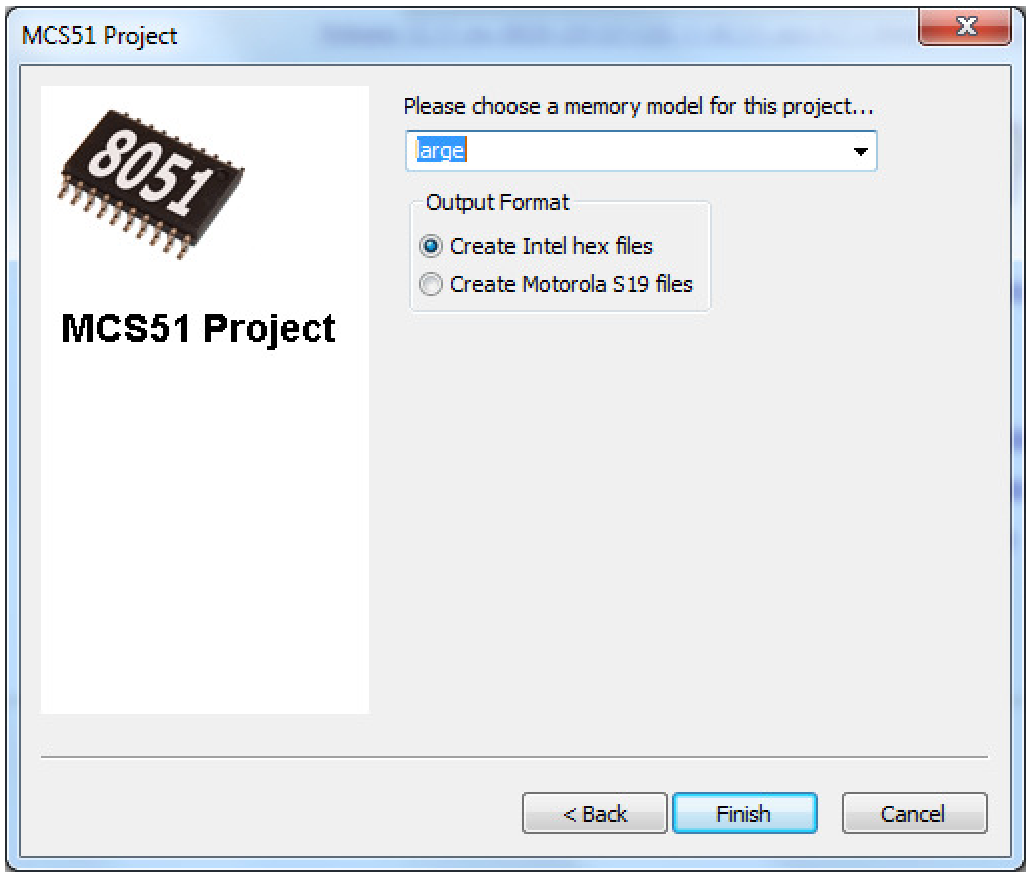
Creating a Project (Cont.)

- When Creating a new project give the project name and file path this is where the hex file will be stored.



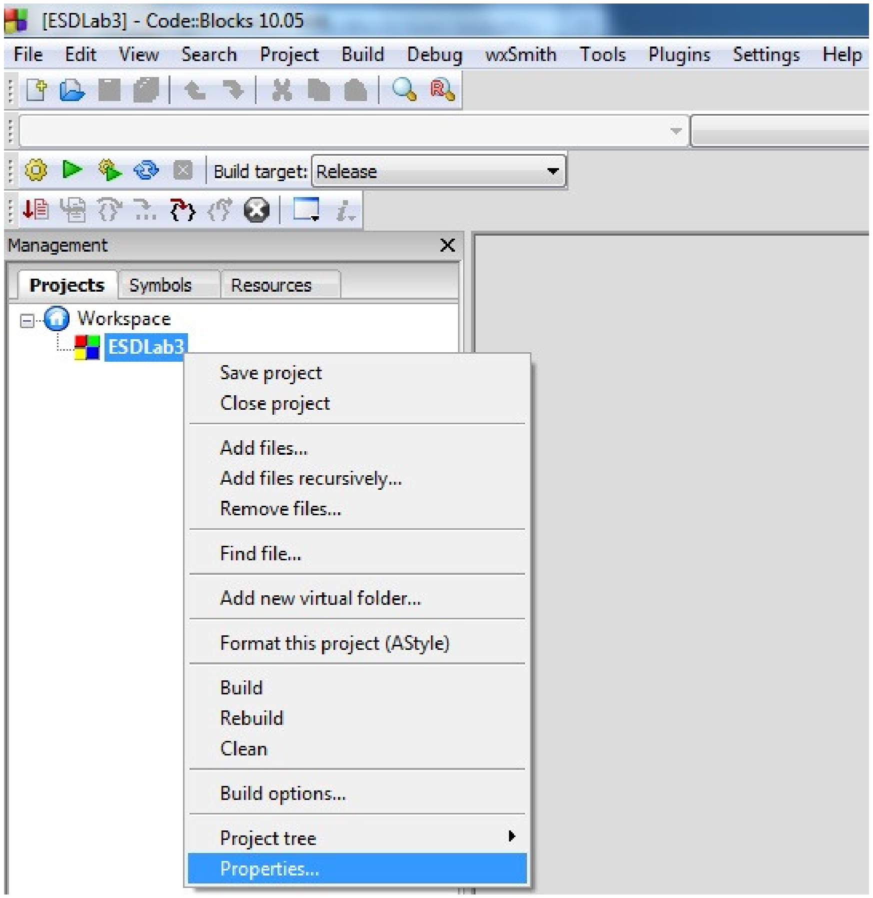
Creating a Project (Cont.)

- Choose the “large” memory model and hex file.



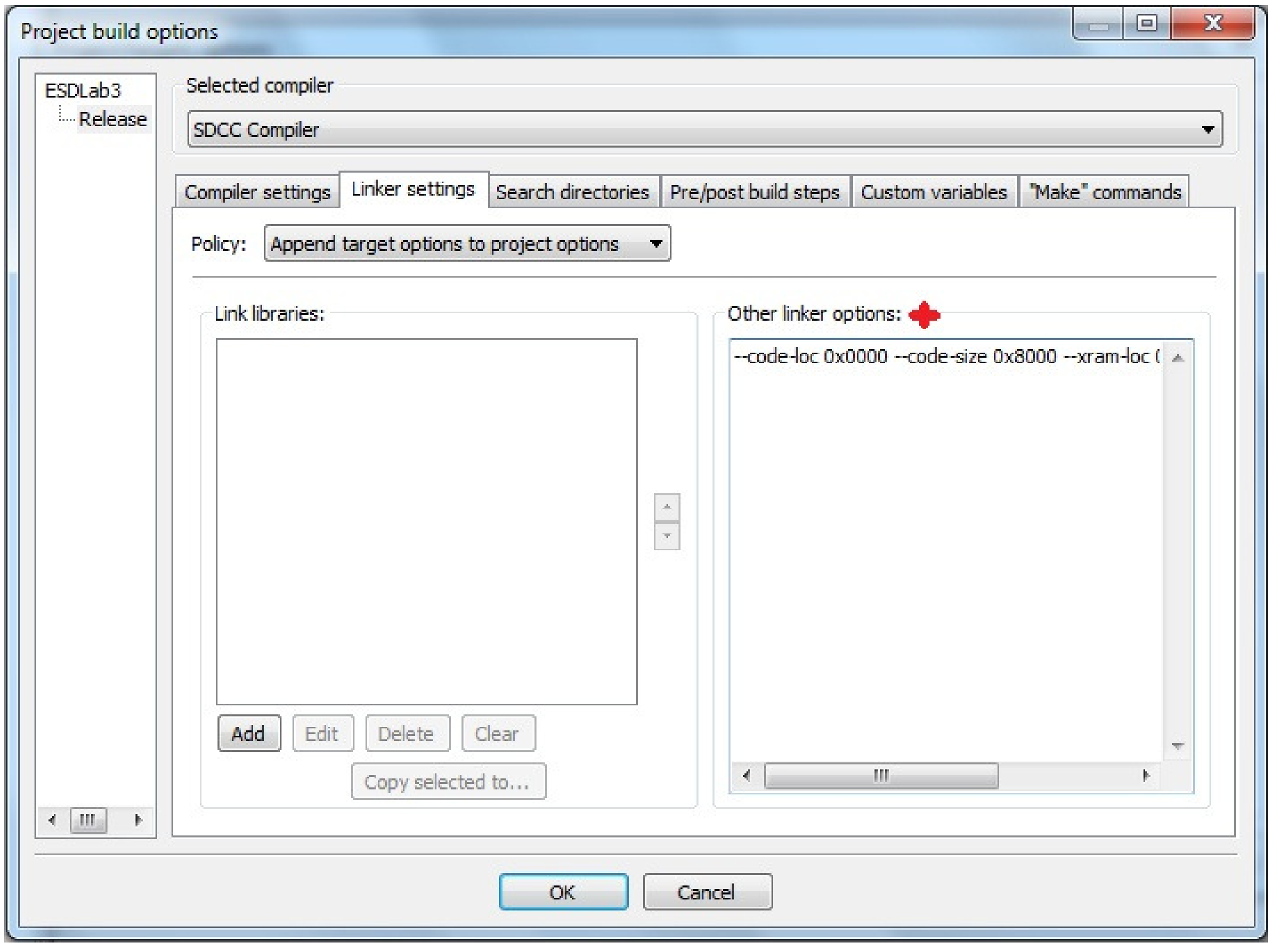
Project properties

- Right-click your project name in the 'Management' window and click 'Properties'.

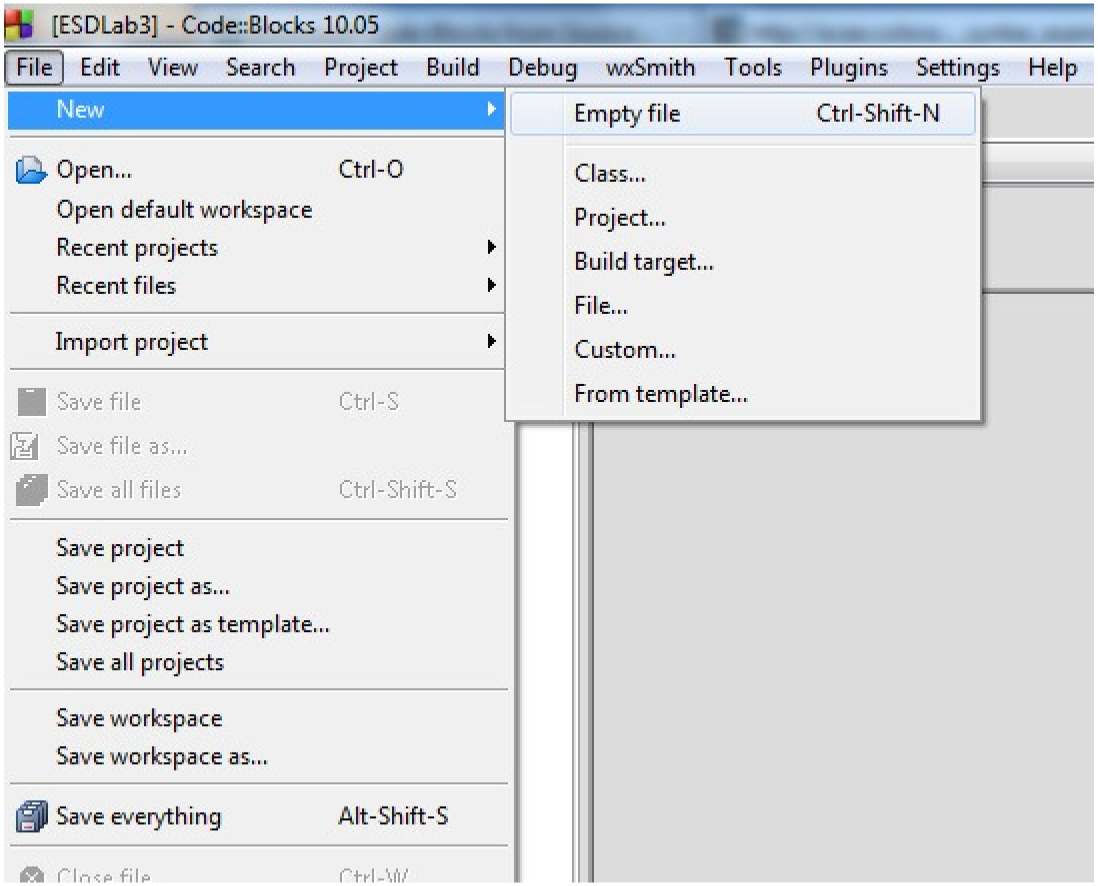


Project properties (cont.)

- You can place your linker options in the ‘Other linker options’ box in the ‘Linker settings’ tab.

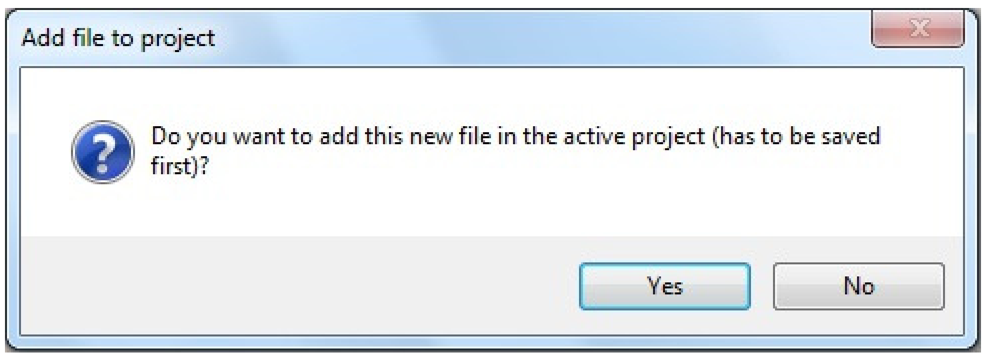


Create a new file

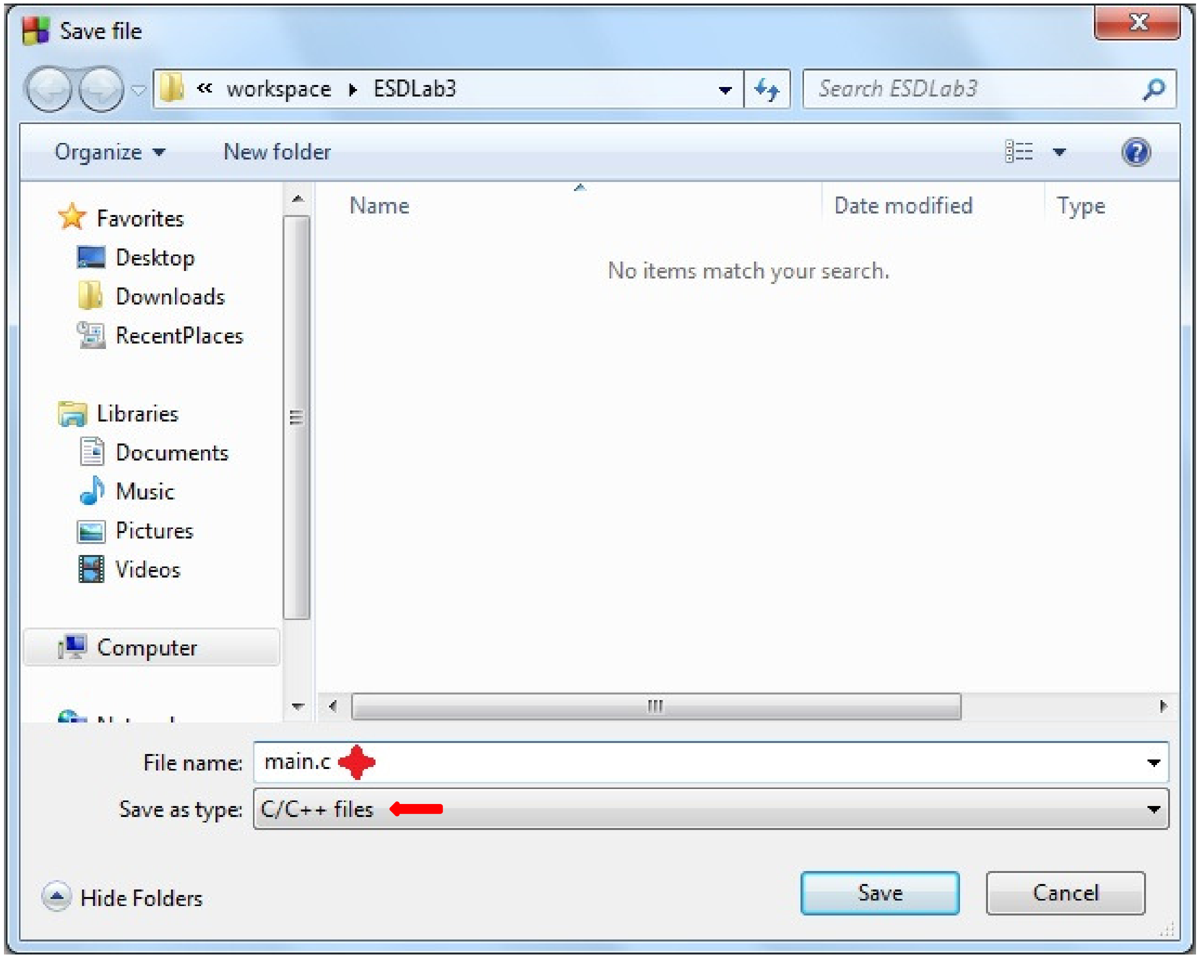


Create a new file (cont.)

- You will be asked if you want the new file to added to the project. Click yes.

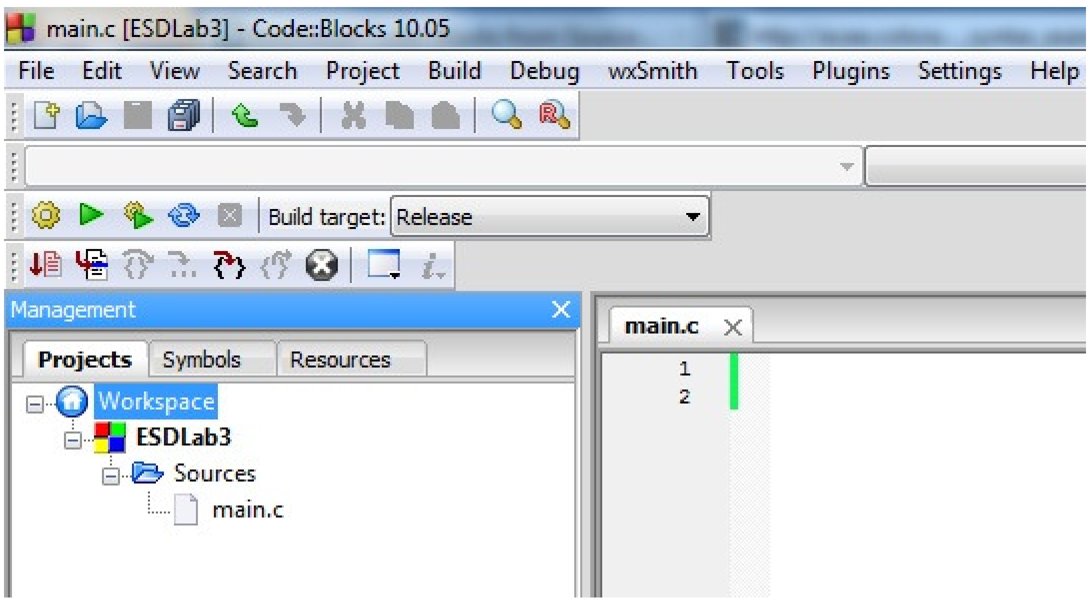


Create a new file (cont.)



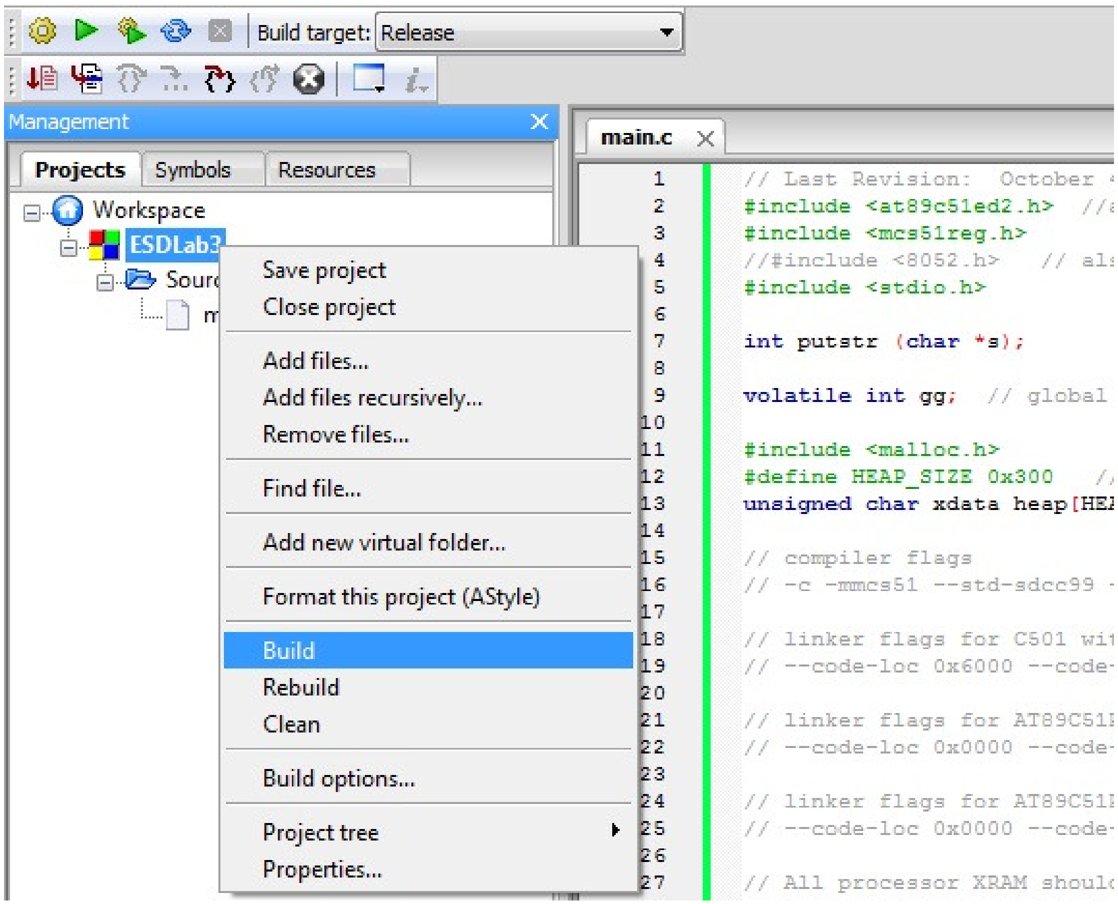
Create a new file (cont.)

- You will notice that a ‘Source’ folder is automatically created for the new file. If the you create a header (.h) file, a ‘Headers’ folder will be created instead.



Build a project

- After you build your project, you will find all the SDCC-generated files in the project folder in your workspace.



Version Control in Embedded Systems with Subversion

Embedded System Design – ECEN5613

Why Use Version Control?


Fig. 1 - Credit: SpaceX (http://www.spacex.com/photo_gallery.php)

SpaceX Demo Flight 2

- Incorrect propellant utilization file loaded into engine computer
- Resulted in:
  - Lower Thrust
  - Lower Trajectory
  - Lower Velocity second stage
  - **Failure to reach orbit**

Popular Free Version Control Systems

- CVS – Concurrent Versioning System
  - Released June 23, 1986
  - Original Purpose
    - Three programmers
    - No common schedule
    - Needed to work together
  - Grew into a widely-used version control program
- SVN – Subversion
  - Development started in 2000
  - Created to be replacement of CVS[1]
  - Fixed bugs, and added features lacking in CVS[2]
  - Used by SourceForge.net, GCC, KDE, Google Code, and more[3]

Fig. 2 - Credit: Subversion Software (http://subversion.tigris.org/)

SVN Version Control Methodology

- Server – Client Based System
  - Most up to date version kept in server-side repository
  - Repository only records changes to files
  - Repository is also compressed
    - Yields very small Subversion repositories[1]
  - Client works off of downloaded copy of repository
- How it is used
  - Projects are "Checked Out" from server repository, creating local copies[2]
  - Already local copies are "updated" before any modifications
  - Project is committed after modifications are made
  - Each commit increments project version number, file version number

Source: Brandon Gilles

How SVN Works

- SVN allows any number of checked-out repositories
  - Each can be modified concurrently
  - ‘Locking’ user out of specific files is not required
- So how are file conflicts avoided?
  - The ***copy-modify-merge*** solution
    - Allows completely parallel development
    - Prevents having to wait for a file
    - But Conflicts occur when:
      - Text files have overlapping modifications
      - Binary file modified by multiple users
        - No way for computer to know correct resolution
        - User must manually merge file

Source: Brandon Gilles

How SVN Works – Copy-Modify-Merge

SVN	Algorithm
Users 1 and 2 Checkout / Update.	Copy
Users edit source files, schematics, etc	Modify
User 1 commits to repository	Merge
User 3 updates from repository	Copy
User 2 commits to repository, only conflicts possible with user 1.	Merge
User 3 commits to repository, only conflicts possible with user 2.	Merge

Source: Brandon Gilles

Final Words

- Version Control speeds up development, eliminating the need for serialized project development
- Can save hours of development and debugging hassle
- SVN is particularly hassle free:
  - Copy-Modify-Merge nearly never has conflicts, even on large projects
  - Allows developers to always have the latest from everyone on a development team

SVN and You!

- It is easy to set up ***your very own*** subversion repository using the ECEE Student Server.
- Below are the step-by-step instructions for setting up your repository on eces-shell.

Creating the SVN on ECES-SHELL

- Connect to eces-shell.colorado.edu using your preferred SSH client

  ~~svnadmin create --fs-type fsfs esdsvn~~

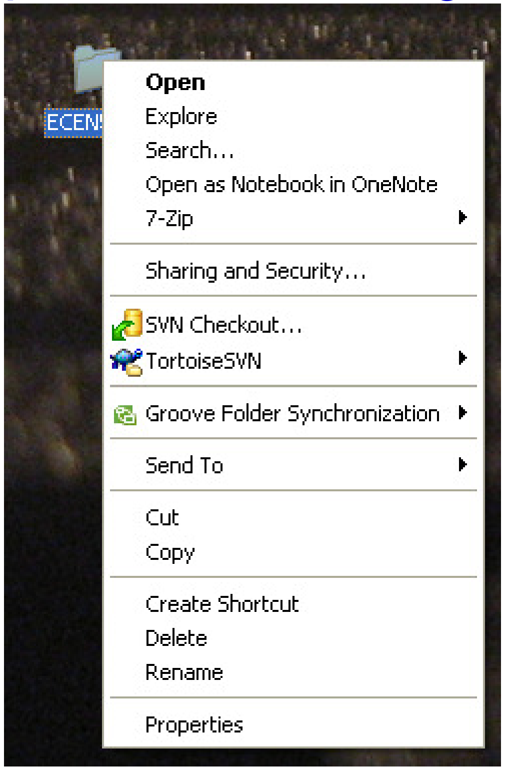
  The ECES server is now running a newer version of svnadmin (1.4.xx) and Tortoise isn't very fond of it. Use the following syntax:

  svnadmin create --pre-1.4-compatible esdsvn

- You are now ready to add files to your svn
- You can do this from a Linux command line or with Windows graphical programs.

Accessing your SVN in Windows GUI

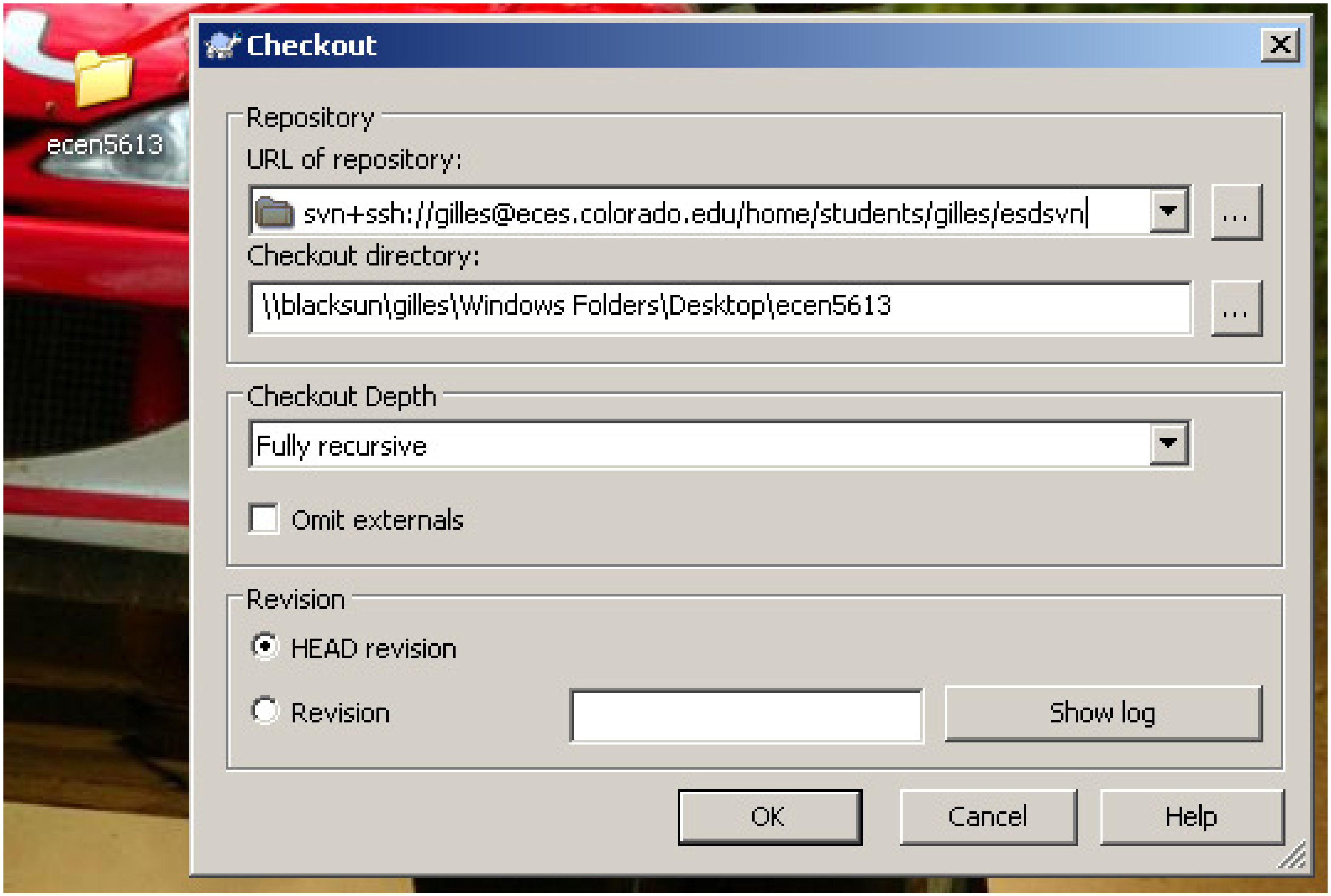
- Download and install TortoiseSVN from: http://tortoisesvn.tigris.org/



- Notice TortoiseSVN integrates into Windows

Accessing your SVN in Windows GUI

- Check out your repository



 Source: Brandon Gilles

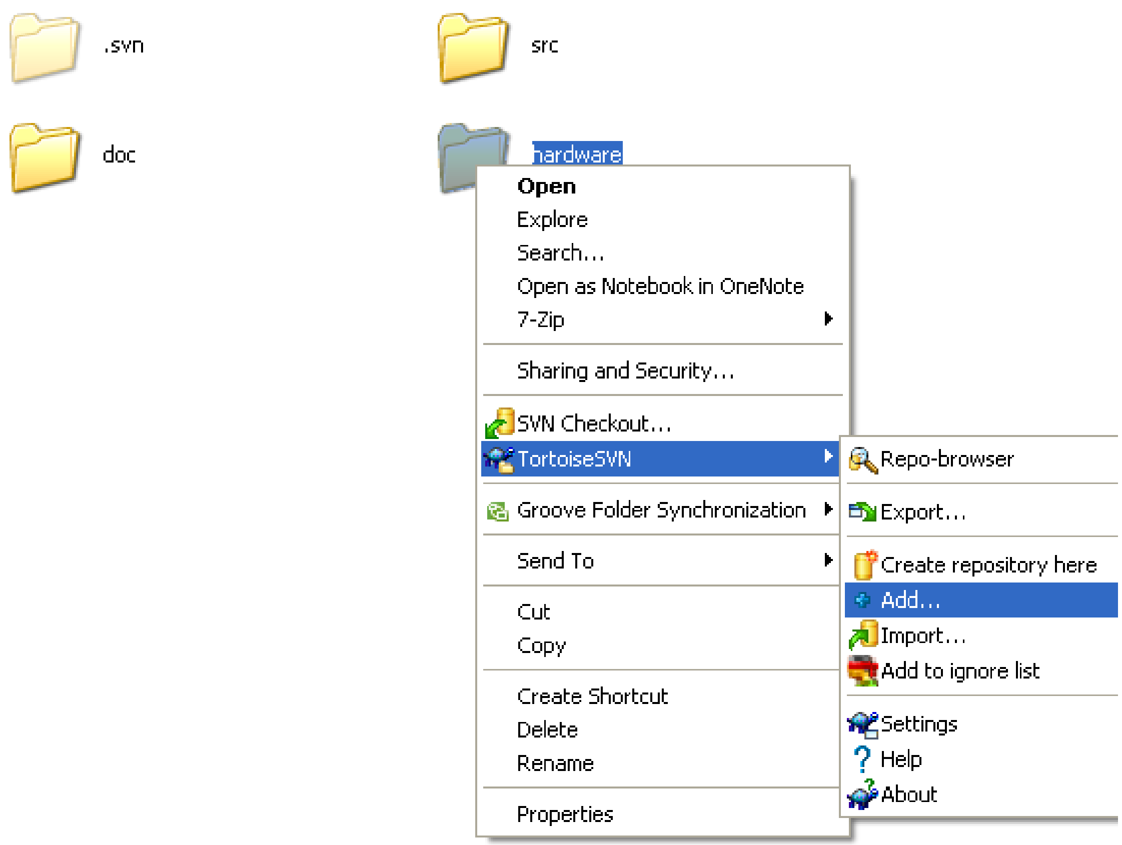
Accessing your SVN in Windows GUI

- It will ask if you trust the server.
- Say Yes (or press enter)

- No password is required in ECEE labs.
- If not in an ECEE lab, you will have to enter your password three times. This happens during checkout only.

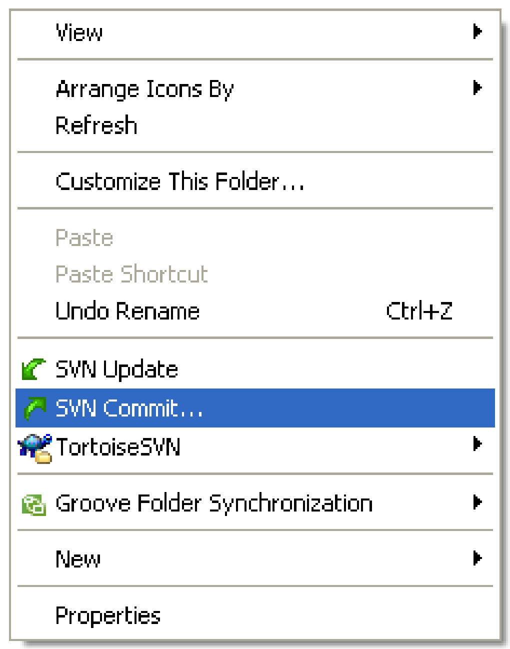
Accessing your SVN in Windows GUI

- Add files to your SVN. Be cautious. Only add files you want version controlled.



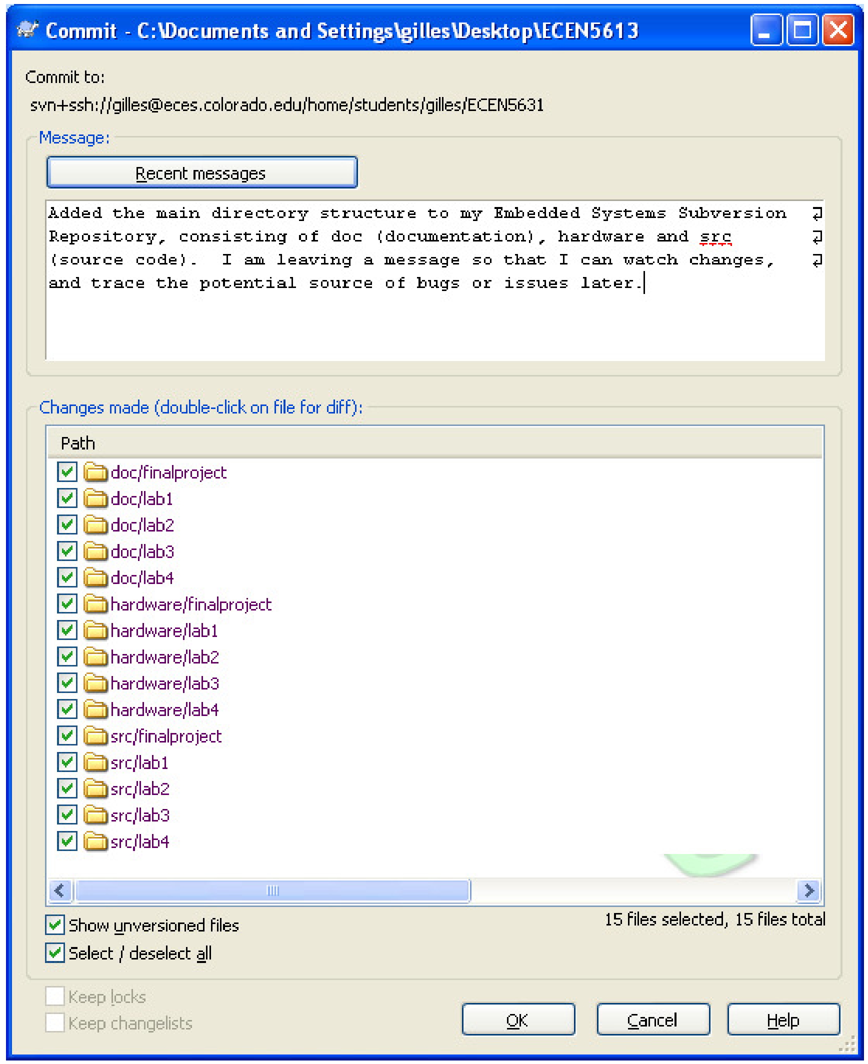
Accessing your SVN in Windows GUI

- Commit your changes to the server.



Accessing your SVN in Windows GUI

- Leave a message so you know what you changed.



Using Your SVN

- Update before you start working
- Commit periodically as you work or when you finish
- Leave messages when you are committing
- You will be happy with yourself later if you leave messages

- If using Linux to access the svn, run ‘svn help’ to learn how to do the equivalent of the above.

References

- The Subversion (SVN) material in this presentation is based on information learned from three sources:
  - 1 – the free Subversion book: http://svnbook.red-bean.com/
  - 2 – the Wikipedia article on Subversion: http://en.wikipedia.org/wiki/Subversion_%28software%29
  - 3 – Use of the TortoiseSVN SVN client: http://tortoisesvn.tigris.org/
- The Concurrent Versions System (CVS) material in this presentation is based on information learned from the Wikipedia article on CVS:
  - http://en.wikipedia.org/wiki/Concurrent_Versions_System
- The SpaceX material in this presentation is from the SpaceX "Demo Flight 2 Review Update" for the Falcon 1 Launch Vehicle, available at:
  - http://www.spacex.com/F1-DemoFlight2-Flight-Review.pdf
- Figure 1: Credit: SpaceX: http://www.spacex.com/photo_gallery.php

Programming Style - C Style Guide Presentation

Code inspections

SDCC Interrupt Exercise

1) Assume no syntax errors
2) Identify areas of possible concern

Expect about 12 unique projects in F2016.

Final Project Ideas

End of Lecture

- Operator collects attendance sheet

- Students should try to have RS-232, Atmel CPU, FLIP, Paulmon, SDCC by Friday, 10/07
  - up through first SDCC exercise (item 16)