

## ECTE333

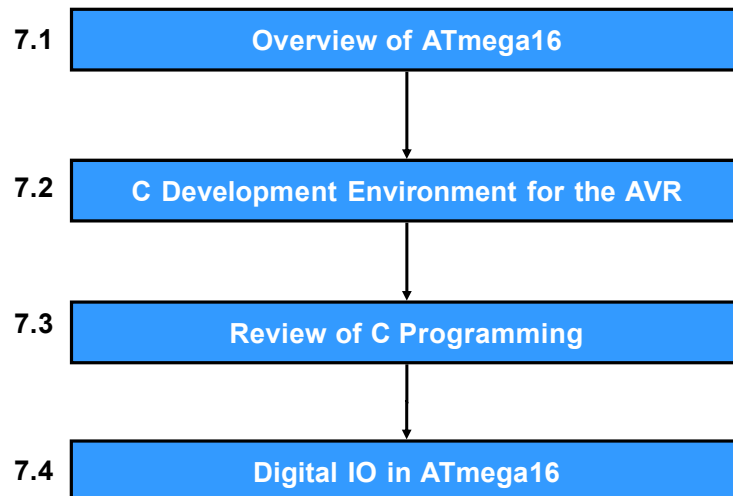
### Lecture 7 - C Programming for the Atmel AVR

School of Electrical, Computer and Telecommunications Engineering  
University of Wollongong  
Australia

## ECTE333's schedule

Week	Lecture (2h)	Tutorial (1h)	Lab (2h)
1	L7: C programming for the ATMEL AVR		
2		Tutorial 7	Lab 7
3	L8: Serial communication		
4		Tutorial 8	Lab 8
5	L9: Timers		
6		Tutorial 9	Lab 9
7	L10: Pulse width modulator		
8		Tutorial 10	Lab 10
9	L11: Analogue-to-digital converter		
10		Tutorial 11	Lab 11
11	L12: Revision lecture		
12			Lab 12
13	L13: Self-study guide (no lecture)		
Final exam (25%), Practical exam (20%), Labs (5%)			

## Lecture 7's sequence



## 7.1 Overview of ATmega16

### Why using ATmega16?

- To support different design requirements, the Atmel AVR family has many microcontrollers: ATmega8515, ATmega16, etc.
- Microcontrollers in the 8-bit AVR family share a similar instruction set and architecture.
- The ATmega16 has components that are useful for microcontroller applications, such as
  - analogue-to-digital converter,
  - pulse-width-modulator.

## Comparison between ATmega8515 and ATmega16

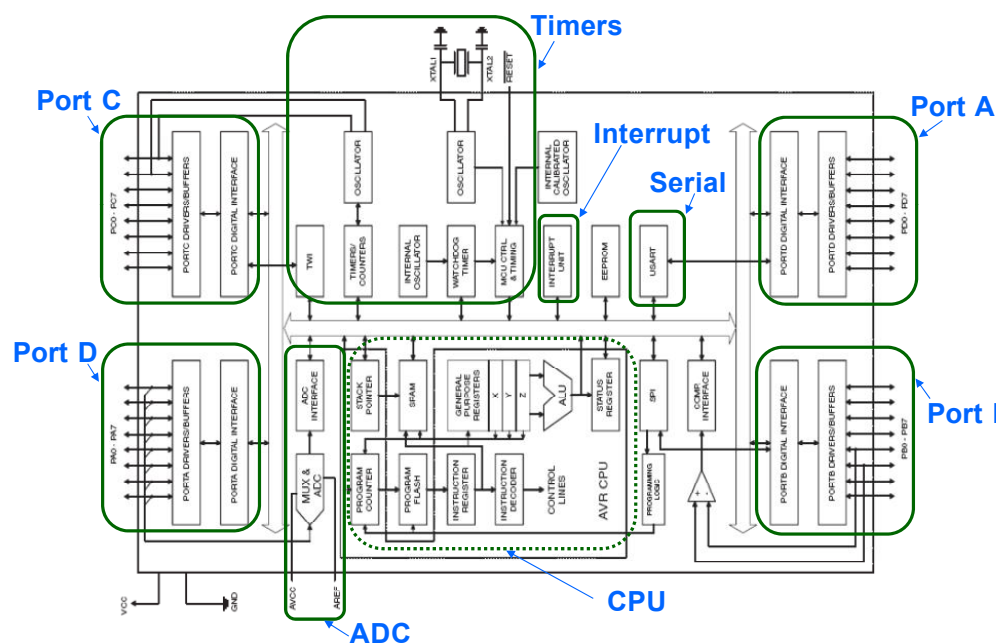
	ATmega8515	ATmega16
Number of instructions	130	131
Flash program memory	8K	16K
Internal SRAM	512 bytes	1K bytes
EEPROM	512 bytes	512 bytes
External memory	up to 64K bytes	none
General-purpose 8-bit registers	32	32
IO ports	5	4
Serial interfaces	SPI, USART	SPI, USART, Two-wire
Timers	one 8-bit, one 16-bit	two 8-bit, one 16-bit
PWM channels	3	4
ADC	Analogue comparator	Analogue comparator, 8-channel 10-bit ADC
Interrupt vectors	17	21

ECTE333

© Lam Phung, 2014.

6/65

## ATmega16 – Block diagram



ECTE333

© Lam Phung, 2014.

7/65

## ATmega16 – Pin layout

(XCK/T0) PB0	1	40	PA0 (ADC0)
(T1) PB1	2	39	PA1 (ADC1)
(INT2/AIN0) PB2	3	38	PA2 (ADC2)
(OC0/AIN1) PB3	4	37	PA3 (ADC3)
(SS) PB4	5	36	PA4 (ADC4)
(MOSI) PB5	6	35	PA5 (ADC5)
(MISO) PB6	7	34	PA6 (ADC6)
(SCK) PB7	8	33	PA7 (ADC7)
RESET	9	32	AREF
VCC	10	31	GND
GND	11	30	AVCC
XTAL2	12	29	PC7 (TOSC2)
XTAL1	13	28	PC6 (TOSC1)
(RXD) PD0	14	27	PC5 (TDI)
(TXD) PD1	15	26	PC4 (TDO)
(INT0) PD2	16	25	PC3 (TMS)
(INT1) PD3	17	24	PC2 (TCK)
(OC1B) PD4	18	23	PC1 (SDA)
(OC1A) PD5	19	22	PC0 (SCL)
(ICP1) PD6	20	21	PD7 (OC2)

**ATmega16 chip**

ECTE333

© Lam Phung, 2014.

8/65

## Using the ATmega16 for embedded applications

C programming for ...	Lecture	Lab	Tutorial
Digital IO	7	7	7
Serial communications	8	8	8
Interrupts, Timers	9	9	9
PWM	10	10	10
Analogue to digital converter	11	11	11
Embedded applications	13	12	-

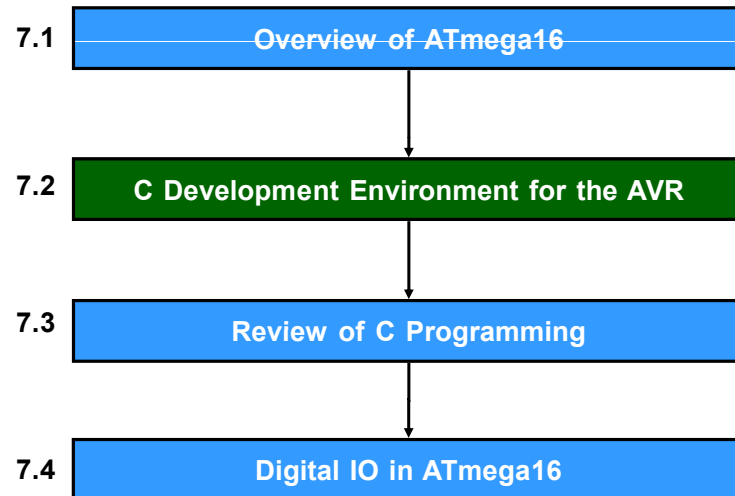
In Lecture 7, we study digital IO ports of the ATmega16.  
From Lectures 8 to 11, we'll study other peripherals in depth.

ECTE333

© Lam Phung, 2014.

9/65

## Lecture 7's sequence



## 7.2 C Development Environment for the AVR

### Why using C?

- **C is a high-level programming language:** C code is easier to understand compared to other languages.
- **C supports low-level programming:** We can use C to access all hardware components of the microcontroller.
- **C has standard libraries for complex tasks:** data type conversions, standard input/output, long-integer arithmetic.
- **The Atmel AVR instruction set is designed to support C compilers:** C code is converted efficiently to assembly code.

## Atmel Studio 6

- We will use Atmel Studio 6.
- Atmel Studio is an integrated development environment for Atmel AVR microcontrollers.
- It includes text editor, C compiler, assembler, emulator, HEX file downloader.
- It is available for free at ATMEL website.



## Installing Atmel Studio 6

For ECTE333, we use the following version:  
Atmel Studio 6 Version 6.1

- 1) Uninstall earlier versions of Atmel Studio 6.
- 2) Download setup files for Atmel Studio 6.
  - **AVR (654 MB):** [www.atmel.com/microsite/atmel\\_studio6/](http://www.atmel.com/microsite/atmel_studio6/)
  - **MVSSP1 (1.5 GB):** [www.microsoft.com/en-us/download/details.aspx?id=23691](http://www.microsoft.com/en-us/download/details.aspx?id=23691)
  - **Or from SECTE lab:** \\eisnas01\EISpub\Windows\Software\ECTE333 Spring
- 3) Run setup file for Atmel Studio 6. Accept default options.
- 4) Run setup file for Microsoft Visual Studio 10 Service Pack 1.

## Development cycle for C

### Step 1: Creating AVR Studio project

- project name
- project type C
- select device

### Step 2: Entering a C program.

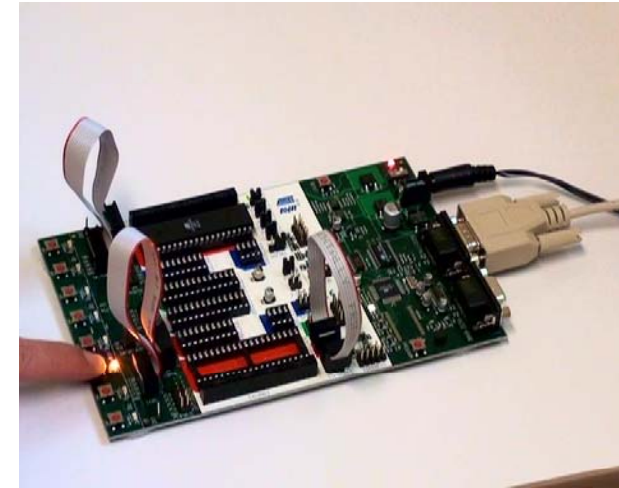
### Step 3: Compiling the C program to produce a HEX file.

### Step 4: Downloading/testing the HEX file on Atmel AVR microcontroller.

We'll illustrate these steps using an example.

Refer to the lab notes for debugging techniques.

## Development cycle for C – Example



A program that lets user turn on LED by pressing a switch.

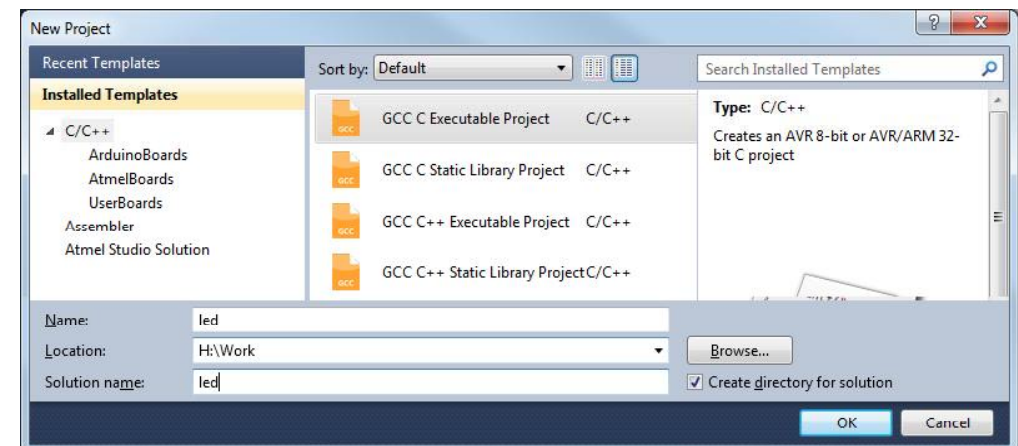
Video demo link: [avr]/ecte333/lab07\_task123.mp4



## Step 1: Creating Atmel Studio project

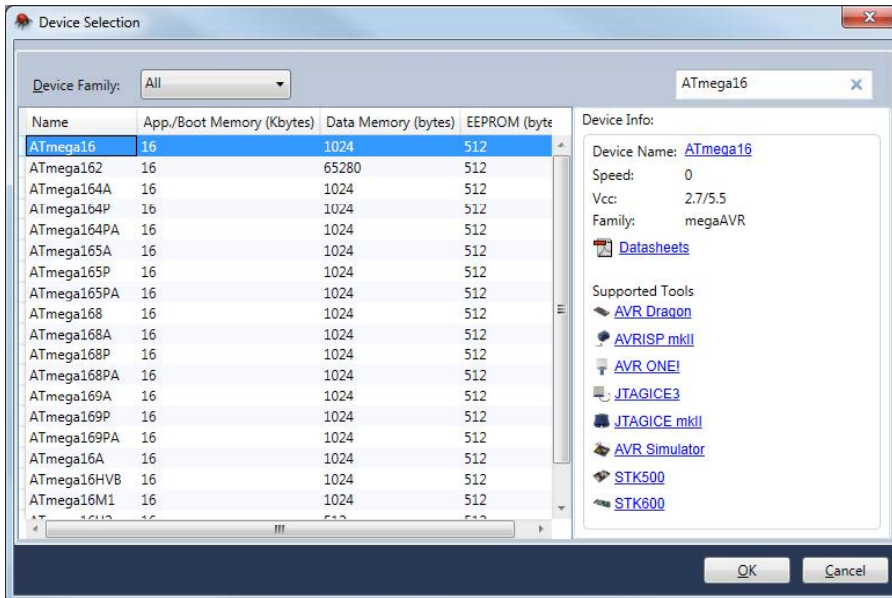
- Start the Atmel Studio 6.
- Select menu **File** | **New Project...**
  - project type: **GCC C Executable Project C/C++**
  - project name: **led**
  - project location: **H:\Work**
  - option **'Create directory for solution'**
- Click 'OK'.

## Creating Atmel Studio project



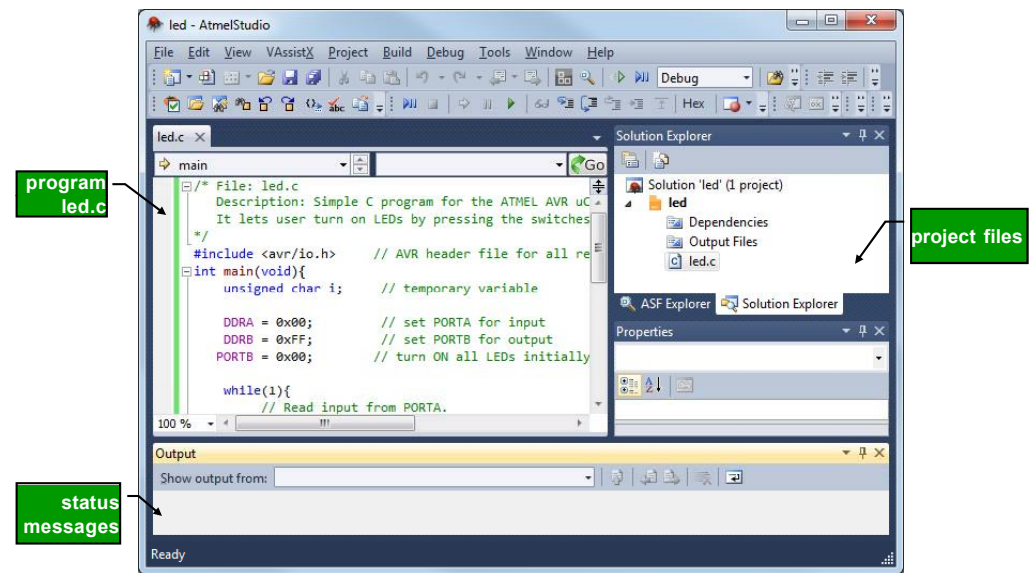
Specify project name, location, and type.

## Selecting device



Search for device **ATmega16**, and click 'OK'.

## Step 2: Entering a C program



The AVR Studio window.

## Entering a C program

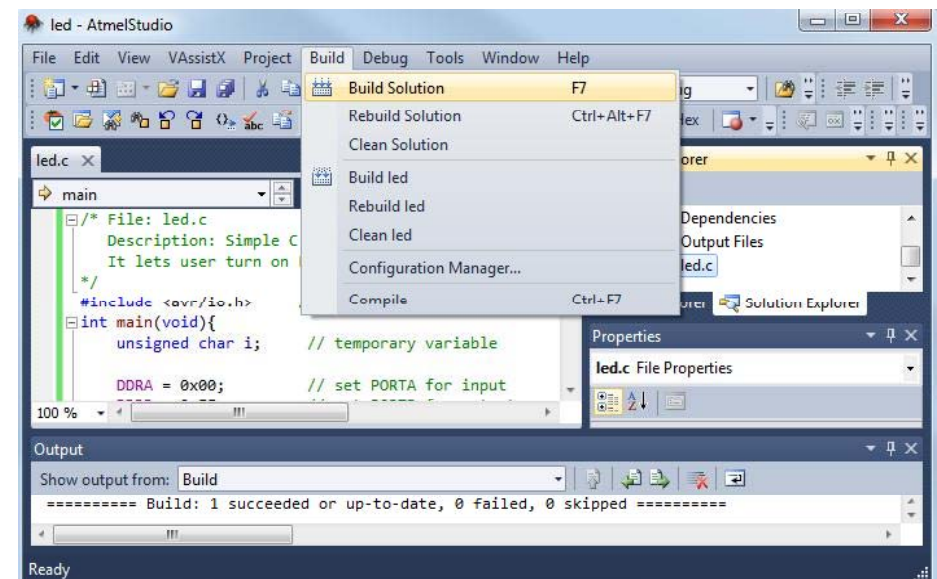
```
/* File: led.c
Description: Simple C program for the ATMEAL AVR uC (ATmega16 chip)
It lets user turn on LEDs by pressing the switches on STK500 board
*/
#include <avr/io.h>      // AVR header file for all registers/pins
int main(void){
    unsigned char i;    // temporary variable

    DDRA = 0x00;        // set PORTA for input
    DDRB = 0xFF;        // set PORTB for output
    PORTB = 0x00;        // turn ON all LEDs initially

    while(1){
        // Read input from PORTA.
        // This port will be connected to the 8 switches
        i = PINA;

        // Send output to PORTB.
        // This port will be connected to the 8 LEDs
        PORTB = i;
    }
    return 1;
}
```

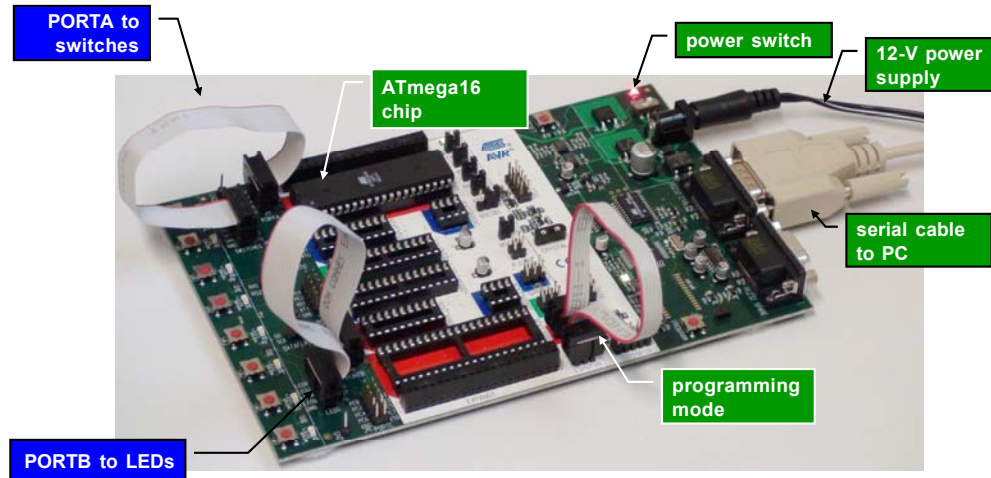
## Step 3: Compiling the C program



Select menu **Build** | **Build Solution** to generate HEX file led.hex



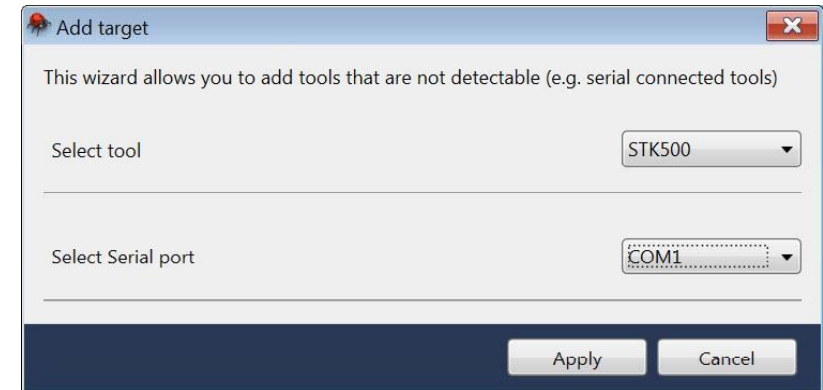
## Step 4: Downloading/testing on microcontroller



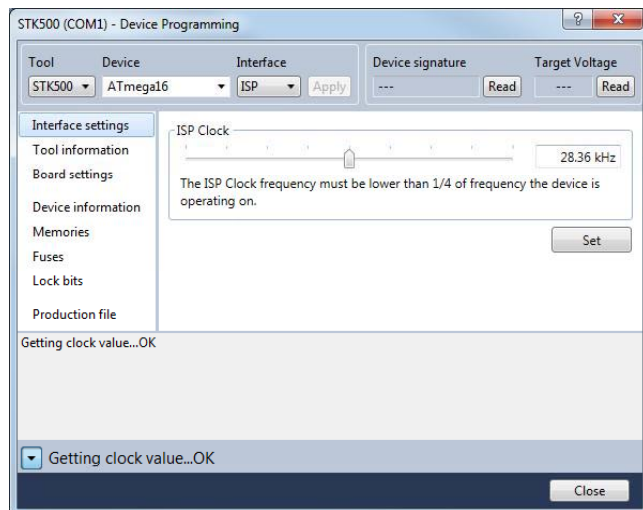
Hardware setup for example program.  
Connections to PORTA & PORTB are only for this example.

## Downloading/testing on microcontroller

- Select menu **Tools** | **Add target**.
- Select the correct serial port.
- Click 'Apply'.

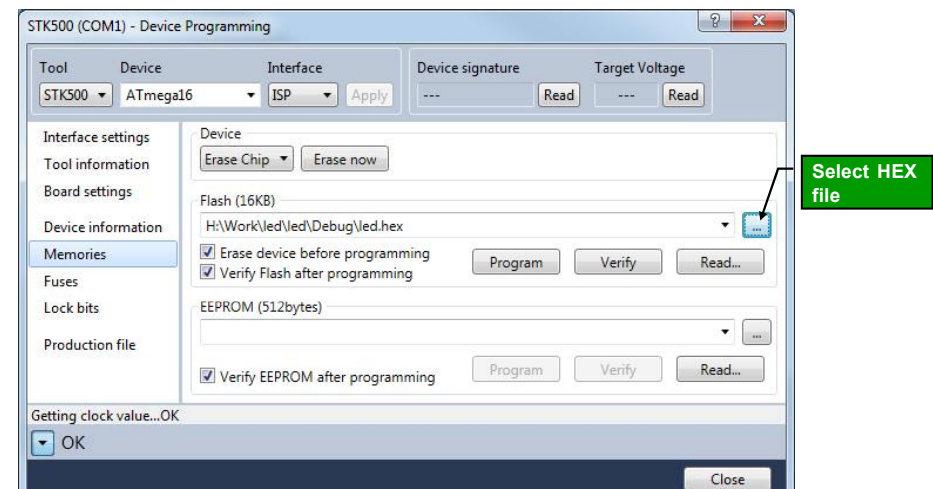


## Downloading/testing on microcontroller



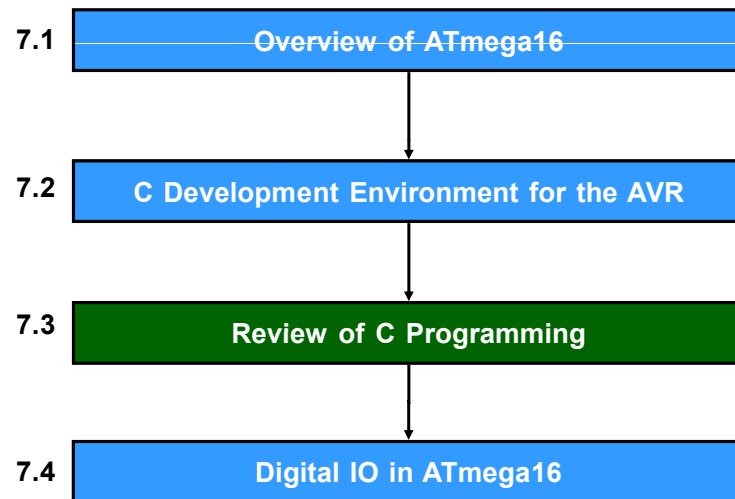
Select '**Tool**' = STK500, '**Device**' = ATmega16, and '**Interface**'= ISP

## Downloading/testing on microcontroller



- In '**Memories**' tab, select HEX file (created in Step 3) & click '**Program**'.
- After this, the program is run on the STK500 board.
- The program remains even after power-off. To erase, click '**Erase Now**'.

## Lecture 7's sequence



## 7.3 Review of C Programming

- Students are assumed to have learnt C programming in the first and second year.
- Here, we review briefly major aspects of the C programming language.
  - 7.3.1 Structure of a C program
  - 7.3.2 Data types and operators
  - 7.3.3 Flow control in C
  - 7.4.4 C functions
- In all lectures of ECTE333, C code examples will be used extensively.

### 7.3.1 Structure of a C program

- A C program typically has two main sections.
  - Section `#include`: to insert header files.
  - Section `main()`: code that runs when the program starts.
- In this example, header file `<avr/io.h>` contains all register definitions for the selected AVR microcontroller.

```
#include <avr/io.h> // avr header file for all registers/pins
int main(void){
    unsigned char i; // temporary variable
    DDRA = 0x00;      // set PORTA for input
    DDRB = 0xFF;      // set PORTB for output
    PORTB = 0x00;     // turn ON all LEDs initially
    while(1){
        // Read input from PORTA, which is connected to the 8 switches
        i = PINA;
        // Send output to PORTB, which is connected to the 8 LEDs
        PORTB = i;
    }
    return 1;
}
```

### C comments

- Comments are text that the compiler ignores.
- For a single-line comment, use double forward slashes:

```
DDRA = 0x00;    // set PORTA for input
```
- For a multi-line comment, use the pair `/*` and `*/`:

```
/* File: led.c
   Description: Simple C program for the ATMEL AVR(ATmega16 chip)
   It lets user turn on LEDs by pressing the switches on the STK500
   board
   */
```
- Always use comments to make program easy to understand.

## C statements and blocks

### ■ C statements

- C statements control the program flow.
- They consist of keywords, expressions and other statements.
- A statement ends with semicolon.

```
DDRB = 0xFF;    // set PORTB for output
```

### ■ C blocks

- A C block is a group of statements enclosed by braces {}.
- It is typically associated with **if**, **switch**, **for**, **do**, **while**, or functions.

```
while (1){  
    // Read input from PORTA - connected to the 8 switches  
    i = PINA;  
    // Send output to PORTB - connected to the 8 LEDs  
    PORTB = i;  
}
```

## Inserting assembly code into a C program

- To insert assembly code into a C program, use the **asm** directive:

```
asm("assembly instructions");
```

- For example, the following code creates a delay of one clock cycle.

```
asm volatile("nop");
```

- Keyword **volatile** is used to prevent compiler from removing the C instruction.

## 7.3.2 Data types and operators

- The main data types in C are

- **char**: 8-bit integer in AVR
- **int**: 16-bit integer in AVR
- **long int**: 32-bit integer in AVR

- The above data types can be modified by keyword '**unsigned**'.

```
char a;           // range of values for a: -128, ..., 0, ..., 127  
unsigned char b;  // range of values for b: 0, 1, 2, ..., 255  
unsigned long int c; // range of values for c: 0, 1, ..., 232 - 1
```

- Examples of variable assignment:

```
a = 0xA0;          // enter value in hexadecimal format  
a = 0b11110000;    // enter value in binary format  
b = '1';           // b stores ASCII code of character '1'  
c = 2000ul;         // c stores an unsigned long integer 2000
```

- In C, a variable or function must be declared before it is used.

## C operators

- C has a rich set of operators:

- Arithmetic operators
- Relational operators
- Logical operators
- Bit-wise operators
- Data access operators
- Miscellaneous operators



## Arithmetic operators

Operator	Name	Example	Description
*	Multiplication	$x * y$	Multiply x times y
/	Division	$x / y$	Divide x by y
%	Modulo	$x \% y$	Remainder of x divided by y
+	Addition	$x + y$	Add x and y
-	Subtraction	$x - y$	Subtract y from x
++	Increment	$x++$ $++x$	Increment x by 1 after using it Increment x by 1 before using it
--	Decrement	$x--$ $--x$	Decrement x by 1 after using it Decrement x by 1 before using it
-	Negation	$-x$	Negate x

## Relational operators

Operator	Name	Example	Description (Return a value of ...
>	Greater than	$x > 5$	1 if x is greater than 5, 0 otherwise
>=	Greater than or equal to	$x >= 5$	1 if x is greater than or equal to 5, 0 otherwise
<	Less than	$x < y$	1 if x is smaller than y, 0 otherwise
<=	Less than or equal to	$x <= y$	1 if x is smaller than or equal to y, 0 otherwise
==	Equal to	$x == y$	1 if x is equal to y, 0 otherwise
!=	Not equal to	$x != 4$	1 if x is not equal to 4, 0 otherwise

## Logical operators

- These operators are applied on logical variables or constants.

Operator	Name	Example	Description (Return a value of ...
!	Logical NOT	$!x$	1 if x is 0, otherwise 0
&&	Logical AND	$x \&\& y$	1 if both x and y are 1, otherwise 0
	Logical OR	$x    y$	0 if both x and y are 0, otherwise 1

## Bit-wise operators

- These operators are applied on individual bits of a variable or constant.

Operator	Name	Example	Description
~	Bit-wise complement	$\sim x$	Toggle every bit from 0 to 1, or 1 to 0
&	Bitwise AND	$x \& y$	Bitwise AND of x and y
	Bitwise OR	$x   y$	Bitwise OR of x and y
^	Bitwise XOR	$x \wedge y$	Bitwise XOR of x and y
<<	Shift left	$x \ll 3$	Shift bits in x three positions to the left
>>	Shift right	$x \gg 1$	Shift bits in x one position to the right

## Data-access operators

- These operators are for arrays, structures, or pointers.
- We'll learn more about them when required.

Operator	Name	Example	Description
[]	Array element	x[2]	Third element of array x
&	Address of	&x	Address of the memory location where variable x is stored
*	Indirection	*p	Content of memory location pointed by p
.	Member selection	x.age	Field 'age' of structure variable x
->	Member selection	p->age	Field 'age' of structure pointer p

## 7.3.3 Flow control in C

- By default, C statements are executed sequentially.
- To change the program flow, there are six types of statements
  - if-else statement
  - switch statement

} **Conditional**

  - while statement
  - for statement
  - do statement

} **Iterative**

  - goto statement

} **Should be avoided!**

## Miscellaneous operators

Operator	Name	Example	Description
()	Function	_delay_ms(250)	Call a library function to create a delay of 250ms
(type)	Type cast	char x = 3; (int) x	x is 8-bit integer x is converted to 16-bit integer
?	Conditional evaluation	char x; y = (x > 5) ? 10 : 20;	This is equivalent to if (x > 5) y = 10; else y = 20;



commonly used by C coders.

## 'if-else' statement

### ■ General syntax

```
if (expression)
    statement_1;
else
    statement_2;
```

### ■ Example

```
char a, b, sum;
a = 4; b = -5;
sum = a + b;
if (sum < 0)
    printf("sum is negative");
else if (sum > 0)
    printf("sum is positive");
else
    printf("sum is zero");
```

## 'switch' statement

### ■ General syntax

```
switch (expression)
case constant_1:
    statement_1;
    break;
case constant_2:
    statement_2;
    break;
...
case constant_n:
    statement_n;
    break;
default:
    statement_other;
}
```

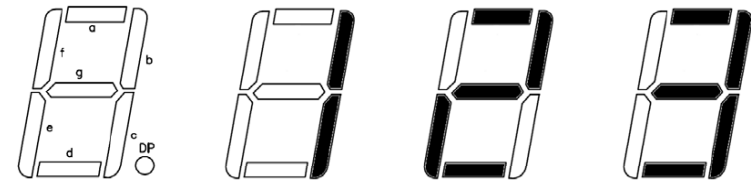
Use 'break' to separate different cases.



## 'switch' statement — Example

**Lab 7:** Find the bit pattern to display a digit on the 7-segment LED.

(a) 7 segments of the LED



(b) Bit assignment on the 8-pin connector

LED segment	DP	g	f	e	d	c	b	a
Bit number	7	6	5	4	3	2	1	0
To display '1'	0	0	0	0	0	1	1	0
To display '2'	0	1	0	1	1	0	1	1

## 'switch' statement — Example

```
unsigned char digit;           // input
unsigned char led_pattern;     // output
switch (digit){
case '0':
    led_pattern = 0b00111111;
    break;
case '1':
    led_pattern = 0b00000110;
    break;
case '2':
    led_pattern = 0b01011011;
    break;
    // You can complete more cases here...
default:
}
PORTB = led_pattern; // send to PORTB and 7-segment LED
```

## 'while' statement

### ■ General syntax

```
while (expression){
    statements;
}
```

■ **Example:** Compute the sum of  $1 + 2 + \dots + 10$ .

```
int sum, i;
i = 1; sum = 0;
while (i <= 10){
    sum = sum + i;
    i = i + 1;
}
```

## 'for' statement

### ■ General syntax

```
for (expression1; expression2; expression3){
    statements;
}
```

- **expression1** is run before the loop starts.
- **expression2** is evaluated before each iteration.
- **expression3** is run after each iteration.

### ■ Example: Compute the sum of $1 + 2 + \dots + 10$ .

```
int sum = 0;

for (int i = 1; i <= 10; i++){
    sum = sum + i;
}
```

## 'do' statement

### ■ General syntax

```
do {
    statements;
} while (expression);
```

### ■ Example: Compute the sum of $1 + 2 + \dots + 10$ .

```
int sum, i;
i = 1; sum = 0; // Initialization
do{
    sum = sum + i;
    i = i + 1;
} while (i <= 10);
```

## 'break' statement in loop

- The **'break'** statement inside a loop forces early termination of the loop.

### ■ What is the value of 'sum' after the following code is executed?

```
int sum, i;
i = 1; sum = 0; // Initialization
while (i <= 10){
    sum = sum + i;
    i = i + 1;
    if (i > 5)
        break;
}
```

sum = ?

## 'continue' statement in loop

- The **'continue'** statement skips the subsequent statements in the code block, and forces the execution of the next iteration.

### ■ What is the value of 'sum' after the following code is executed?

```
int sum, i;
i = 1; sum = 0; // Initialization
while (i <= 10){
    i = i + 1;
    if (i < 5)
        continue;
    sum = sum + i;
}
```

sum = ?

## C arrays

- An array is a list of values that have the same data type.
- In C, array index starts from 0.
- An array can be one-dimensional, two-dimensional, or more.
- This code example creates a 2-D array (multiplication table):

```
int a[8][10];
for (int i = 0; i < 8; i++)
    for (int j = 0; j < 10; j++)
        a[i][j] = i * j;
```

- An array can be initialized when it is declared.

```
int b[3] = {4, 1, 10};
unsigned char keypad_key[3][4] = {{ '1', '4', '7', '*' },
                                   { '2', '5', '8', '0' },
                                   { '3', '6', '9', '#' } };
```

## 7.3.4 C functions

- C functions are sub-routines that can be called from the main program or other functions.
- Functions enable modular designs, code reuse, and hiding of complex implementation details.
- A C function can have a list of parameters and produce a return value.
- Let's study C functions through two examples.

## Functions – Example 1

Write a function to compute the factorial  $n!$  for a given  $n$ .

```
// factorial is the name of the custom function
// it accepts an input n of type int, and return an output of type int
int factorial(int n){
    int product = 1;
    for (int i = 1; i <=n; i++)
        product = product * i;
    return product;    // return the result
}
```

```
int main(void){
    int n = 5;           // some example value of n
    int v;               // variable to storage result
    v = factorial(n);    // call the function, store return value in v
    return 1;
}
```

## Functions – Example 2

Write a function to compute the factorial  $n!$  for a given  $n$ .

```
// factorial is the name of the custom function
// it accepts an input n of type int,
// it stores output at memory location by int pointer p
void factorial(int n, int* p){
    int product = 1;
    for (int i = 1; i <=n; i++)
        product = product * i;
    *p = product; // store output at memory location pointed by p
}
```

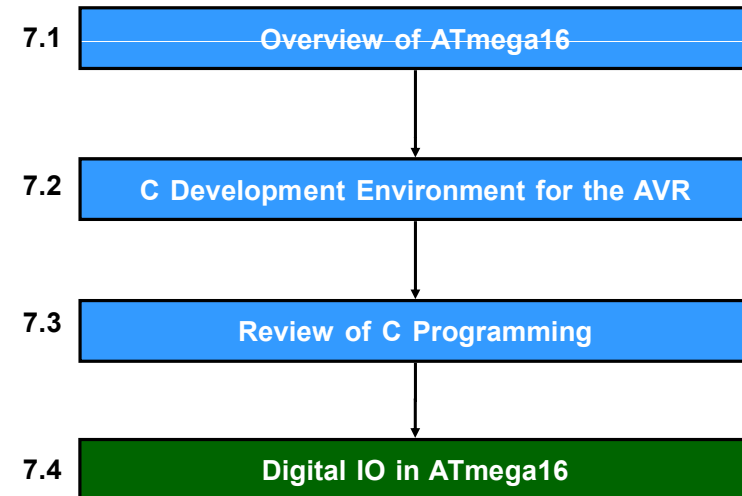
```
int main(void){
    int n = 5;           // some example value of n
    int v;               // variable to storage result
    factorial(n, &v);    // call the function, store return value in v
}
```



## Guidelines on C coding and documentation

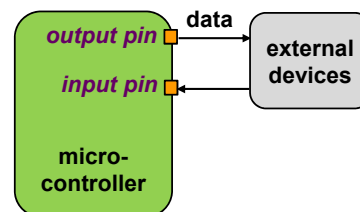
- Optimise the C code for efficiency and length.
- Delete unnecessary lines of code.
- The C code must be properly formatted.
- Use correct indentation to show the logical structure of the program.
- Use a blank line to separate code sections.
- Use meaningful names for variables and functions.
- Use concise C comments to explain code.
- For printing, use a fixed-width font such as Courier New for code.
- If a C statement is too long, split it logically into multiple lines.
- Observe the way that C code is presented in the lecture/lab notes.

## Lecture 7's sequence

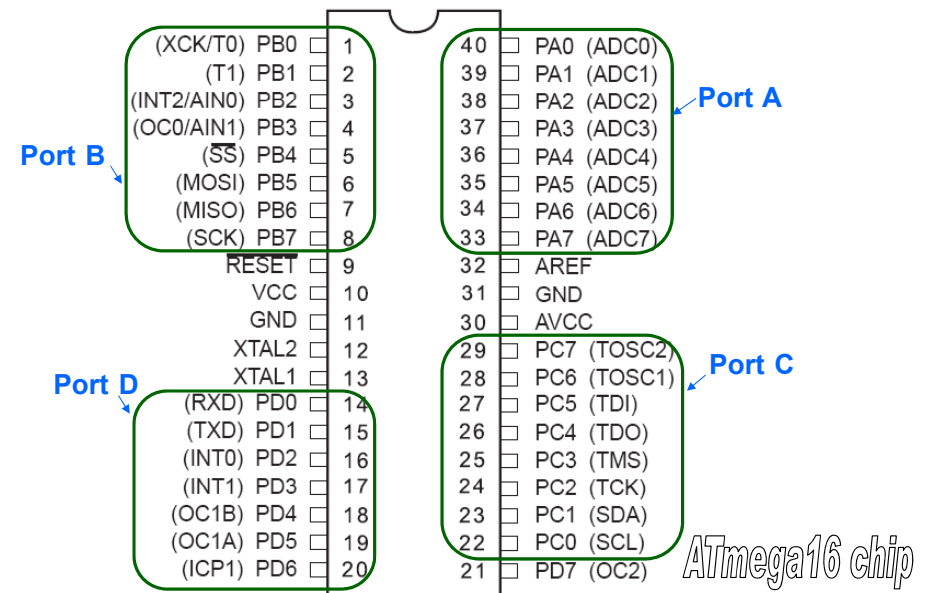


## 7.4 Digital IO in ATmega16

- ATmega16 has four 8-bit digital IO ports:
  - PORT A,
  - PORT B,
  - PORT C, and
  - PORT D.
- Each port has 8 data pins.
- Every port is bi-directional.
- Each of the 8 pins can be individually configured for input or output.



## Digital IO in ATmega16 – Pins



## Digital IO in ATmega16 – Configuring for input/output

- For each port, there are three relevant 8-bit registers.

- Data Direction Register (DDRx)
- Input Pins Address (PINx)
- Data Register (PORTx)

Here, x denotes A, B, C, or D.

- **Data Direction Register (DDRx)** is used to configure a specific port pin as output (1) or input (0).

- **Example:** To set Port A pins 0 to 3 for input, pins 4 to 7 for output, we write C code

```
DDRA = 0b11110000; // configure pins
```

bit	7	6	5	4	3	2	1	0
DDRA	1	1	1	1	0	0	0	0
	for output				for input			

## AVR header file

- To access all AVR microcontroller registers, your program must include the header file <io.h>, which is found in the WinAVR folder.

```
#include <avr/io.h>
```

- Depending on the device selected in your project, file 'io.h' will automatically redirect to a specific header file.

- **Example**

- For ATmega16, the specific header file is 'avr/iom16.h'.
- This header file is in Appendix A of the lab notes.
- It lists the C names and addresses for all ATmega16 registers.
- We always use the C names in our code.

## Digital IO in ATmega16 – Reading from/Writing to Port

- Register **Data Register (PORTx)** is used to write output data to port.

- **Example:** To write a binary 0 to output pin 6, binary 1 to other pins of Port A, we write C code

```
PORTA = 0b10111111; // write output
```

- Register **Input Pins Address (PINx)** is used to read input data from port.

- **Example:** To read the input pins of Port A, we write C code

```
unsigned char temp; // temporary variable
```

```
temp = PINA; // read input
```

- Where do the C names PINA, PORTA, and DDRA come from?

```
// Extract from header file <avr/iom16> ...
```

```
#define PINA _SFR_IO8(0x19)
```

```
#define DDRA _SFR_IO8(0x1A)
```

```
#define PORTA _SFR_IO8(0x1B)...
```

## Digital IO in ATmega16 – Example

```
/* File: led.c
   Description: Simple C program for the ATMEL AVR uC (ATmega16 chip)
   It lets user turn on LEDs by pressing the switches on STK500 board
*/
#include <avr/io.h> // AVR header file for all registers/pins
int main(void){
    unsigned char i; // temporary variable

    DDRA = 0x00; // set PORTA for input
    DDRB = 0xFF; // set PORTB for output
    PORTB = 0x00; // turn ON all LEDs initially

    while(1){
        // Read input from PORTA.
        // This port will be connected to the 8 switches
        i = PINA;

        // Send output to PORTB.
        // This port will be connected to the 8 LEDs
        PORTB = i;
    }
    return 1;
}
```



Demo in  
slide 15.


## Lecture 7's summary

---

### ■ What we learnt in this lecture:

- An overview of ATmega16.
- The tools and the steps for programming the Atmel AVR.
- Basics about C programming.
- Programming the digital IO ports of ATmega16.

### ■ What are the next activities?

- Tutorial 7: 'Introduction to C programming for the AVR' .
- Lab 7: 'Introduction to C programming for the AVR'
  - ❖ Complete online Pre-lab Quiz for Lab 7.
  - ❖ Write programs for tasks 4 and 5 of Lab 7.
  - ❖ Use video demo of Lab 7: [avr]/ecte333/lab07\_task123.mp4  
[avr]/ecte333/lab07\_task4.mp4 

## Lecture 7's references

---

- J. Pardue, C Programming for Microcontrollers, 2005, SmileyMicros, [Chapters 1-6].
- S. F. Barrett and D. J. Pack, Atmel AVR Microcontroller Primer: Programming and Interfacing, 2008, Morgan & Claypool Publishers, [Chapter 1].
- Atmel Corp., 8-bit AVR microcontroller with 16K Bytes In-System Programmable Flash ATmega16/ATmega16L, 2007. *Manual*