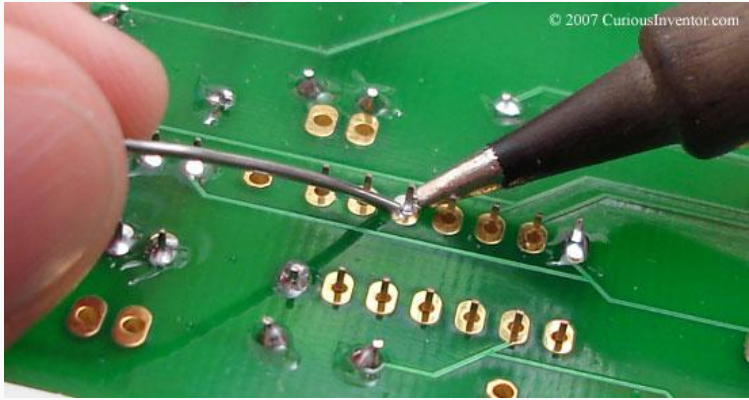
UCLA Rocket Project Electronics Workshop 5

2016/10/28

Next week - Probably Soldering!

1. Some theory of soldering
2. Coverage of most common mistakes and reasoning behind not making them
3. Invitation to practise



Arduino Libraries

1. What is a library?
2. Servo library example
3. External Libraries
4. Compilation Pipeline
5. Makefile
6. Compiling Arduino Libraries

What is a library?

1. Code - like everything else
2. Code written to provide some **functionality**
3. If C++, then library consists of objects with functions
4. If C, then library consists of functions
5. Using libraries is unavoidable, writing own code is slow
6. To learn how to use library code - examples, documentation

Servo library example

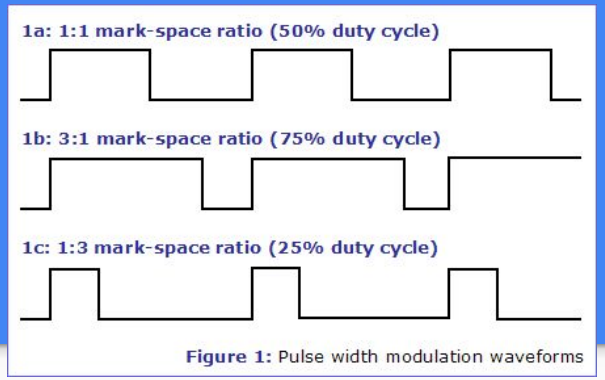


Figure 1: Pulse width modulation waveforms

1. Servomotors are controlled by PWM - pulse width modulation
2. PWM modulation is fairly difficult to set up
3. Calibration requires reading the servomotor datasheet
4. Arduino library already implementing that functionality

Using a library - guide

1. Confirm that the library solves the problem
2. Look at the examples (with pinout)
3. Look at the documentation if necessary
4. Experiment by making small, incremental changes to the example code
5. Write own code through incremental changes

Using the Servo library - 1. Confirm

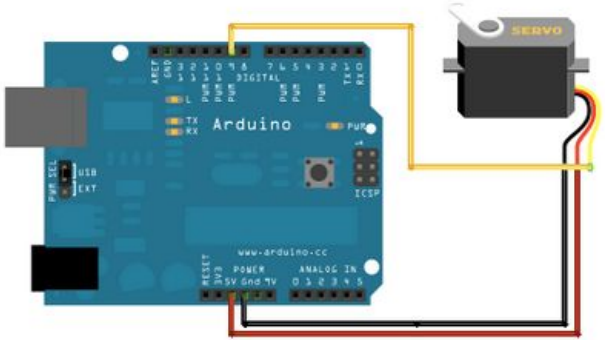
Servo library

This library allows an Arduino board to control RC (hobby) servo motors. Servos have integrated gears and a shaft that can be precisely controlled. Standard servos allow the shaft to be positioned at various angles, usually between 0 and 180 degrees. Continuous rotation servos allow the rotation of the shaft to be set to various speeds.

Excerpt from Arduino website servo library

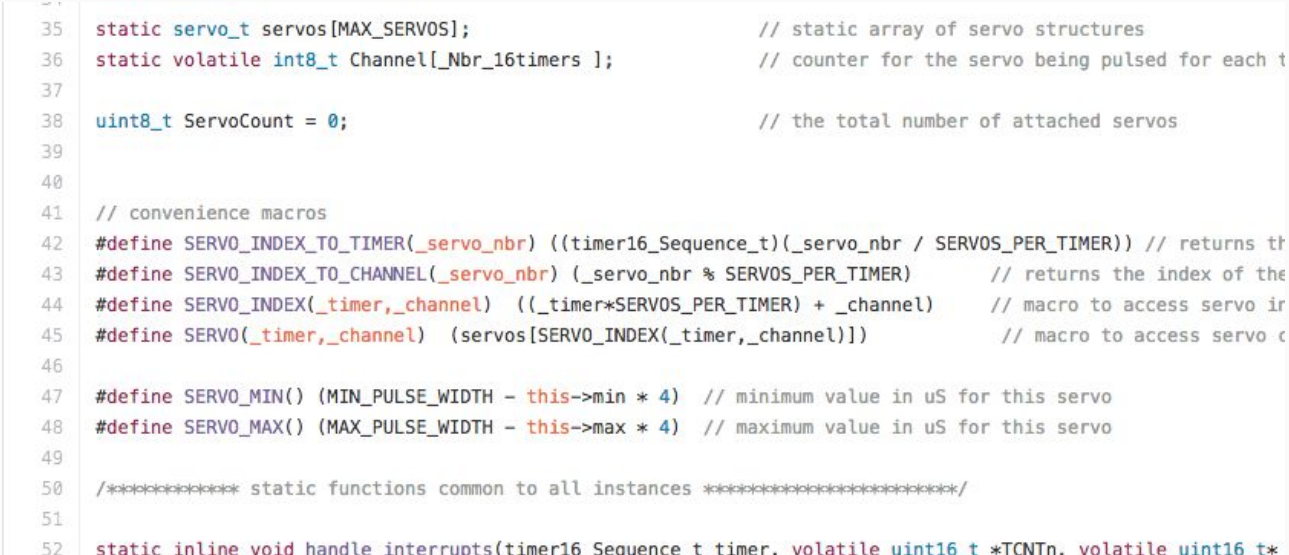
Using the Servo library - 2. Examples

```
#include <Servo.h>

Servo myservo;  // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0;    // variable to store the servo position

void setup() {
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
}
```



Using the Servo library - 3. Documentation

1. Arduino.cc and Arduino source code available on github



```
35  static servo_t servos[MAX_SERVOS];                          // static array of servo structures
36  static volatile int8_t Channel[_Nbr_16timers ];             // counter for the servo being pulsed for each t
37
38  uint8_t ServoCount = 0;                                     // the total number of attached servos
39
40
41  // convenience macros
42  #define SERVO_INDEX_TO_TIMER(_servo_nbr) ((timer16_Sequence_t)(_servo_nbr / SERVOS_PER_TIMER)) // returns th
43  #define SERVO_INDEX_TO_CHANNEL(_servo_nbr) (_servo_nbr % SERVOS_PER_TIMER)       // returns the index of the
44  #define SERVO_INDEX(_timer,_channel)  ((_timer*SERVOS_PER_TIMER) + _channel)     // macro to access servo in
45  #define SERVO(_timer,_channel)  (servos[SERVO_INDEX(_timer,_channel)])           // macro to access servo c
46
47  #define SERVO_MIN() (MIN_PULSE_WIDTH - this->min * 4)  // minimum value in uS for this servo
48  #define SERVO_MAX() (MAX_PULSE_WIDTH - this->max * 4)  // maximum value in uS for this servo
49
50  /************ static functions common to all instances ***********************/
51
52  static inline void handle_interrupts(timer16_Sequence_t timer, volatile uint16_t *TCNTn, volatile uint16_t*
```

Using a library - 4. Small changes

1. First test if the example works
2. Make small change
3. Confirm that the changed code works
4. If not go back
5. Make another small change
6. And so on...

Using a library - 5. Final code

1. Made from the example to avoid rewriting and forgetting about important setup steps

2. Done through a series of small changes to the example

Using external libraries

1. Most libraries will already be in an Arduino acceptable form
2. Extract library package and place it in own folder in $HOME/Documents/Arduino/libraries
3. Restart Arduino Software
4. Confirm that the new library shows in the examples drop-down menu

Good sources of external libraries

1. Adafruit (components vendor)

2. Mike McCauley (independent programmer)

3. Anything to which more than three people contributed

Compilation Pipeline

1. Pipeline - programming word-concept to describe a series of steps in fixed order and always necessary
2. Compilation is translating human (or at least programmer) readable code into machine code (consisting of a set of machine commands)
3. Compilation occurs in several steps
4. Steps are explicit in C, C++ and hidden in Python, Node

Compilation Pipeline

1. Preprocess -> Compile -> Link
2. Preprocess - all macros (#define, #include); output: human readable program without any directives (all code “pasted in”)
3. Compile - convert our written code into set of commands: output binary file of our code
4. Link - add external binary files of code we’re using (libraries); output: full executable

Makefile

1. Useful way of doing many compilation step in one shortcut
2. Create file “Makefile”
3. Add variables
4. Add directives (packages of commands)
5. Save and execute by typing “make” when in the folder
6. See tutorials online

Makefile - Example

```
CC=g++

CFLAGS= -g -Wall

TARGET=main

#-----------------a comment--------------------------------

all:

	$(CC) $(CFLAGS) -o $(TARGET) $(TARGET).c #first character must be TAB
```

Compiling Arduino Libraries

1. Absolutely possible to use Arduino libraries without Arduino IDE since Libraries are just code
2. Steps:
    a. Extract Arduino Libraries source code
    b. Eavesdrop on Arduino compilation process
    c. Make a Makefile compiling Arduino Libraries with

Extract Arduino Libraries

1. Located on github or own computer when Arduino IDE installed
2. Code located in “hardware” folder
3. Most code located in “cores” folder
4. Some libraries (SPI) located in own folders

Eavesdrop on Arduino compilation process

1. Open Arduino IDE with any example
2. In Setting/Preferences check the box “Show verbose output during: compilation”
3. Close preferences and click **the tick sign** on any Arduino examples
4. Drag the log to cover most of the screen
5. Read through the compilation log

Make a Makefile compiling Arduino Libraries

1. Place extracted code in one location
2. Compile all source files mimicking Arduino compilation (use correct flags which can affect e.g. executable size)
3. Archive all binary files into one “arduino” archive (archiving is just joining binary files for easier linking)
4. Compile own code
5. Link own code with the “arduino” archive