ESE 381 Embedded Microprocessor System Design II

Prof. Ken Short

Agenda

- Course organization
- Design environment
- Review of AVR Port Logic
- In and Out Program in C

Lecture, Office Hours, and Lab Times

- Lecture: Tue. and Thu. 10:00 - 11:20 am; Light Engineering 154
- Office Hours: Tue. and Thu. 12:30 – 1:30 and 4:00 - 5:00 pm Light Engineering 229
- Lab L01 Weds. 9:00 am to 12:00 pm; Light Engineering 230
- Lab L02 Weds. 2:30 pm to 5:30 pm; Light Engineering 230

Class Participation

- Attendance
- Punctuality
- Participation in class discussions
- Conduct in class

Important Note

- Turn **OFF** cell phones, computers, and other electronic devices and put them away before the start of class.

Primary Course Objectives

The primary course objectives are for you to learn:

- Additional general embedded system design concepts
- Detailed design techniques for implementing embedded systems using AVR family microcontrollers
- Embedded software development using embedded C and assembly language
- Use of internal peripherals and interface and use of external peripherals
- Interfacing sensors and transducers to a microcontroller
- Creating the overall architecture, design, and implementation of an embedded system

Common Peripherals in User Operated Embedded Systems

- Operator input devices: switches, pushbuttons, keypads, optical shaft encoders
- Operator output devices: LEDs and LCD displays
- Sensors: temperature, pressure, light, humidity, and so on
- Analog-to-digital (A to D) and digital-to-analog (D to A) converters
- Serial and parallel interfaces to peripheral devices

Tentative Course Topics

- 1. Introduction
- 2. Bit Manipulation in C
- 3. Digital Input and Switch Debouncing
- 4. Keypad Scanning Hardware and Assembler Software
- 5. Keypad Scanning C Software
- 6. LCD Display Hardware
- 7. LCD Display Software
- 8. Sensors and Analog Information
- 9. ATmega128 Analog-to-Digital Converter
- 10. Common Operational Amplifier Signal Conditioning Circuits
- 11. Serial Peripheral Interface (SPI)
- 12. Digital-to-Analog Conversion
- 13. Analog-to-Digital Conversion
- 14. Pointers and Pointers to Functions
- 15. Table Driven Finite State Machines in C
- 16. ATmega128 Interrupts and Interrupt Driven Systems
- 17. Real Time and Calendar Clocks
- 18, Power on Self Test (POST)
- 19. Driving High Power Loads
- 20. Wireless Data Transfer
- 21. Storage Classes in Single and Multifile Programs
- 22. Parameter Passing and the Stack
- 23. Mixed C and Assembler Programs
- 24. Data Storage and Memory Models
- 25. External Memory Interface
- 26. Low Power Operation and Battery Powered Systems

Course Structure

- Lecture
  - Usually one primary topic will be covered per class meeting
  - A general overview of each topic will be provided
  - Overviews include example components, code, and/or applications
- Readings
  - Assigned readings from textbook, articles, application notes, data sheets, and compiler user manuals
- Discussion
  - Selected concepts from reading assignments
  - Issues related to laboratory modules and project
- Laboratory
  - Design modules are typically of two weeks duration each
  - Final project design and documentation

Laboratory

- Introductory design module of one week duration
- Five or six design modules of two weeks duration each
- Design modules culminate in a complete system
- User manual and design document required for the complete system

Exam Schedule and Grading

- Tentative schedule for exams:
  - First Exam Thursday, March 8th
  - Second Exam Thursday, April 26th
- Course grade computation:
  - Exams 40%
  - Laboratory 35%
  - Design Doc. and User's Manual 15%
  - Class Participation 10%

Source Material

- Either *Programming in C (3rd edition)* by Stephen Kochan (ISBN 978-0-672-32666-0) or *C Programming (2nd Addition)* by K. N. King (ISBN 978-0-393-97950-3) can be used for you to review introductory C topics and to learn some advance topics that you might not have previously mastered.
- *The AVR Microcontroller and Embedded System*, Mazidi, Naimi, and Naimi, Prentice Hall, Copyright 2011, ISBN 0-13-800331-9. (Text from ESE 380)
- Lecture Notes (provided on Blackboard)
- Application notes, data sheets, and IAR compiler user documentation (provided on Blackboard)

Design Environment for ESE 381

- ATmega128 Microcontroller
- ET AVR Stamp Module
  http://www.futurlec.com/ET-AVR_Stamp.shtml
- IAR Embedded Workbench Integrated Design Environment (IDE). Used in the Embedded System Design Laboratory (ESDL)
- KickStart Version of IAR Embedded Workbench. For use on students' PCs

Atmel’s AVR Family of Microcontrollers

- The AVR 8-bit family of microcontrollers consists of more than 220 devices that have the same core architecture but differ in the sizes of their Flash and SRAM memory, in the number and complexity of their on-chip peripherals, and temperature ranges
- Many of these devices have multiple package options
- In ESE 380 you used the ATmega16
- In ESE 381 we will use the ATmega128
- Because the core architectures are the same the assembly language instruction sets are the same

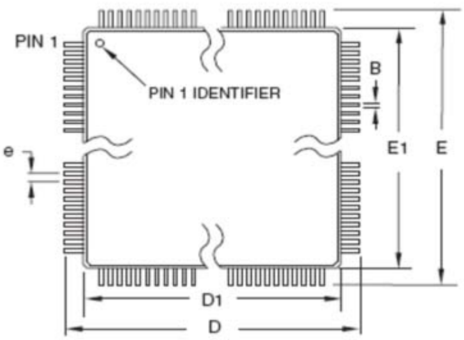
ATmega128 – Selected Features

- 128K Bytes of In-System Reprogrammable Flash
- 4K Bytes Internal SRAM (4 x ATmega16)
- 4K Bytes EEPROM (8 x ATmega16)
- Up to 64K Bytes Optional External Memory Space
- Two Expanded 16-bit Timer/Counters with Separate Prescalers, Compare Mode, and Capture Mode
- Real Time Counter with Separate Oscillator
- Dual Programmable Serial USARTs
- 53 Programmable I/O Lines
- 64-lead TQFP (thin quad flat pack) and 64-pad MLF (MicroLeadFrame)

ATmega128 Packaging and Pins

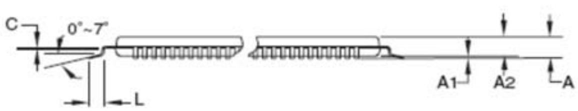
- AVR 8-bit microcontrollers come in packages that range from 8 to 100 pins
- ATmega16: TQFP 44, MLF 44, and PDIP 40
- ATmega128: TQFP 64 and MLF 64

Physical Layout of 64-pin TQFP



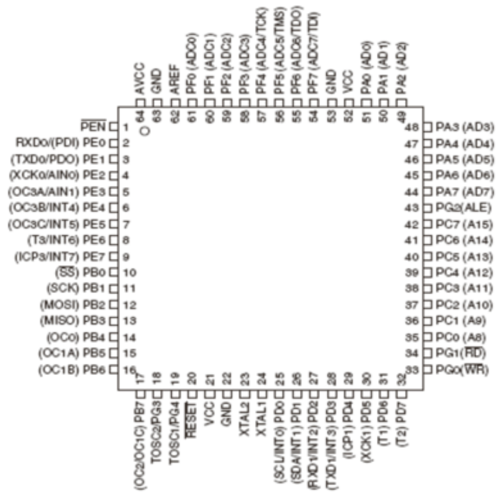
COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	1.20	
A1	0.05	–	0.15	
A2	0.95	1.00	1.05	
D	15.75	16.00	16.25	
D1	13.90	14.00	14.10	Note 2
E	15.75	16.00	16.25	
E1	13.90	14.00	14.10	Note 2
B	0.30	–	0.45	
C	0.09	–	0.20	
L	0.45	–	0.75	
e	0.80 TYP			



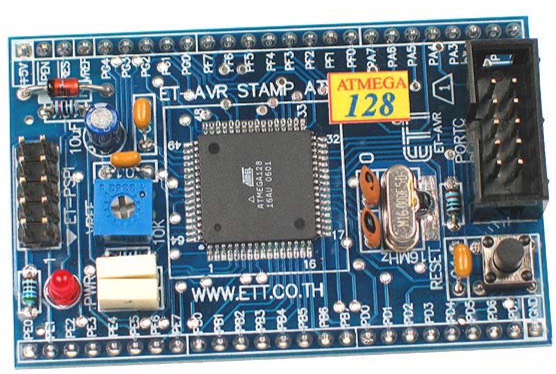
ATmega128 TQFP Pinout

Figure 1. Pinout ATmega128

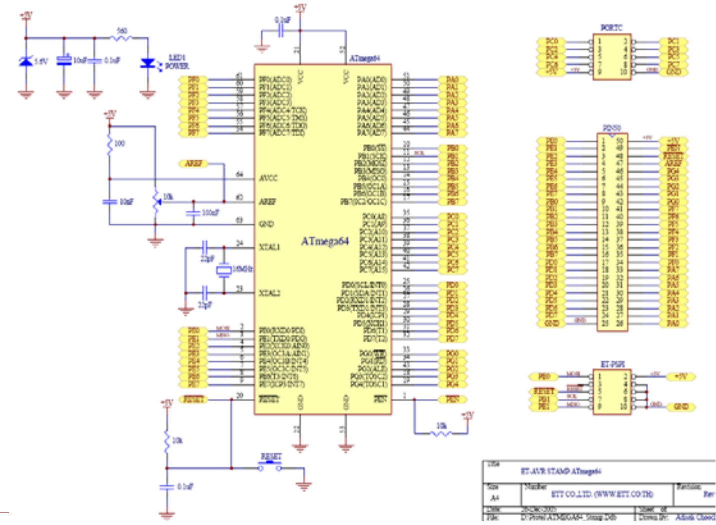


ET-AVR Stamp ATMega128 Board (breakout board or evaluation board)

- This semester we will use an ATmega128 microcontroller that is mounted on an ET-AVR Stamp board.



ET-AVR Stamp Schematic



IAR Embedded Workbench

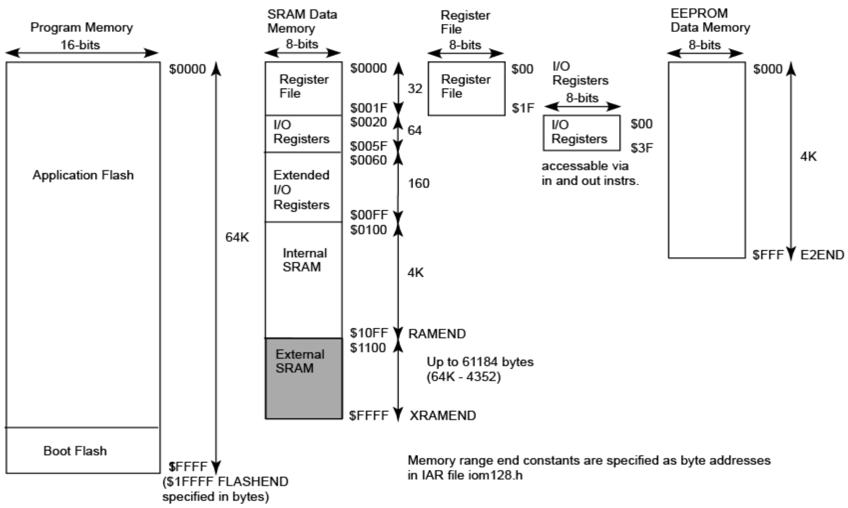
- IAR Embedded Workbench is a set of development tools for building and debugging embedded applications using assembler, C, and C++:
  - Highly optimizing AVR compiler supporting C and C++
  - Configuration files for all AVR Classic, ATmega, and FPSLIC (Field Programmable System Level Integrated Circuit) families, including devices with the enhanced core
  - JTAGICE debugger support
  - Run-time libraries
  - Relocating AVR assembler
  - Linker and librarian tools, C-SPY debugger with AVR simulator, and support for RTOS-aware debugging on hardware

Module 0: Introduction to the Design Environment

- The purpose of this module is to introduce you to the design environment you will use throughout the semester
- Unlike later modules, this module is only one week in duration
- You will interface a bank of switches to Port D and a bank of LEDs to Port B of the EV-AVR Stamp Board
- You will compile and debug C versions of programs similar to those of Laboratories 1 and 2 of ESE 380

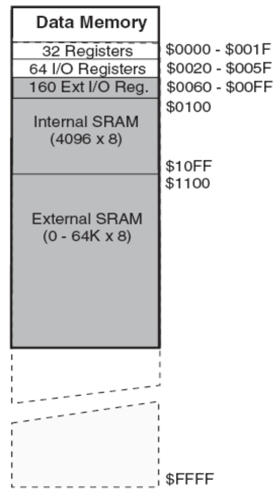
1/20/2016 © Copyright Kenneth Short 2006 21

AVR ATmega128 Address Maps



1/20/2016 © Copyright Kenneth Short 2006 22

I/O Ports



1/20/2016 © Copyright Kenneth Short 2006 23

Use and Discussion of C in this Course

- ESE 380, ESE 271, and a working knowledge of C (such as ESE 124) are prerequisites for this course
- The C related discussions in this course focus primarily on embedded C and mixed language (with assembler) aspects of the use of C

1/20/2016 © Copyright Kenneth Short 2006 24

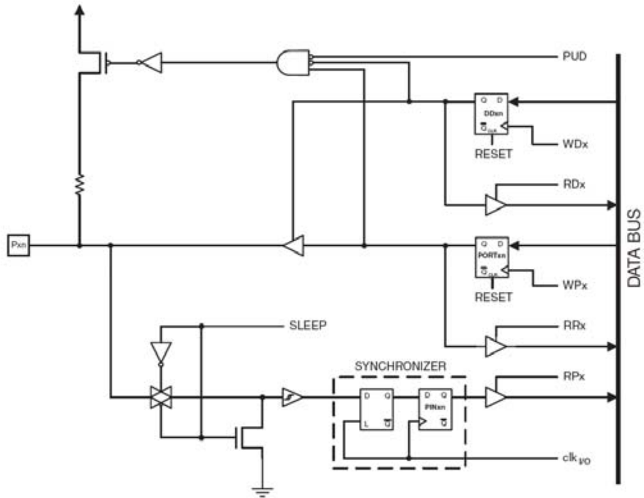
ATmega128 I/O Ports

- The ATmega16 has a maximum of 32 I/O pins associated with four 8-bit I/O ports (ports A, B, C, D)
- The ATmega128 has a maximum of 53 I/O pins associated with six 8-bit I/O ports (A, B, C, D, E, F) and one 5-bit I/O port (G)
- The ATmega128 has an ATmega103 compatibility mode, **which we do not use**. In this mode, Port F is an input only and Port G does not provide general I/O

Disable ATmega103 Compatibility Mode

- The ATmega128 has an ATmega103 compatibility mode for backward compatibility with the ATmega103 microcontroller
- This mode is enabled by fuse M103C
- In the 103 compatibility mode, none of the functions in the Extended I/O space are in use and Port F is input only and Port G is not available for general I/O
- Make sure that this mode is disabled when using JTAGICE. This is done in the IAR compiler in JTAGICE mkII > Fuse Handler > Extended Fuse (tab)

Review of AVR I/O Port Logic



Registers Associated with a Port

- Three registers and associated address locations are associated with each port
  - Data Direction Register – DDRx : (read/write) sets the direction of each port pin
  - Data Register – PORTx : (read/write) If a pin is configured as an output, PORTx sets the drive value of the pin. If a pin is configured as an input, PORTx enables/disables pull-up resistor
  - Input Pins – PINx : (read only) reads value at port pin

Registers Associated with a Port (cont.)

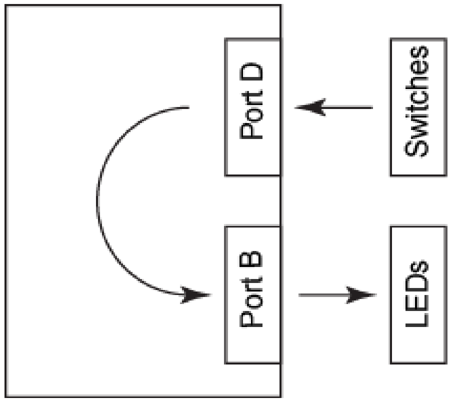
- DDRx and PORTx are both automatically initialized to \$00 during reset
- These initial values make Port X an input port with its internal pull-up resistors disabled. Therefore, only input ports that require pull-ups and output ports must be explicitly configured. However, for readability of our code we will explicitly configure all ports
- The three registers associated with a port are independent in that writing one of them has no effect on the contents of the other two

Configuring Port Pins

DDxn	PORTxn	PUD (in SFIOR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if externally pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

In and Out Task

- Assume that we want to read a byte of data from switches connected to Port D and display the value read on LEDs driven by Port B
- Configure Port D as an input and Port B as an output
- Read data from Port D
- Output a copy of the read data to Port B



In Out Program in Atmel AVR Assembly Language

```
.include "m16def.inc"  ;include microcontroller specific header file

reset:
    ; Configure PortB as an output port
    ldi r16, $FF        ;load r16 with all 1s
    out DDRB, r16       ;PortB - all bits configured as outputs

    ; Configure PortD as an input port with pull-up resistors
    ldi r16, $00        ;load r16 with all 0s
    out DDRD, r16       ;PortD - all bits configured as inputs
    ldi r16, $FF        ;enable pull-up resistors by outputting
    out PORTD, r16      ;all 1s to port

    ; Continually input switch values and output to LEDs
again:
    in r16, PIND        ;read switch values
    com r16             ;complement switch values to drive LEDs
    out PORTB, r16      ;output to LEDs values read from switches
    rjmp again          ;continually repeat last four instructions
```

In Out Program in C

```c
#include <iom128.h> //File with register addresses for ATmega128

int main(void)
{
  char temp;        //Input byte. unsigned char by default.

  DDRB = 0xFF;    //PORTB - all bits configured as outputs.
  DDRD = 0x00;    //PORTD - all bits configured as inputs.
  PORTD =0xFF;    //PORTD enable pullups

  while(1) {
    temp = PIND;    //Input byte from SWITCHES.
    temp = ~temp;    //Complement to drive LEDS
    PORTB = temp;   //Output byte to LEDS.
  }
}
```

1/20/2016 © Copyright Kenneth Short 2006 33

<iom128.h> Header File

- The iom128.h header file declares the internal register addresses for the ATmega128
- It uses macros from the file iomacro.h to assign names to register addresses and to bits in a particular register. The iom128.h header file contains a directive to “include” the iomacro.h file
- The macros use unions and field names in structures to assign names to bits
- You must include the iom128.h file at the beginning of all your programs to access registers or their bits by name:
  #include <iom128.h>

1/20/2016 © Copyright Kenneth Short 2006 34

C Integer Data Types

- The size (in bits) of C's basic data types are implementation dependent
- C uses type char to denote the size used to hold a system's character set. Therefore a char is not always 8-bits, it depends on the C implementation
- The representation of signed numbers is determined by the hardware, not C. However, the most commonly used representation is 2's complement (IAR's complier uses 2's complement)

Basic Integer Data Types in AVR IAR C

The following table gives the size and range of each integer data type:

Data type	Size	Range	Alignment
bool	8 bits	0 to 1	1
char	8 bits	0 to 255	1
signed char	8 bits	-128 to 127	1
unsigned char	8 bits	0 to 255	1
signed short	16 bits	-32768 to 32767	1
unsigned short	16 bits	0 to 65535	1
signed int	16 bits	-32768 to 32767	1
unsigned int	16 bits	0 to 65535	1
signed long	32 bits	-2^{31} to $2^{31}-1$	1
unsigned long	32 bits	0 to $2^{32}-1$	1
signed long long	64 bits	-2^{63} to $2^{63}-1$	1
unsigned long long	64 bits	0 to $2^{64}-1$	1

Table 31: Integer types

Signed variables are represented using the two's complement form.

Data Type Alignment

- Alignment constrains where a C data object can be stored in memory. That is, it places limitations on the value of the starting address of the stored object
- The address of a data object must be divisible by its alignment value
- Alignment is related to the limitations in how a given microcontroller can access memory
- Since the AVR microcontroller's SRAM memory is byte addressable it has no alignment restrictions, resulting in an alignment value of 1

Compiler List File

- When setting the options for a project you can choose to have the compiler generate a list file (options>C/C++ Compiler>List>Output list file
- The list file will show you the assembly language instructions that the compiler has generated for each C statement in your source program
- This is a convenient way to determine how the C compiler implements C instructions at the assembly level

In Out C Program Compiler List File Output

```
\                                In  segment CODE, align 2, keep-with-next
   9          int main(void)
\                     main:
  10          {
  11            char temp;        //Input byte. unsigned char by default.
  12
  13            DDRB = 0xFF;    //PORTB - all bits configured as outputs.
\   00000000   EF0F               LDI     R16, 255
\   00000002   BB07               OUT     0x17, R16
  14            DDRD = 0x00;    //PORTD - all bits configured as inputs.
\   00000004   E000               LDI     R16, 0
\   00000006   BB01               OUT     0x11, R16
  15            PORTD =0xFF;    //PORTD enable pullups
\   00000008   EF0F               LDI     R16, 255
\   0000000A   BB02               OUT     0x12, R16
  16
```

1/20/2016 39

In Out C Program Compiler List File Output (2)

```
  17            while(1) {
  18              temp = PIND;    //Input byte from SWITCHES.
\                     ??main_0:
\   0000000C   B300               IN      R16, 0x10
  19              temp = ~temp;     //Complement to drive LEDS
\   0000000E   9500               COM     R16
  20              PORTB = temp;   //Output byte to LEDS.
\   00000010   BB08               OUT     0x18, R16
\   00000012   CFFC               RJMP    ??main_0
\   00000014                      REQUIRE _A_PIND
\   00000014                      REQUIRE _A_DDRD
\   00000014                      REQUIRE _A_PORTD
\   00000014                      REQUIRE _A_DDRB
\   00000014                      REQUIRE _A_PORTB
  21            }
  22          }
```

1/20/2016 40

Compiler's Use of a Register Variable

- In the C program, variable temp is declared as a char:
  char temp;
- In a typical C program the compiler would usually allocate a byte of memory for this variable
- As can be seen in the previous slides the IAR C compiler has optimized the program by treating temp as a register variable (register storage class), the same as if temp had been declared as:
  register char temp;
- The compiler assigned variable temp to register R16 as can be seen from the assembly code the compiler produced

1/20/2016 © Copyright Kenneth Short 2006 41

Next Class

- Bit manipulation
- Bitwise logical operators
- Shift operators
- Intrinsic functions

1/20/2016 © Copyright Kenneth Short 2006 42

Reading Assignment

- Lecture 2: Bit Manipulation Techniques in C
- Chapter 1: Introduction and Chapter 2: C Fundamentals of King text