Topic 7. Introduction to Microcontrollers

CAB202. Topic 7

Luis Mejias

outline

- Microcontroller basic definitions
- Development board – QUT Teensy
- Operations with bits
- Programming cycle: compiling your code
- Example programs

Introduction to microcontrollers

- What is a microcontroller?
  - An integrated chip that is often part of an embedded system.
  - It is programmed to perform a single specific task/application. They are really just “mini-computers”.
  - A micro includes CPU, RAM, ROM, I/O ports and timers all in a single chip.

Introduction to microcontrollers

- What is a microcontroller?
  - They do not need to be powerful because most applications only require a clock of a few Mhz and a small amount of storage.
  - A microcontroller needs to be programmed to be useful
  - Microcontrollers are only as useful as the code written for it.
    - If you wanted to turn on a red light when the temperature reached a certain point, the programmer would have to explicitly specify how that will happen through his code

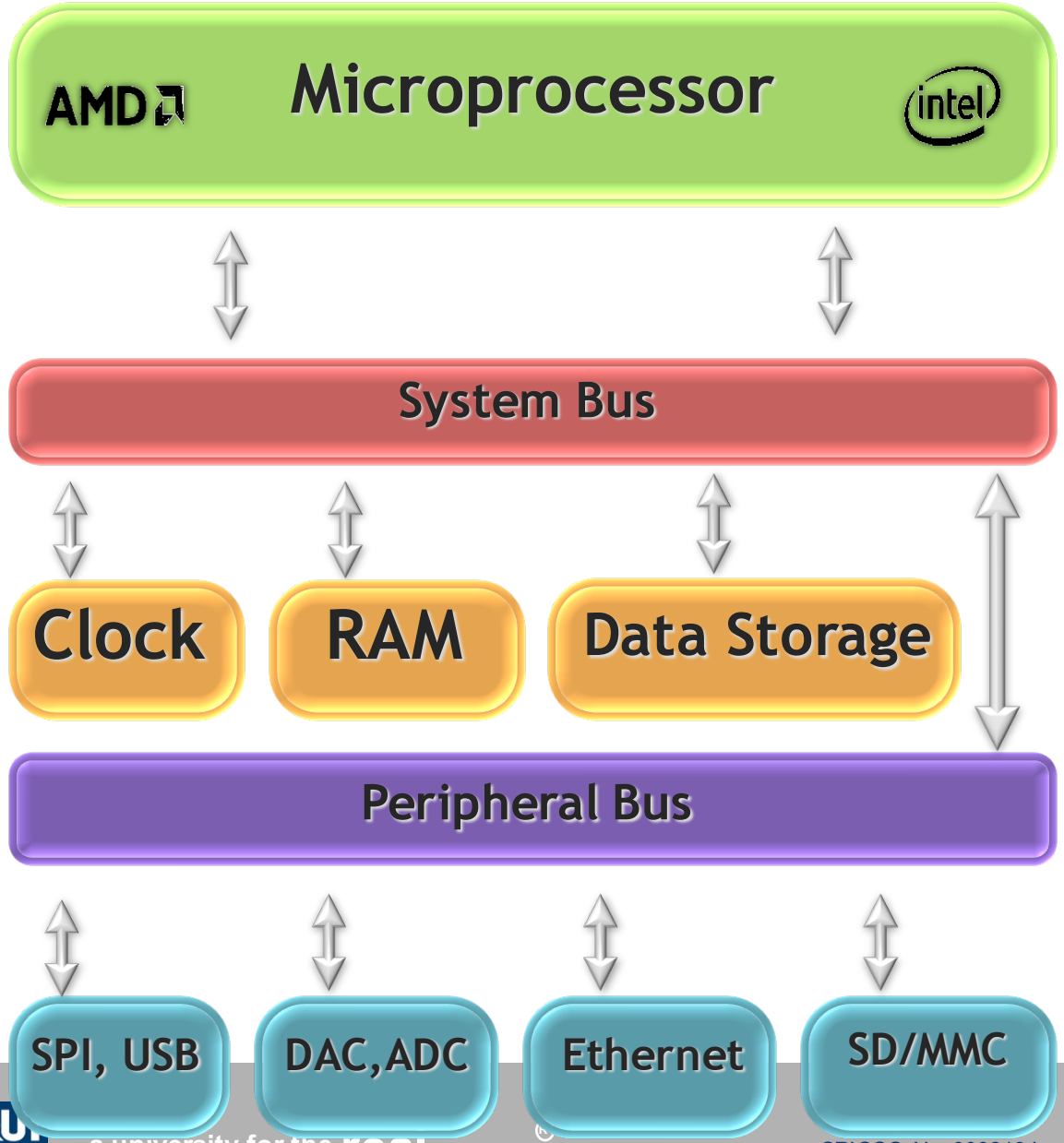
Introduction to microcontrollers

Where do you find them?

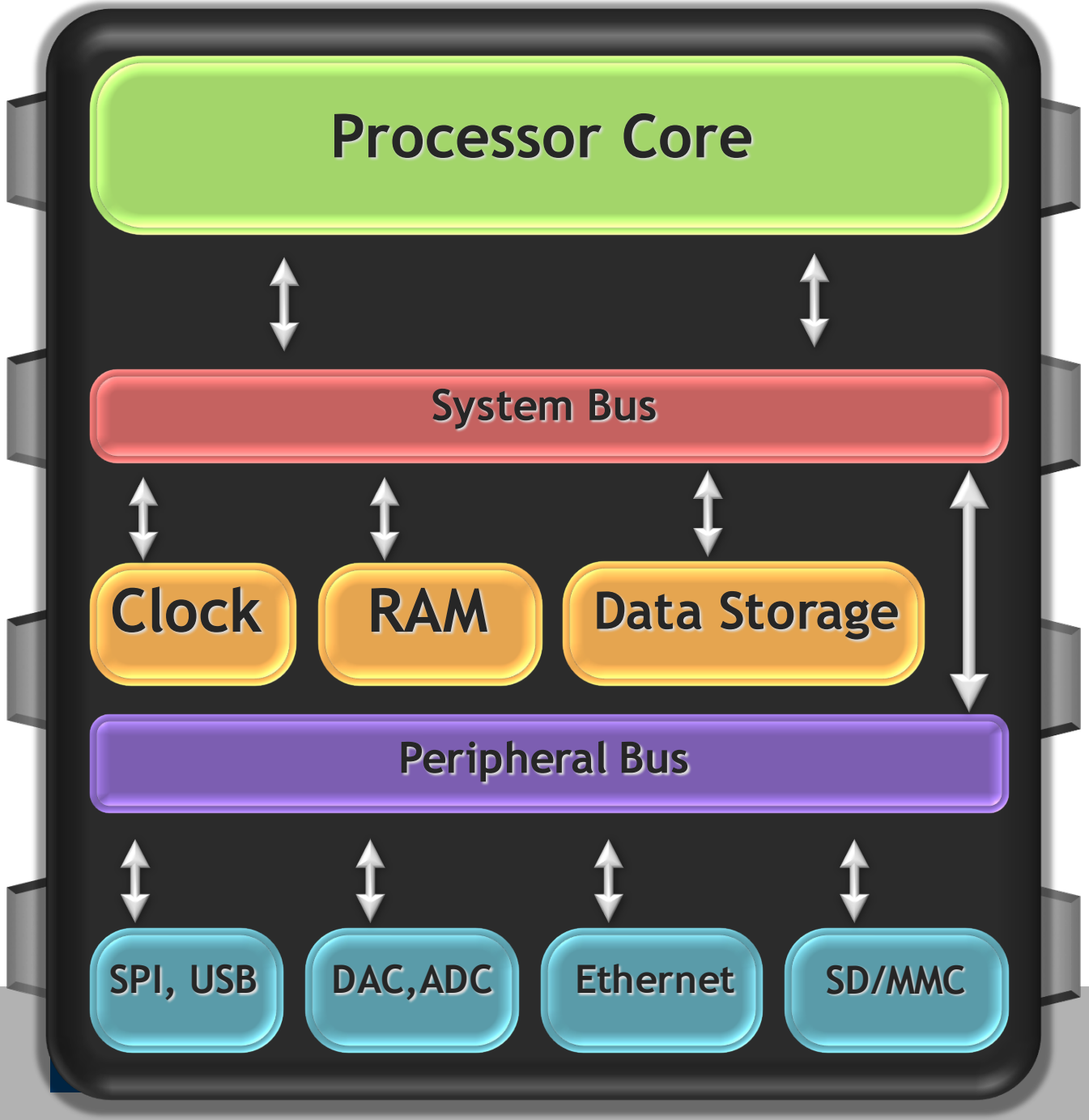
- Microcontrollers are hidden in tons of appliances, gadgets, and other electronics.
- They’re everywhere!

Introduction to microcontrollers

- Microcontroller programming
  1. Code is written for the microcontroller in an integrated development environment (IDE) or text editor. The code is written in a programming language (e.g c, basic or assembly).
  2. The IDE debugs the code for errors, and then compiles it into binary code which the micro-controller can execute.
  3. A programmer (a piece of hardware, not the person) is used to transfer the code from the pc to the microcontroller. The most common type of programmer is an ICSP (in-circuit serial programmer).



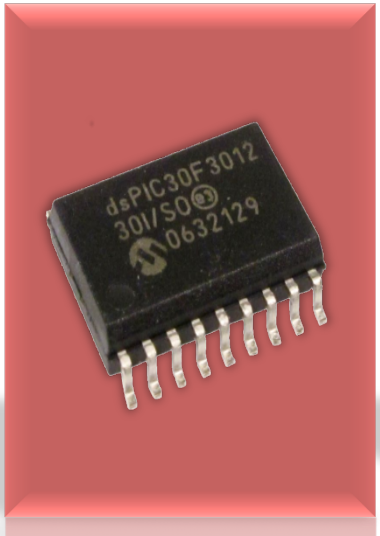
Microprocessor



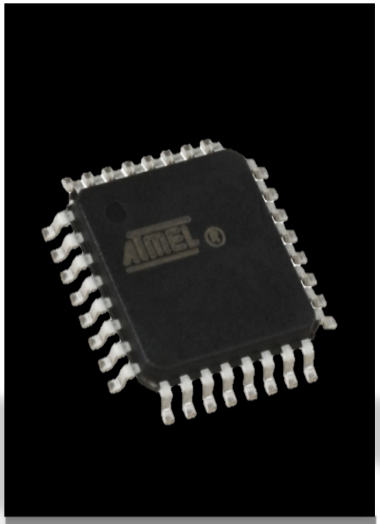
Microcontroller

Microcontroller Packaging

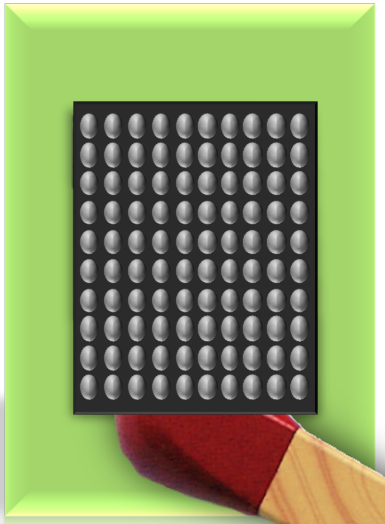
DIP
(Dual Inline Package)
Through hole
8 pins
9mm x 6mm
0.15pins/mm²



SOIC
(Small Outline IC)
Surface Mount
18 pins
11mm x 7mm
0.23pins/mm²



QFP
(Quad Flat Package)
Surface Mount
32 pins
7mm x 7mm
0.65pins/mm²



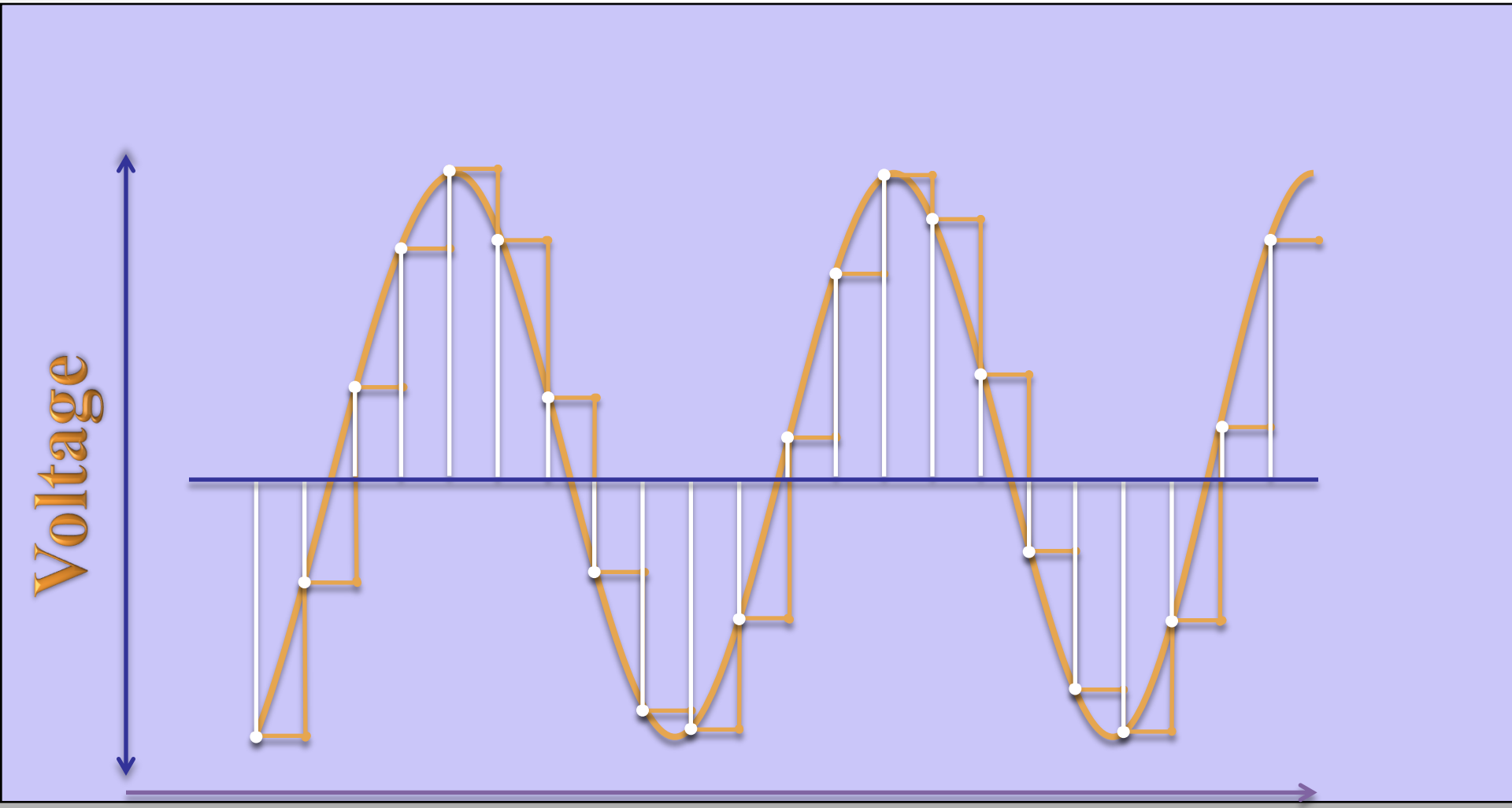
BGA
(Ball Grid Array)
Surface Mount
100 pins
6mm x 6mm
2.78pins/mm²

	Microprocessor	Microcontroller
Applications	General computing (i.e. Laptops, tablets)	Appliances, specialized devices
Speed	Very fast	Relatively slow
External Parts	Many	Few
Cost	High	Low
Energy Use	Medium to high	Very low to low
Vendors	intel AMD ARM	ATMEL ST Texas Instruments Microchip

The Analog to Digital Converter (ADC)

- Just about every modern microcontroller contains an ADC(s).
- It converts analog voltages into digital values.
- These digital representations of the signal at hand can be analyzed in code, logged in memory, or used in practically any other way possible.

The Analog to Digital Converter (ADC)



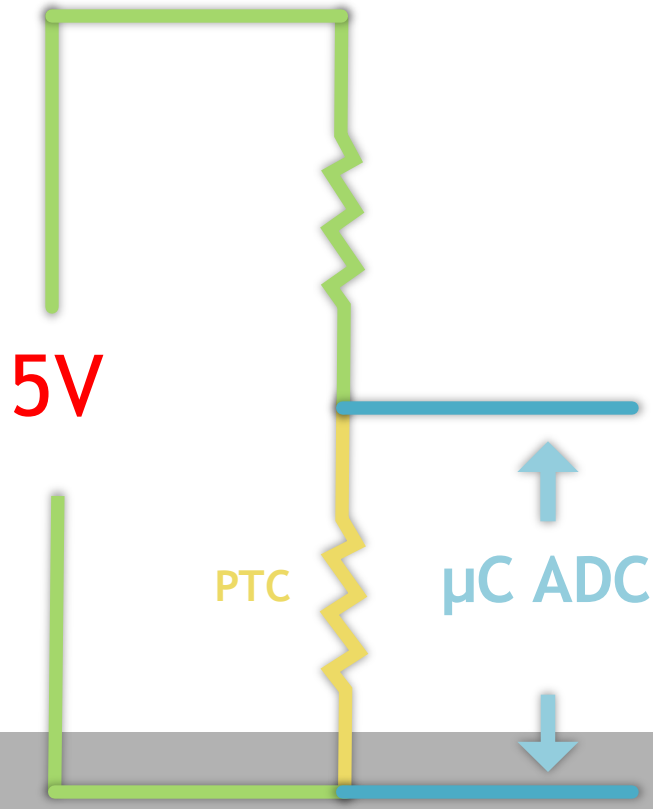
The Analog to Digital Converter (ADC)

PTC Specifications:

100Ω @ 25℃
+ 1Ω/ 1℃

(ex. @ 26℃, R = 101Ω
24℃, R = 99Ω

A 10-bit ADC will represent a voltage between 0 to 5 as a number between 0 to 1024.



code

```
Void Loop()

voltage25C  = 512
voltageADC  = ADC.input(pin1)

ratio = voltageADC / voltage25C
temperature  = ratio * 25
```

The Digital to Analog Converter (DAC)

- You guessed it! Microcontrollers have accompanying DACs.

- It does exactly the opposite function of an ADC. It takes a digital value and converts it into an pseudo-analog voltage.

- It can be used to do an enormous amount of things. One example is to synthesize a waveform. We can create an audio signal from a microcontroller. Imagine that!

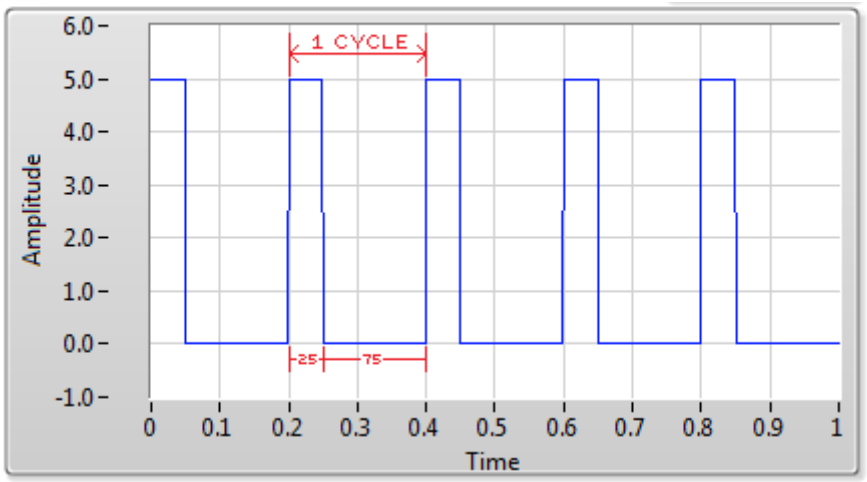
Pulse-Width Modulation (PWM)

- Modulation technique used to encode information into a signal, although its main use is for regulating power supplied to a load.

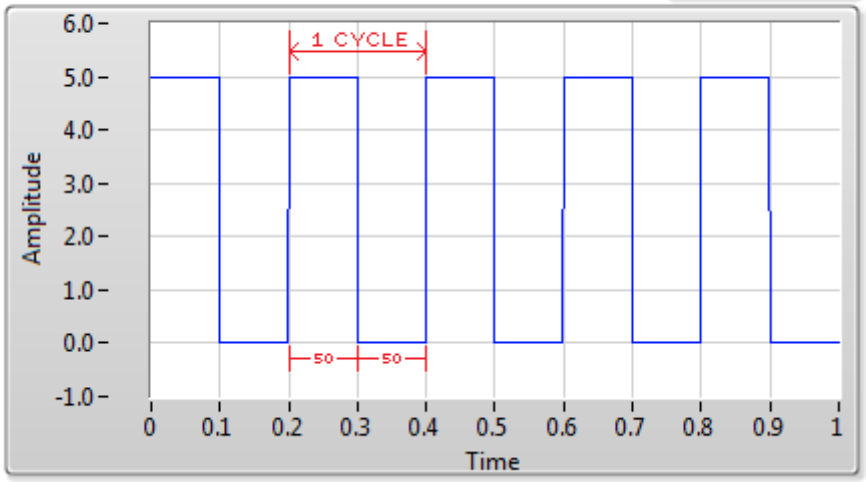
- An Analog signal can be generated using a digital source.

- Consist of two main components that define its behavior: a duty cycle and a frequency.
  - The duty cycle describes the amount of time the signal is in a high (on) stated as a percentage of the total time of it takes to complete one cycle.
  - The frequency determines how fast the PWM completes a cycle (i.e. 1000 Hz would be 1000 cycles per second), and therefore how fast it switches between high and low states.
- By cycling a digital signal off and on at a fast enough rate, and with a certain duty cycle, the output will appear to behave like a constant voltage analog signal when providing power to devices.

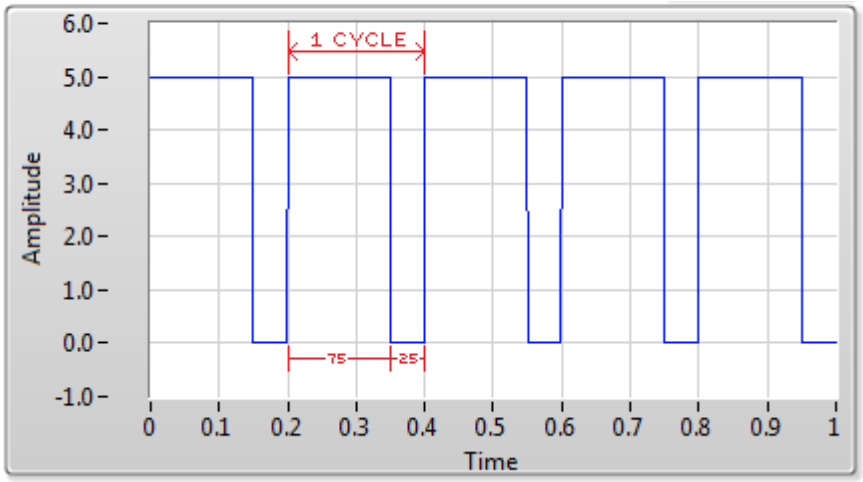
Pulse-Width Modulation (PWM)



25% duty cycle



50% duty cycle



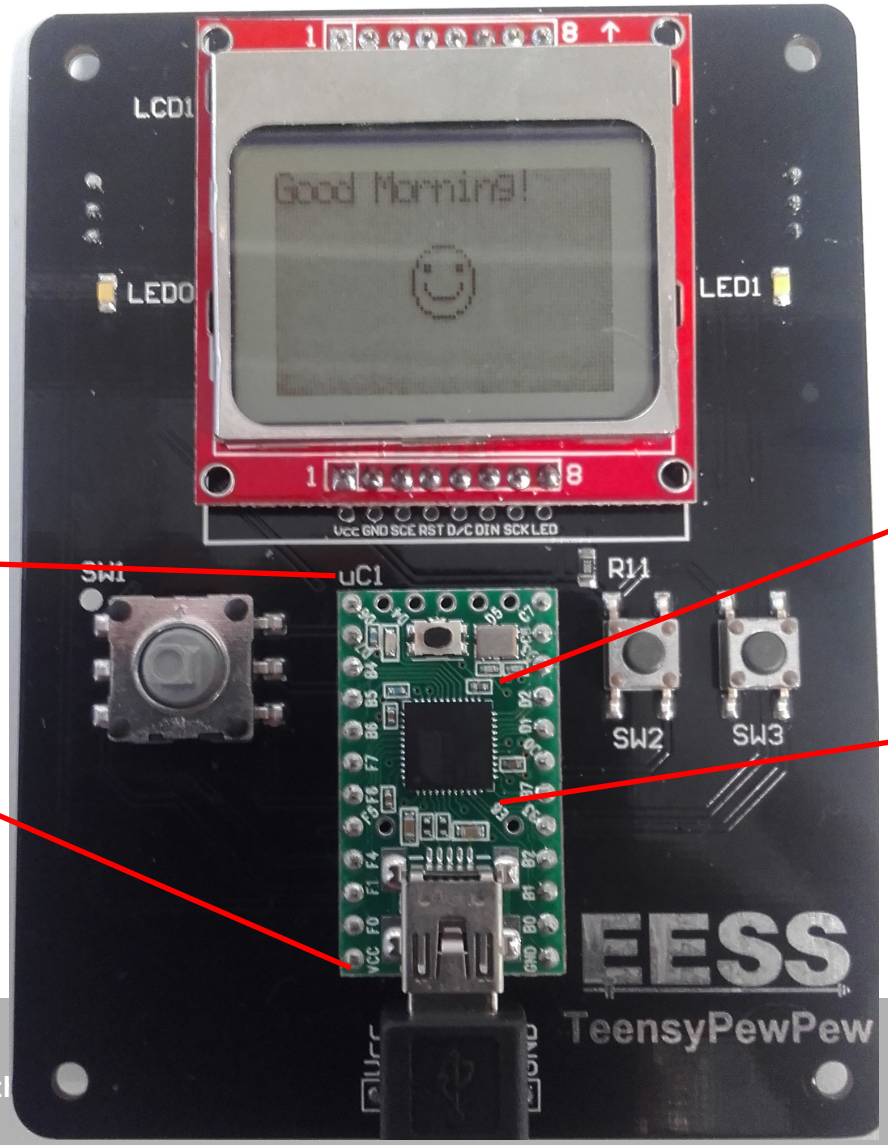
75% duty cycle

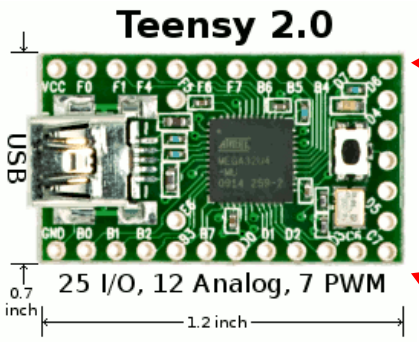
Microcontroller Applications

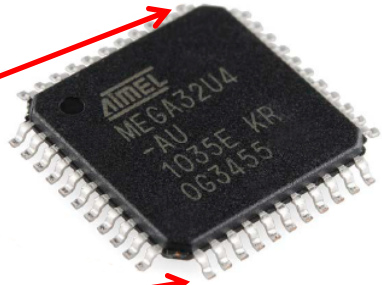
- Many robots use microcontrollers to allow robots to interact with the real world.

- Ex. If a proximity sensor senses an object near by, the microcontroller will know to stop its motors and then find an unobstructed path.

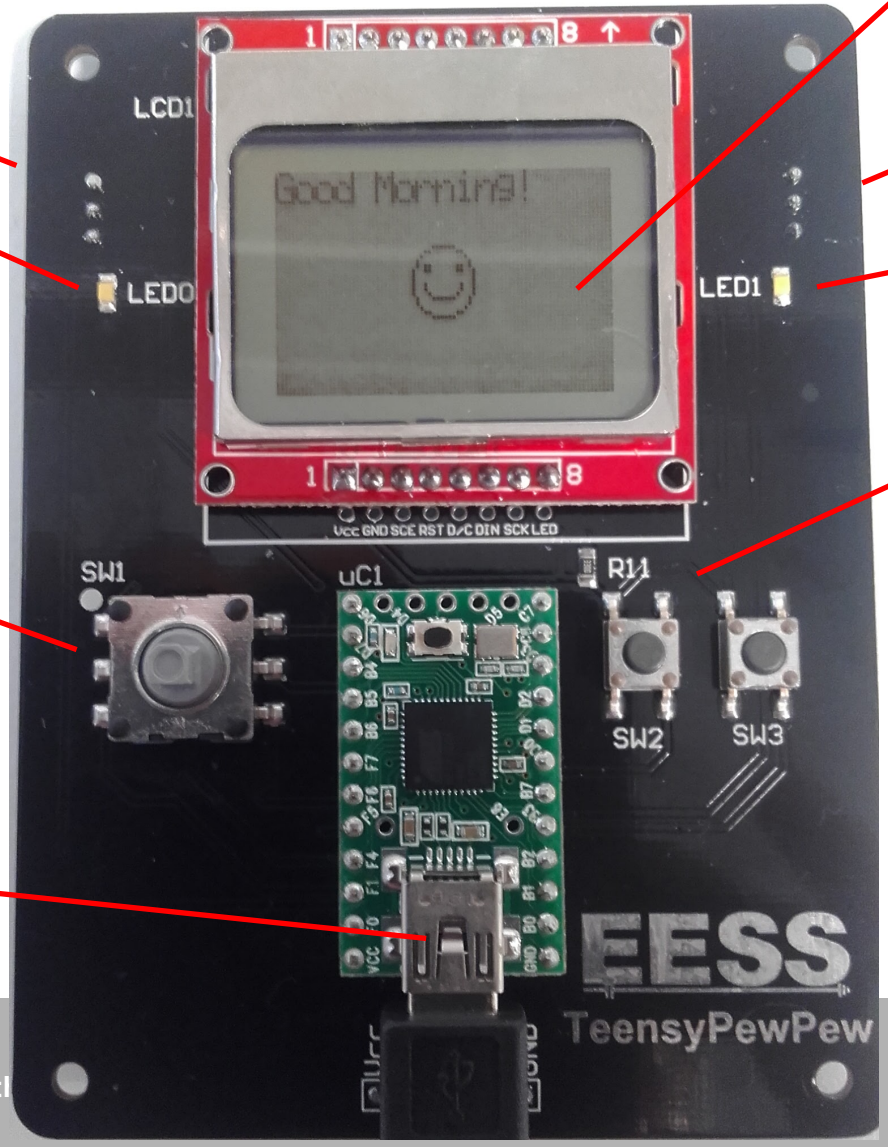
QUT Teensy Board



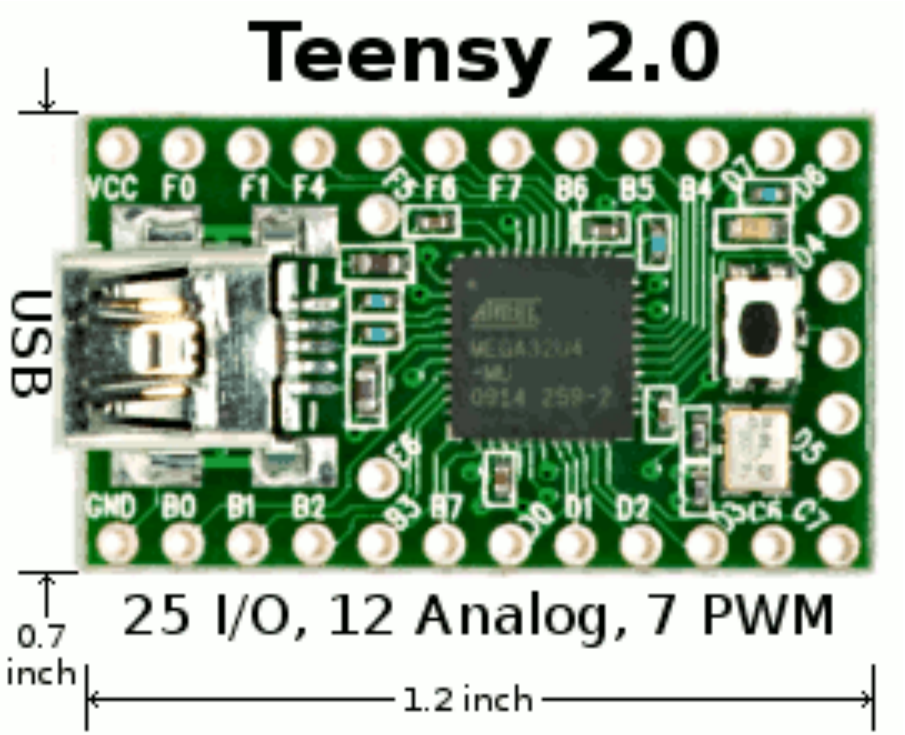




QUT Teensy Board



Teensy



http://www.pjrc.com/teensy/

Teensy

- Teensy 3.1, $19.80
- Teensy LC, $11.65
- Teensy 2.0, $16.00
- Teensy++ 2.0, $24.00

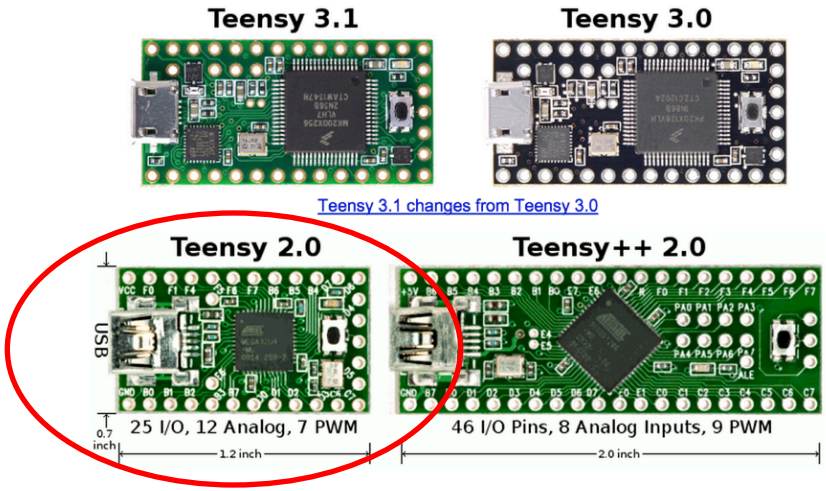
Teensy

- Main Page
- Teensy 3.1
- Teensy-LC
- **Getting Started**
- **How-To Tips**
- **Code Library**
- Projects
- **Teensyduino**
- **Reference**

Teensy USB Development Board

The Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port. No special programmer is needed, only a standard "Mini-B" USB cable and a PC or Macintosh with a USB port.

Update: Discussion / Support Forum



Key Features:

- USB can be any type of device
- AVR processor, 16 MHz
- Single pushbutton programming
- Easy to use Teensy Loader application
- Free software development tools
- Works with Mac OS X, Linux & Windows
- Tiny size, perfect for many projects
- Available with pins for solderless breadboard
- Very low cost & low cost shipping options

Specification	Teensy 2.0	Teensy++ 2.0	Teensy 3.0	Teensy 3.1
Processor	ATMEGA32U4 8 bit AVR 16 MHz	AT90USB1286 8 bit AVR 16 MHz	MK20DX128 32 bit ARM Cortex-M4 48 MHz	MK20DX256 32 bit ARM Cortex-M4 72 MHz
Flash Memory	32256	130048	131072	262144
RAM Memory	2560	8192	16384	65536
EEPROM	1024	4096	2048	2048
I/O	25, 5 Volt	46, 5 Volt	34, 3.3 Volt	34, 3.3V, 5V tol
Analog In	12	8	14	21
PWM	7	9	10	12
UART,I2C,SPI	1,1,1	1,1,1	3,1,1	3,2,1

https://www.pjrc.com/teensy/

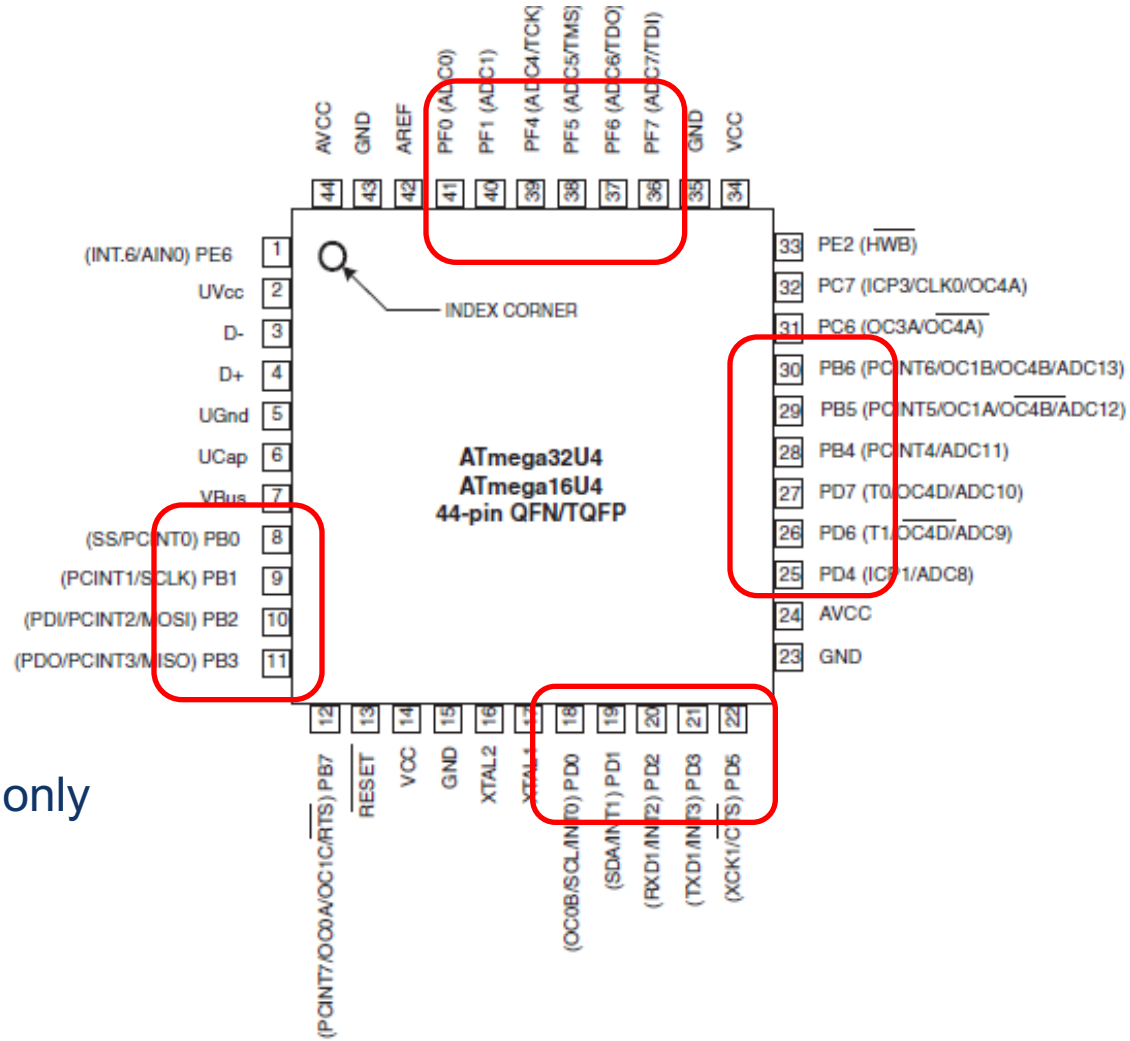
QUT Teensy Board - recap

- QUT teensy board consist of
  - A teensy USB dev board (with an ATmega 32U4 microcontroller)
  - Potensiometers
  - LED
  - Buttons
  - LCD

  - Let's examine the ATmega32U4 pinout

Teensy - ATmega32U4 pinout

- Atmega32U4 has three 8-bit digital IO ports
  - Ports B, D, and F
- Each port has 8 data pins
- Each port is bidirectional
  - Input or
  - Output
- Ports C and E also exist but only limited number of pins are available



Teensy - Atmega32U4 Datasheet

Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 135 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-Chip 2-cycle Multiplier
- Non-volatile Program and Data Memories
  - 16/32K Bytes of In-System Self-Programmable Flash (ATmega16U4/ATmega32U4)
  - 1.25/2.5K Bytes Internal SRAM (ATmega16U4/ATmega32U4)
  - 512Bytes/1K Bytes Internal EEPROM (ATmega16U4/ATmega32U4)
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/ 100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
    - All supplied parts are preprogramed with a default USB bootloader
  - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- USB 2.0 Full-speed/Low Speed Device Module with Interrupt on Transfer Completion
  - Complies fully with Universal Serial Bus Specification Rev 2.0
  - Supports data transfer rates up to 12 Mbit/s and 1.5 Mbit/s
  - Endpoint 0 for Control Transfers: up to 64-bytes
  - 6 Programmable Endpoints with IN or Out Directions and with Bulk, Interrupt or Isochronous Transfers
  - Configurable Endpoints size up to 256 bytes in double bank mode
  - Fully independent 832 bytes USB DPRAM for endpoint memory allocation
  - Suspend/Resume Interrupts
  - CPU Reset possible on USB Bus Reset detection
  - 48 MHz from PLL for Full-speed Bus Operation
  - USB Bus Connection/Disconnection on Microcontroller Request
  - Crystal-less operation for Low Speed mode
- Peripheral Features
  - On-chip PLL for USB and High Speed Timer: 32 up to 96 MHz operation
  - One 8-bit Timer/Counter with Separate Prescaler and Compare Mode
  - Two 16-bit Timer/Counter with Separate Prescaler, Compare- and Capture Mode
  - One 10-bit High-Speed Timer/Counter with PLL (64 MHz) and Compare Mode
  - Four 8-bit PWM Channels
  - Four PWM Channels with Programmable Resolution from 2 to 16 Bits
  - Six PWM Channels for High Speed Operation, with Programmable Resolution from 2 to 11 Bits
  - Output Compare Modulator
  - 12-channels, 10-bit ADC (features Differential Channels with Programmable Gain)
  - Programmable Serial USART with Hardware Flow Control
  - Master/Slave SPI Serial Interface

ATMEL

8-bit AVR® Microcontroller with 16/32K Bytes of ISP Flash and USB Controller

ATmega16U4
ATmega32U4

Preliminary

7766F–AVR–11/10

ATMEL

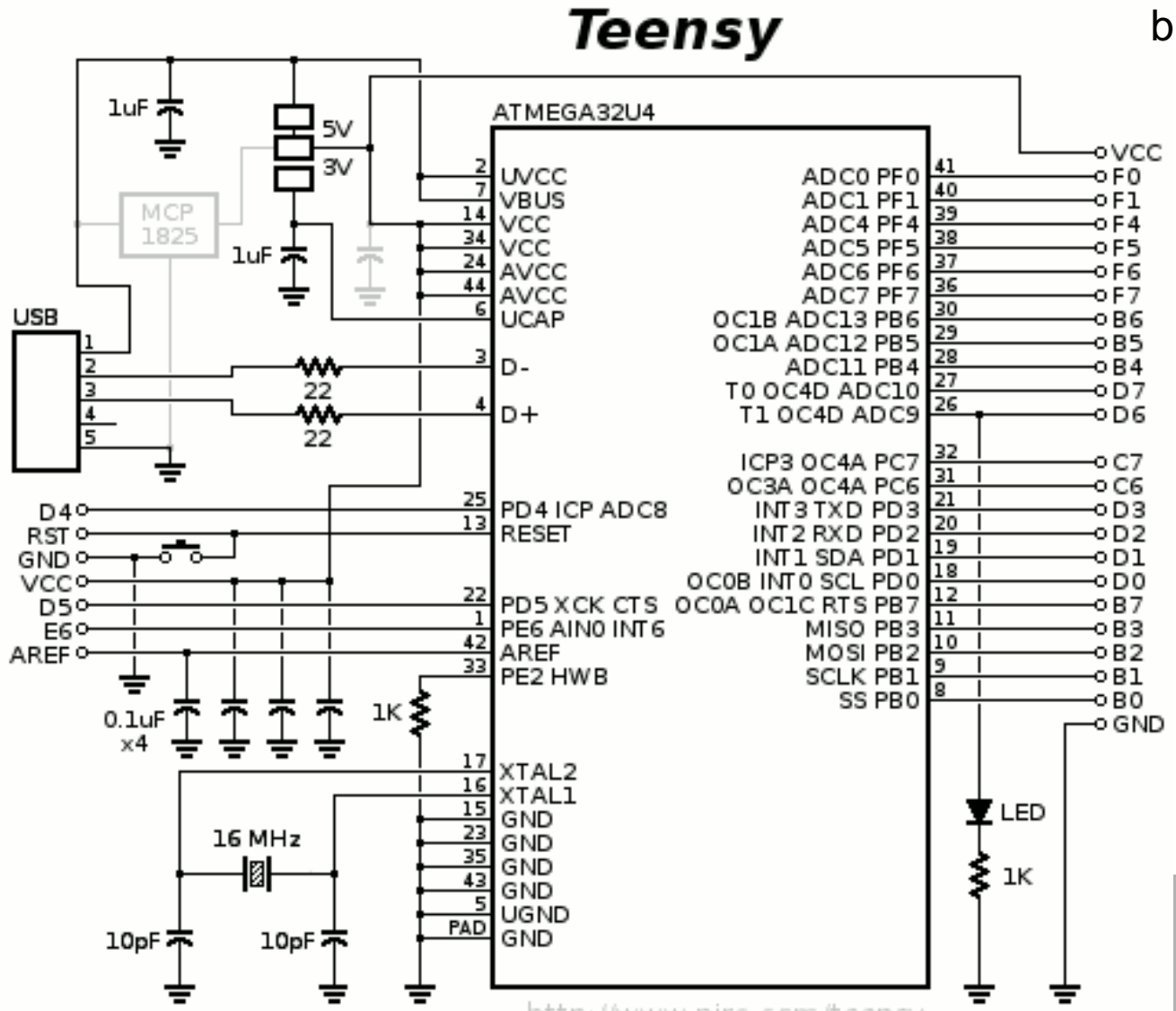
- Datasheet can be found on blackboard under Learning Resources – Microcontrollers

and under

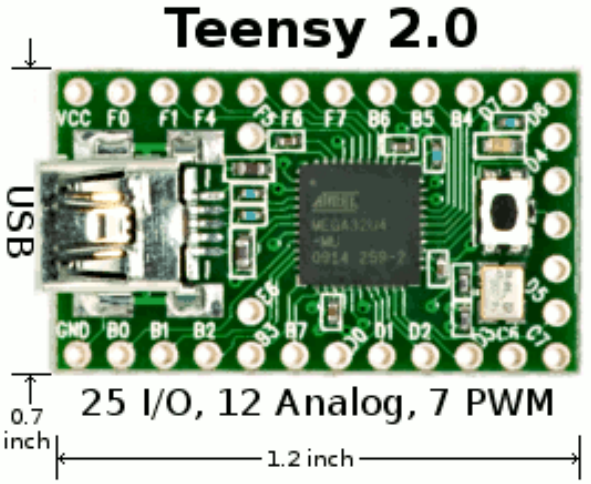
- Topic 7 – Intro to microcontrollers

Teensy - Atmega32U4 pinout

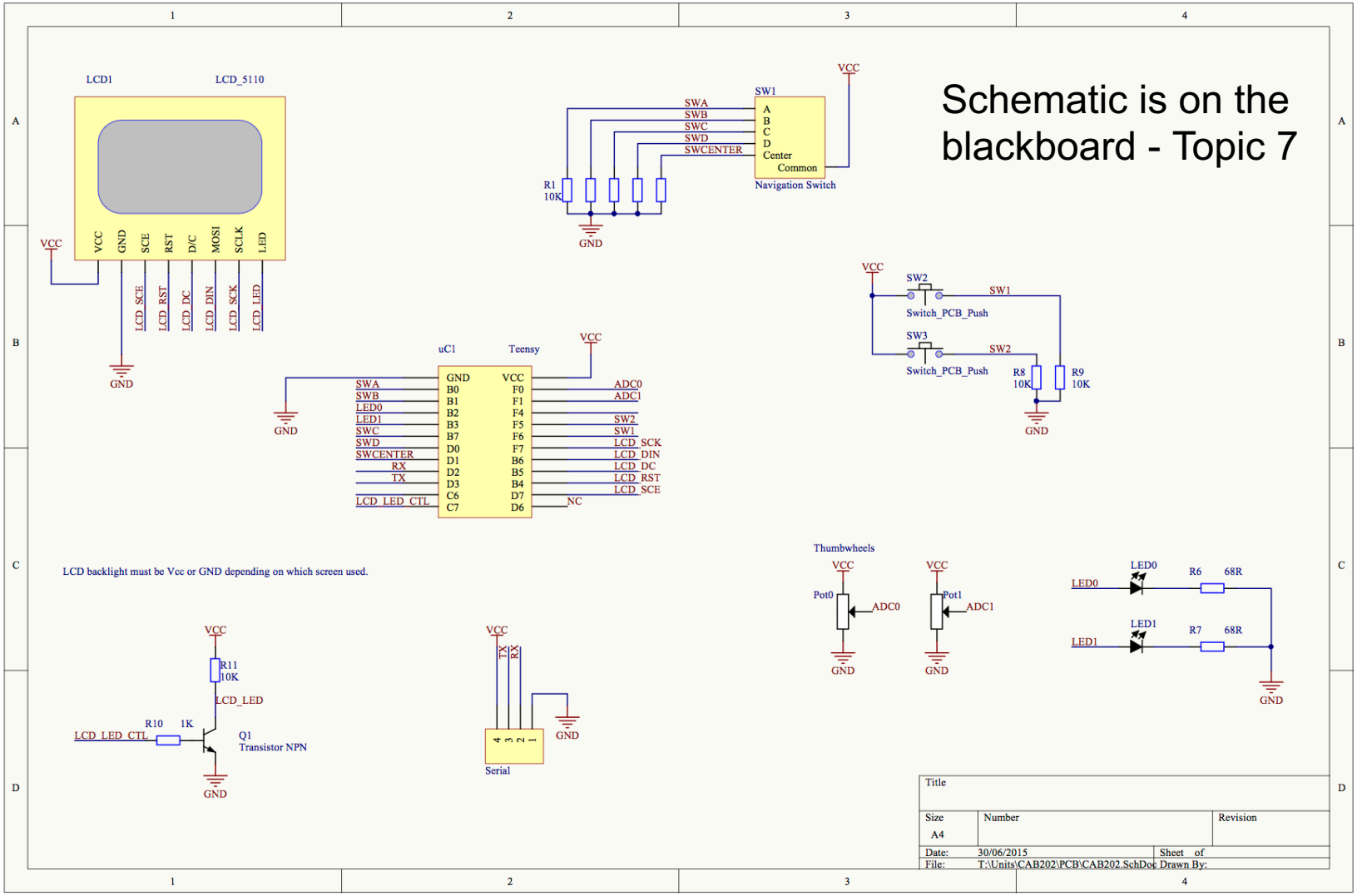
Schematic is on the blackboard - Topic 7



Schematic
just for the teensy 2.0
(without the QUT board)



Teensy - Atmega32U4 pinout



Teensy - Atmega32U4 Configuring Data Pins

- For each port, there are three 8 bit registers to control pin functionality
  - Data Direction Register (DDRx) – used to configure the port
  - Input Pins Address (PINx) – used to read values from the port
  - Data Register (PORTx) - used to write values to a port

where x denotes B,D, or F

Page 66 ATmega datasheet

Teensy - Atmega32U4 Registers

- Registers are special storage with 8 bits capacity

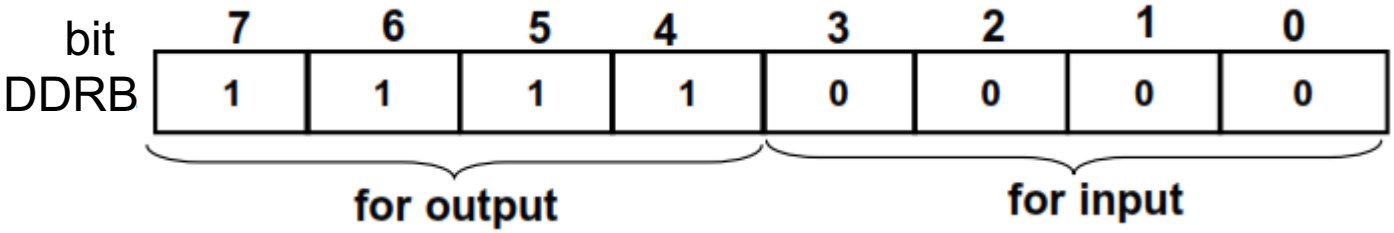
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

- The special character of registers, compared to other storage locations, is that
  - they are connected directly to the central processing unit called the accumulator,
  - they can be used directly in assembler instructions, either as a target register for the result or as read register for a calculation or transfer,
  - operations on their content require only a single instruction.

Teensy - Atmega32U4
Data Direction Register (DDRx)

- Configures a port to be output (1) or input (0)
  - For example, if we want to set Port B pins 0 to 3 for input and pins 4-7 for output, we would write the following C code:

```
DDRB = 0b11110000;
```



Teensy - Atmega32U4 Data Register (PORTx)

- Writes output data to port
  - For example, if want to write a binary 0 to output pin 6, binary 1 to other pins of Port B, we write C code:

```
PORTB = 0b10111111;
```

Teensy - Atmega32U4 Input Pins Address (PINx)

- Reads input data from port
  - For example, if we want to read the input pins of Port B, we write C code:

```
unsigned char temp; // temporary variable
temp = PINB;        // read input
```

Operations with Bits - Intro

- Covered in detail in tutorial week 8.
- Binary operations
  - Bit shift: this operation move/shift bits in a particular direction (either left or right)
  - There are two bit shift operators
  - The left shift operator <<
  - The right shift operator >>
  - These operators cause the bits in the left operand to be shifted left or right by the number of positions specified by the right operand.

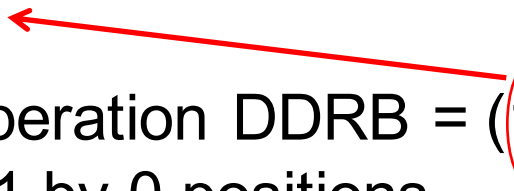
    In a left shift, bits get "shifted out" on the left and 0 bits get "shifted in" on the right.
    The opposite goes for a right shift

Operations with Bits - Intro

- Binary operations
  - Bit shift:
  - Let's say DDRB has an initial value of 0b00000000

In binary is 0b00000001

DDRB = 0b00000000;



  - The operation DDRB = (1<<0); shift bits in the binary value 1 by 0 positions

DDRB = 0b00000001;

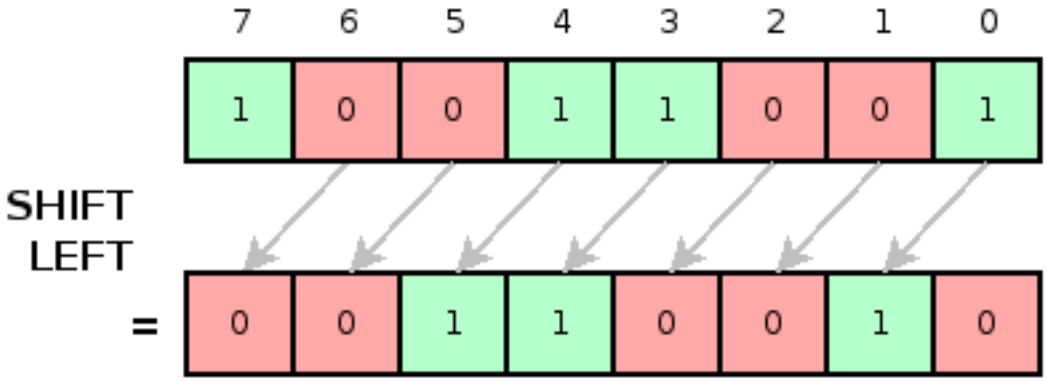
Operations with Bits - Intro

- Binary operations
  - Bit shift:
  - Let's say DDRB has an initial value of 0b10011001

DDRB = 0b10011001;

  - The operation DDRA = (DDRB<<1); shift bits in the binary value DDRB by 1 position

DDRB = 0b00110010;



Operations with Bits - Intro

- Binary operations
  - NOT operation: Also known as a one's complement. The NOT operation will simply negate each bit. Every 1 becomes 0 and every 0 becomes 1.

```
DDRB = ~(0b10011001<<1);
```

```
(0b10011001<<1) =    0b00110010
~(0b10011001<<1) = ~(0b00110010) =
                         = 0b11001101
```

Operations with Bits - Intro

- Binary operations
  - NOT operation:

DDRB = ~(1<<3);

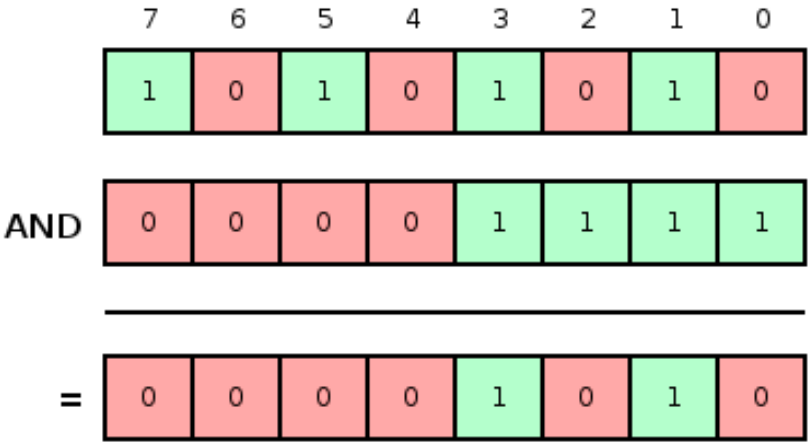
(0b00000001<<3) = 0b00001000

~(0b00000001<<3) = ~(0b00001000) =

= 0b11110111

Operations with Bits - Intro

- Binary operations
  - AND operation: this operation results in bits being set only if the same bits are set in both of the operands. In other words: bit n will be set in the result if bit n is set in the first operand and the second operand.

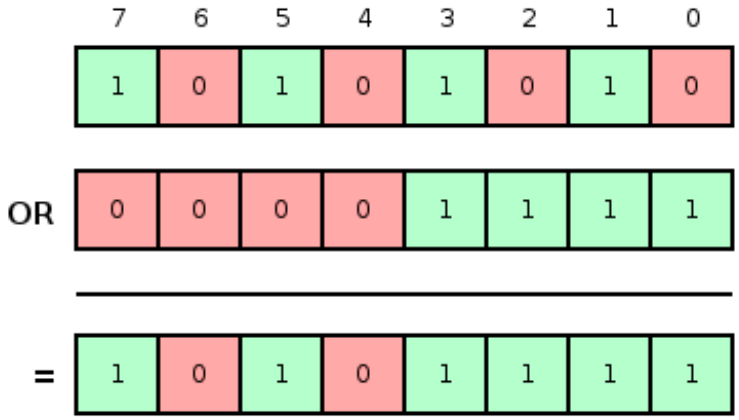


DDRB = 0b10101010 & 0b00001111

DDRB = 0b00001010

Operations with Bits - Intro

- Binary operations
  - OR operation: This results in bits being set if the same bits are set in either of the operands. In other words: bit n will be set in the result if bit n is set in the first operand or the second operand



DDRB = 0b10101010 & 0b00001111

DDRB = 0b10101111

Programming Cycle

1. First off, you have to write a program that tells the chip what to do.

2. Then you have to compile it, that is, turn the program description into machine code.

3. Next you program the chip using a programmer, which will transfer the machine code to the device

4. Test, debug, repeat!

5. First step is setting up your computer for programming, so follow the setup steps on Blackboard under Topic 7 – Intro to Microcontroller

Programming Cycle

- When code is compiled in your terminal, it is turned into binary
- Under windows, these programs (often called *applications* or *executables*) are often named ending with **.exe** (for example notepad.exe or winword.exe), on Macs, they are often named ending with **.App** (although the Finder hides it).
- For microcontrollers, binary files end in **.hex** (short for Intel Hex Format) There are other formats possible, but this is pretty much standard.
- Your compiler will generate a **.hex** file from code, and then all you have to do is transfer that **.hex** program to the chip!
- Teensy getting started: https://www.pjrc.com/teensy/first_use.html

Programming Cycle

- AVR chip has a small amount of flash memory in it. That memory is where the program is stored.
- When the chip starts up (you give it power) it starts running whatever program is in the flash.
- So all we have to do is to figure out how to write to that flash
- The Teensy Loader program communicates with your Teensy board so you can download new programs and run them.

Compiling your program

- Once you have your environment set up (installed **avr-gcc),** you can compile through the following process in a terminal window. Make sure the terminal is in the same directory as your source code. The compilation for the Teensy microcontroller is done with the following **avr-gcc** commands:

Windows. Note: tested on windows xp only	
# avr-gcc –c –mmcu=atmega32u4 –Os –DF_CPU=8000000UL <C source files> -o <applicationName>.o	
# avr-objcopy –O ihex <applicationName>.o <applicationName>.hex	Once prev. command finishes, use the following command to complete compilation and produce a ***.hex** file:
Mac	
# avr-gcc –c –mmcu=atmega32u4 –Os –DF_CPU=8000000UL <C source files> -o <applicationName>.o	
# avr-gcc –mmcu=atmega32u4 –Os –DF_CPU=8000000UL <C source files>.o -o <applicationName>.elf	
# avr-objcopy –O ihex <applicationName>.elf <applicationName>.hex	Once prev. command finishes, use the following command to complete compilation and produce a ***.hex** file:

Compiling your program

- So for a **button** program you would use the following two lines in a terminal to create a ***.hex** file ready for upload to the Teensy with the Teensy uploader application:

Windows. Note: tested on windows xp only	
# avr-gcc –c –mmcu=atmega32u4 –Os –DF_CPU=8000000UL button.c –o button.o	
# avr-objcopy –O ihex button.o button.hex	Once prev. command finishes, use the following command to complete compilation and produce a ***.hex** file:
Mac	
# avr-gcc –c –mmcu=atmega32u4 –Os –DF_CPU=8000000UL button.c –o button.o	
# avr-gcc –mmcu=atmega32u4 –Os –DF_CPU=8000000UL button.o –o button.elf	
# avr-objcopy –O ihex button.elf button.hex	Once prev. command finishes, use the following command to complete compilation and produce a ***.hex** file:

Compiling your program

- Using Makefile
  - For simple projects you will only need to change the line TARGET = name_of_file with the name of your file.
  - For example, if you wanted to compile turn_on_led.c for your microcontroller, you would change the line to: TARGET = turn_on_led
  - Copy the Makefile into the directory where turn_on_led.c is and then run "make".
  - The Makefile Template is in the Topic 7 – Introduction to Microcontrollers on Blackboard

Teensy
Let's write some code

- LED On/Off
  - Turn led on once.
- Buttons
  - Turn led on when button is pressed

Summary

- Key learning objectives:
  - Differences between microcontroller and CPU
  - Operations with bits: bitwise operator to handle reading and setting individual pins in a given port
  - Basic compiling commands: with or without make
- Example programs