Tutorial

CIP Statemachine - Arduino

Tutorial	Actifsource Tutorial – CIP Statemachine - Arduino
Required Time	• 120 Minutes
Prerequisites	• Actifsource Tutorial – Installing Actifsource • Actifsource Tutorial – Simple Service • Actifsource Tutorial – CIP Statemachine - Lamp
Goal	• Compiling and running the CIP Statemachine on the Arduino UNO
Topics covered	• Arduino SDK • AVR Eclipse Plugin • Simple Arduino project • CIP Arduino project
Notation	✎ To do ⓘ Information • **Bold**: Terms from actifsource or other technologies and tools • **Bold underlined**: actifsource Resources • Underlined: User Resources • *UnderlinedItalics*: Resource Functions • `Monospaced`: User input • *Italics*: Important terms in current situation
Disclaimer	The authors do not accept any liability arising out of the application or use of any information or equipment described herein. The information contained within this document is by its very nature incomplete. Therefore the authors accept no responsibility for the precise accuracy of the documentation contained herein. It should be used rather as a guide and starting point.
Contact	**actifsource GmbH** Täfernstrasse 37 5405 Baden-Dättwil Switzerland www.actifsource.com
Trademark	**actifsource** is a registered trademark of **actifsource GmbH** in Switzerland, the EU, USA, and China. Other names appearing on the site may be trademarks of their respective owners.
Credits	Francesco Rigoni: Arduino development with Eclipse – A step by step tutorial to the basic setup

- Arduino SDK
  - The Arduino SDK contains all the necessary source code for your Arduino
  - Installing the com port driver for your Arduino
- AVR Eclipse Plugin
  - The AVR Plugin is an extension to the C/C++ Development Toolkit to support development for the Atmel AVR series of embedded processors.
  - Configure the AVR Plugin for your Arduino board
- Simple Arduino project
  - Setup a new C/C++ project
  - Write a very simple code which switches an LED on and off
  - Compile the project including the Arduino core
  - Download to the Arduino target and run
- CIP Arduino project
  - Modeling a reactive state machine using the CIP method
  - Generating C code for the state machine
  - Connecting the generated state machine code to the Arduino I/O
  - Downloading and testing to the Arduino board

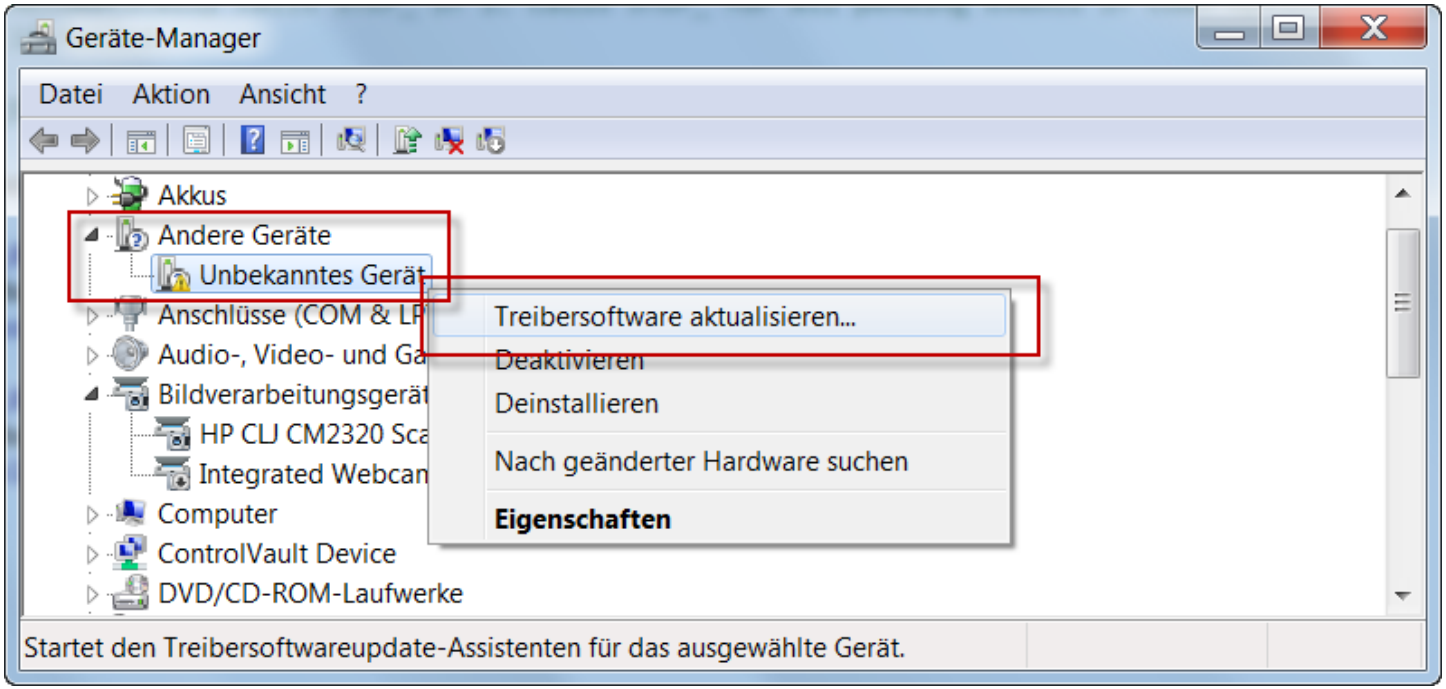
Part 0: Installing Eclipse

- Install the Eclipse IDE for C/C++ Developers (C/C++ Development Tool)
  - http://www.eclipse.org/downloads/
- Install the Actifsource Enterprise plugin

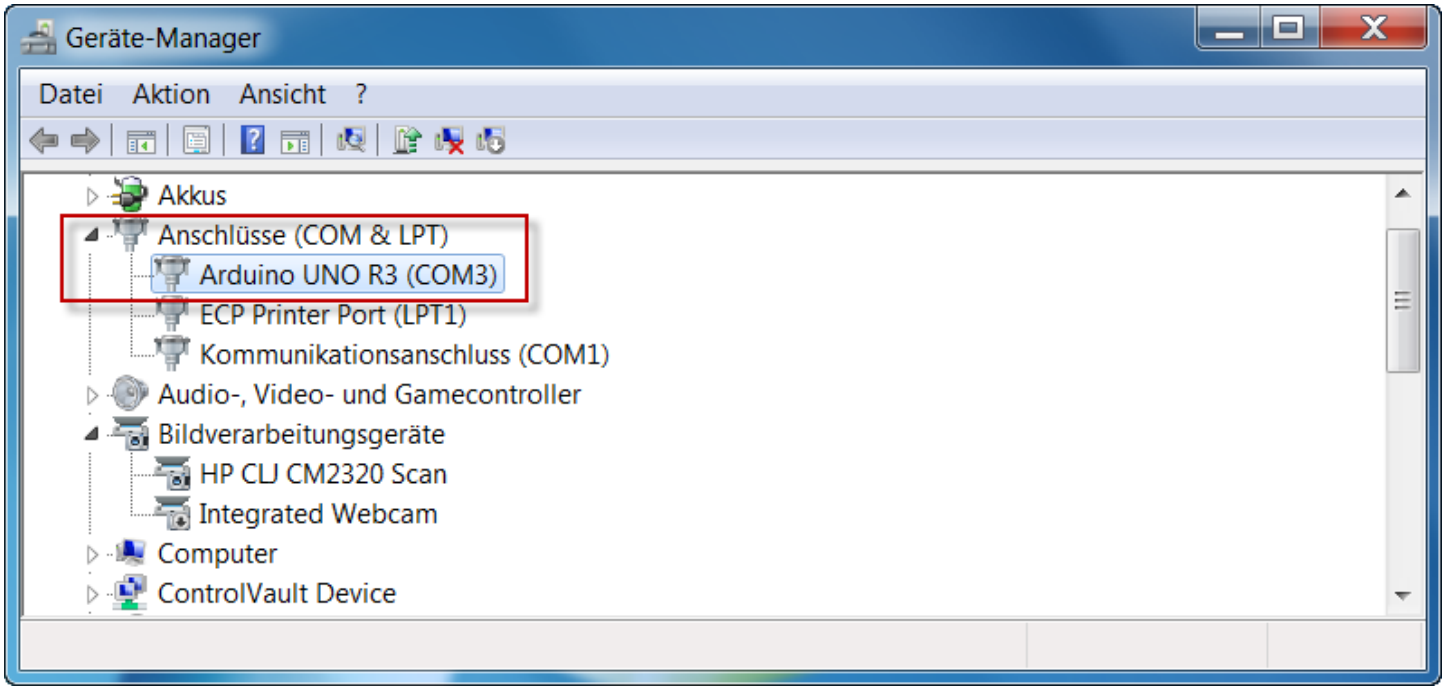
Part 1: Arduino SDK

- The Arduino SDK contains all the necessary source code for your Arduino
- Installing the com port driver for your Arduino

- Download your Arduino SDK from www.arduino.cc
- Unzip the Arduino SDK on your hard disk
  - We use `C:\arduino-1.0.3\` in this example



- Plugin your Arduino to a USB port
  - The installation will fail
- Open the Device Manager from the Control Panel
- Find the failed device
- Install new driver
- Search the driver on your computer
- Specify `C:\arduino-1.0.3\drivers` as the place to look for the driver
  - Warning: You must not select `C:\arduino-1.0.3\drivers\FTDI USB Drivers`



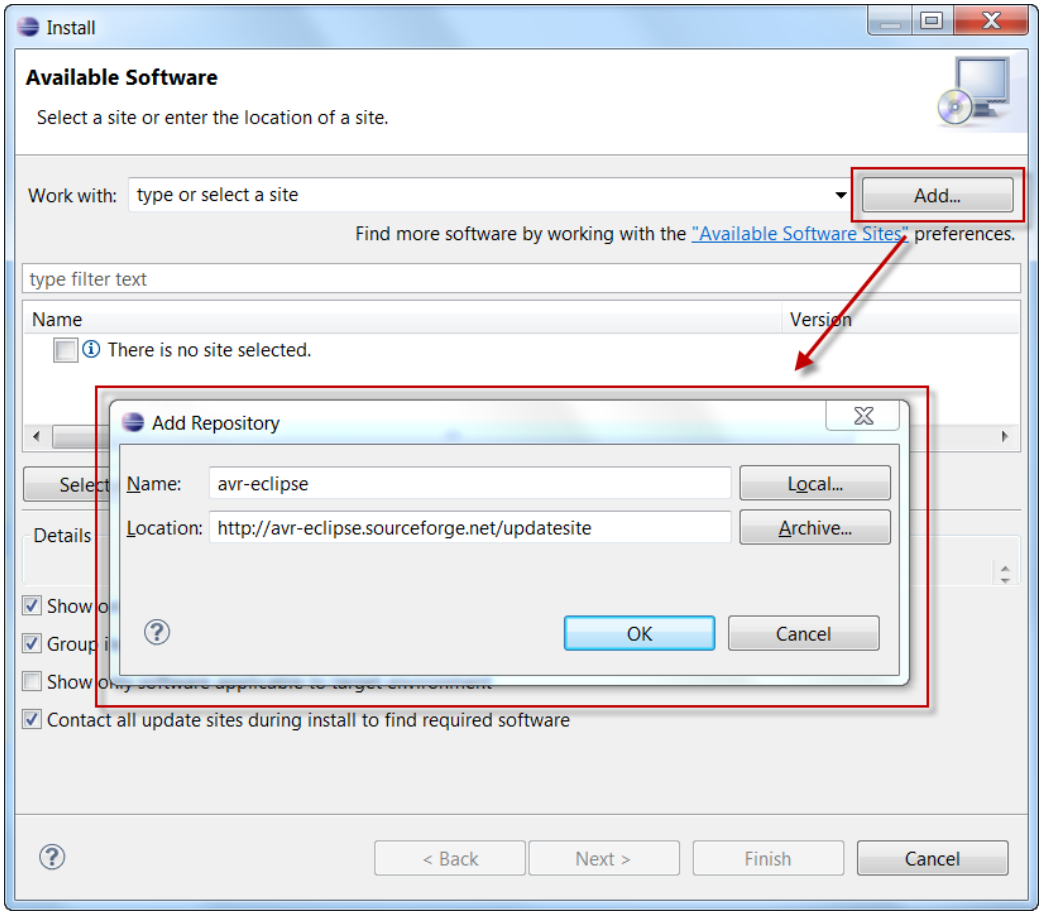
- Check if the driver has been installed correctly
- Important: Remember the com port
  - `COM3` in this example

```
uno.name=Arduino Uno
uno.upload.protocol=arduino
uno.upload.maximum_size=32256
uno.upload.speed=115200
uno.bootloader.low_fuses=0xff
uno.bootloader.high_fuses=0xde
uno.bootloader.extended_fuses=0x05
uno.bootloader.path=optiboot
uno.bootloader.file=optiboot_atmega328.hex
uno.bootloader.unlock_bits=0x3F
uno.bootloader.lock_bits=0x0F
uno.build.mcu=atmega328p
uno.build.f_cpu=16000000L
uno.build.core=arduino
uno.build.variant=standard
```

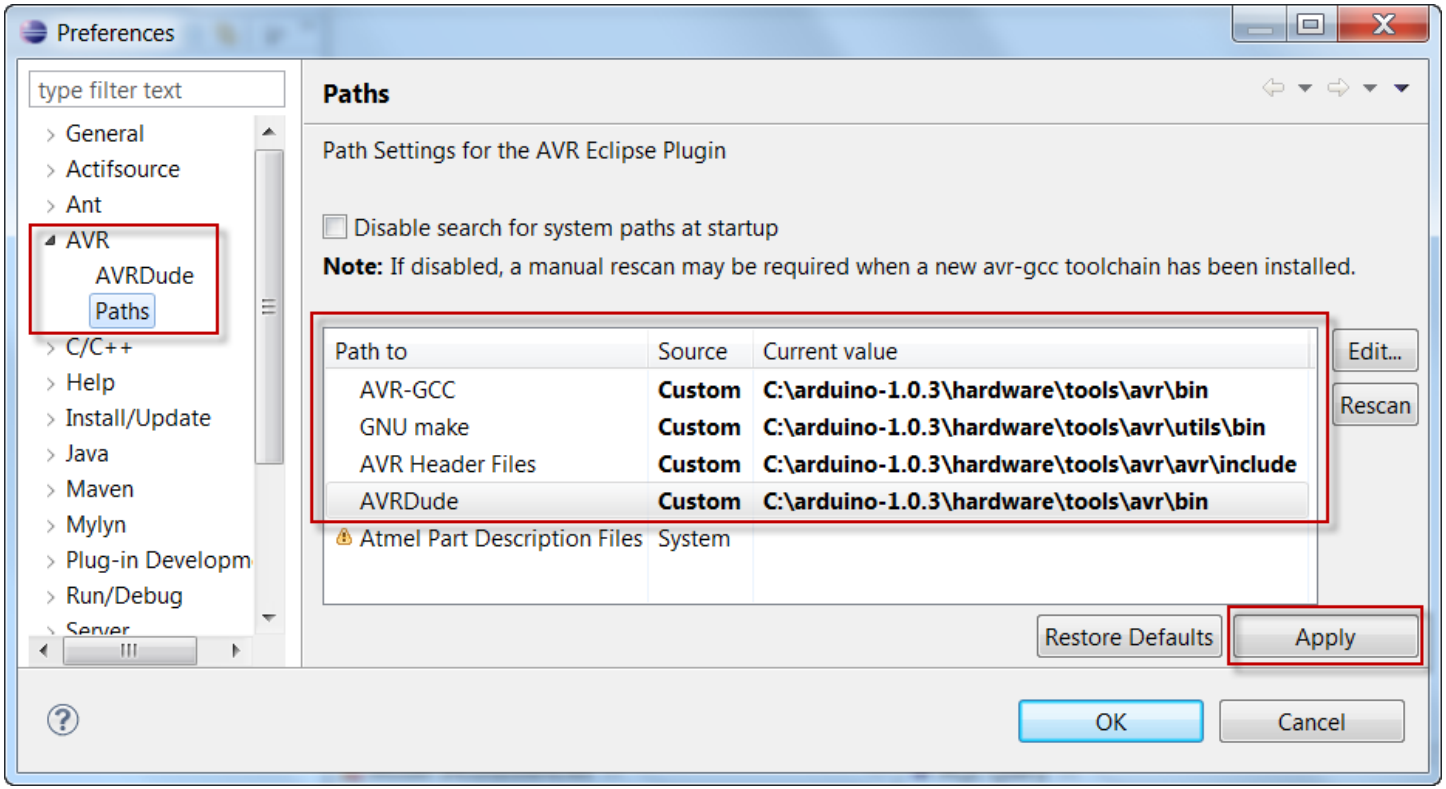
- Open the `boards.txt` file in `C:\arduino-1.0.3\hardware\arduino`
- Print the information for your hardware platform
  - `Arduino Uno` in this example

Part 2: AVR Eclipse Plugin

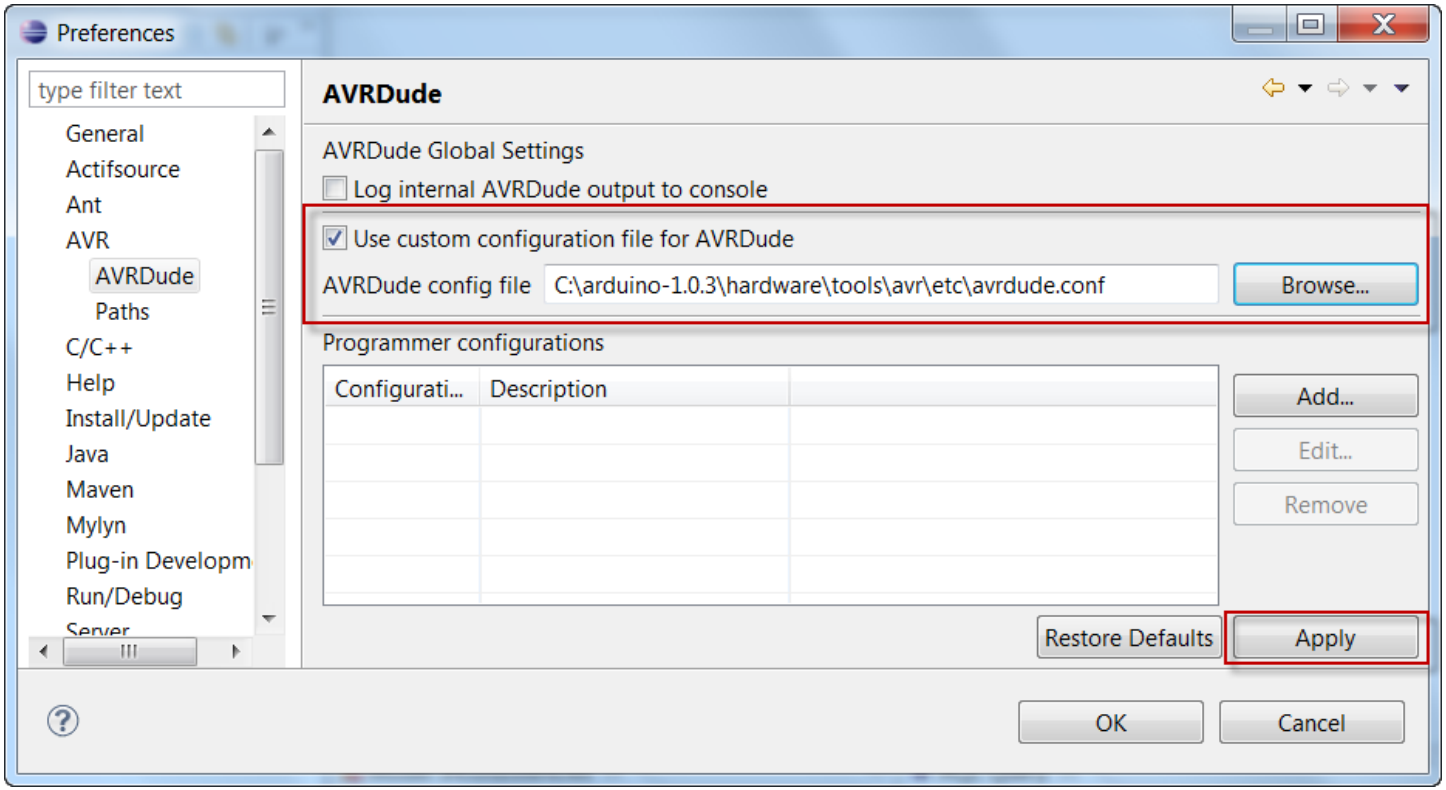
- The AVR Plugin for Eclipse is an extension to the C/C++ Development Toolkit to support development for the Atmel AVR series of embedded processors.
- Configure the AVR Plugin for your Arduino board



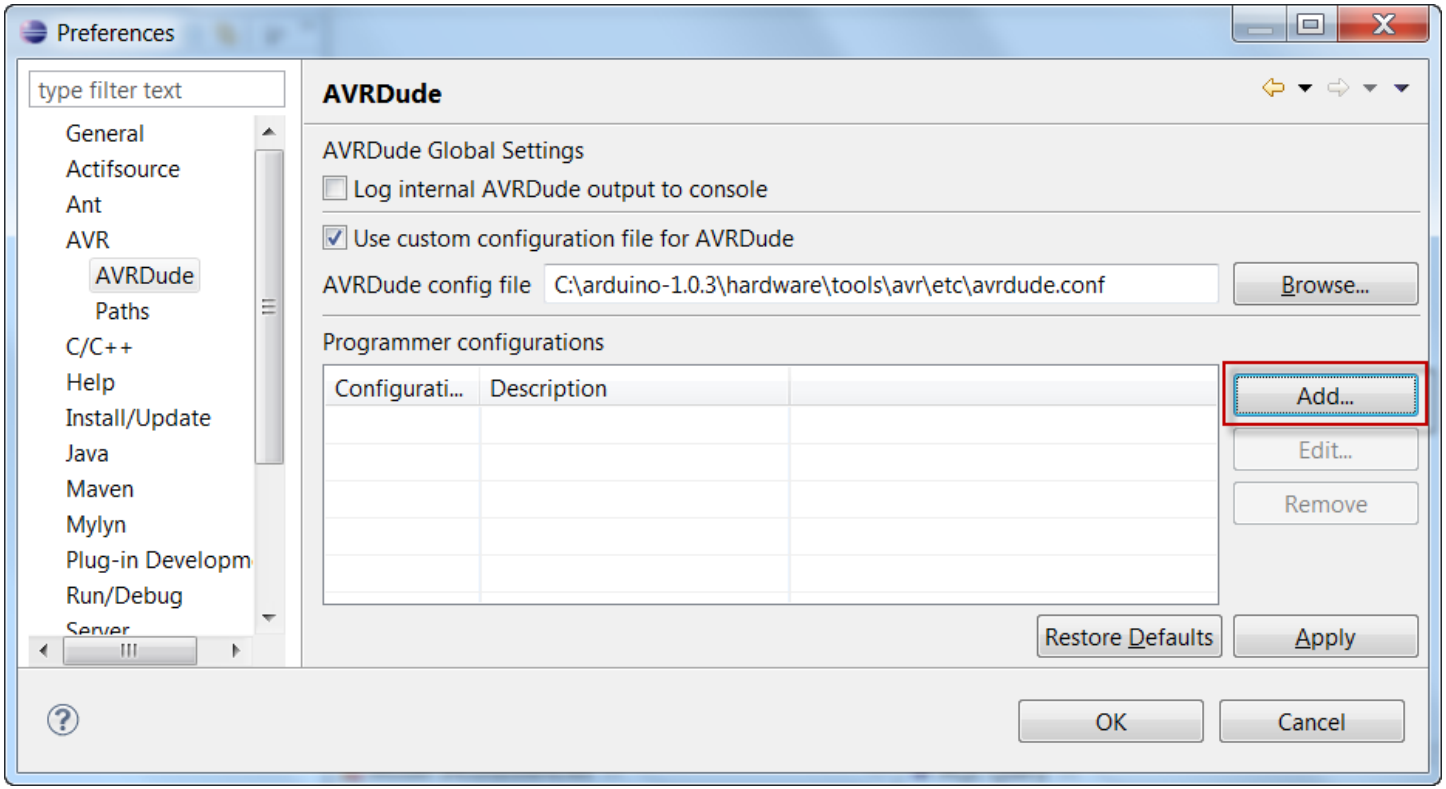
- Start your Eclipse with a new workspace
  - `workspace_arduino` in this example
- In your Eclipse, click Help/Install new Software…
- Add the AVR-Eclipse plugin update site
  - `http://avr-eclipse.sourceforge.net/updatesite`
- ⓘ Installing the plugin requires eclipse to restart



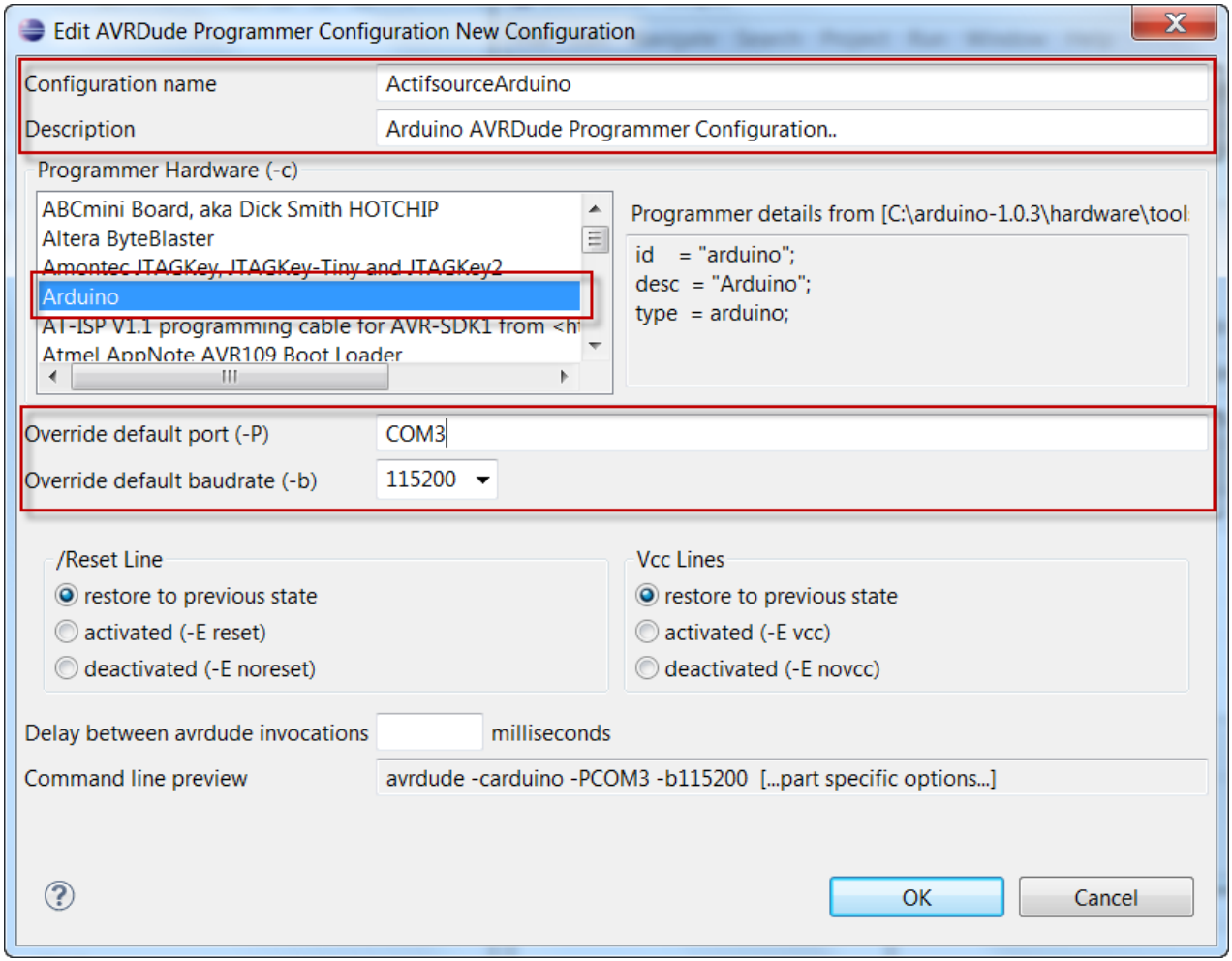
- Configure the AVR Eclipse Plugin under Windows/Preferences/AVR/Paths
- Set `AVR-GCC` to custom value `C:\arduino-1.0.3\hardware\tools\avr\bin`
- Set `GNU make` to custom value `C:\arduino-1.0.3\hardware\tools\avr\utils\bin`
- Set `SVR-GCC` to custom value `C:\arduino-1.0.3\hardware\tools\avr\avr\include`
- Set `SVR-GCC` to custom value `C:\arduino-1.0.3\hardware\tools\avr\bin`
- Important: Press **Apply**



- Configure the AVR Eclipse Plugin under Windows/Preferences/AVR/AVRDude
- Use AVRDude config file `C:\arduino-1.0.3\hardware\tools\avr\etc\avrdude.conf`
- Important: Press **Apply**



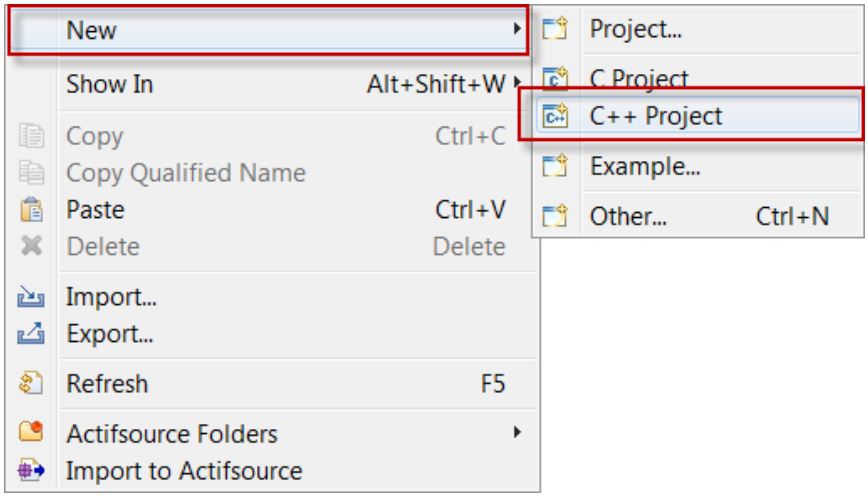
- Configure the AVR Eclipse Plugin under Windows/Preferences/AVR/AVRDude
- Add a new Programmer configuration
- If this is not working you probably forget to press **Apply** when configuring the AVR/Paths



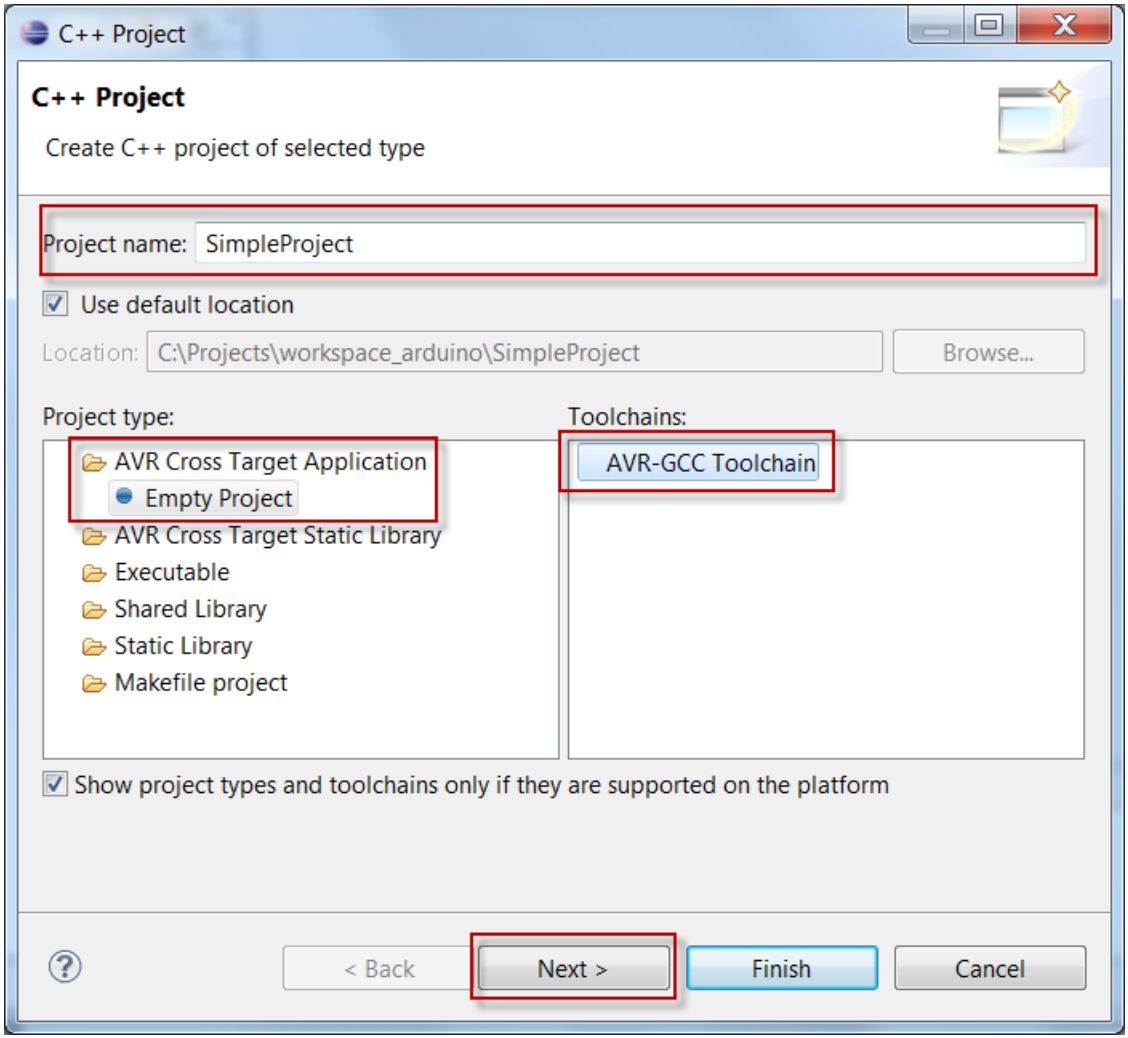
- Select the Programmer Hardware as found in the boards.txt file in `uno.upload.protocol`
  - `Arduino` In this example
- Select the com port that you remembered when installing the driver in the Device Manager
  - `COM3` In this example
- Select the baud rate as found in the boards.txt file in `uno.upload.speed`
  - `115200` In this example

Part 3: Simple Arduino project

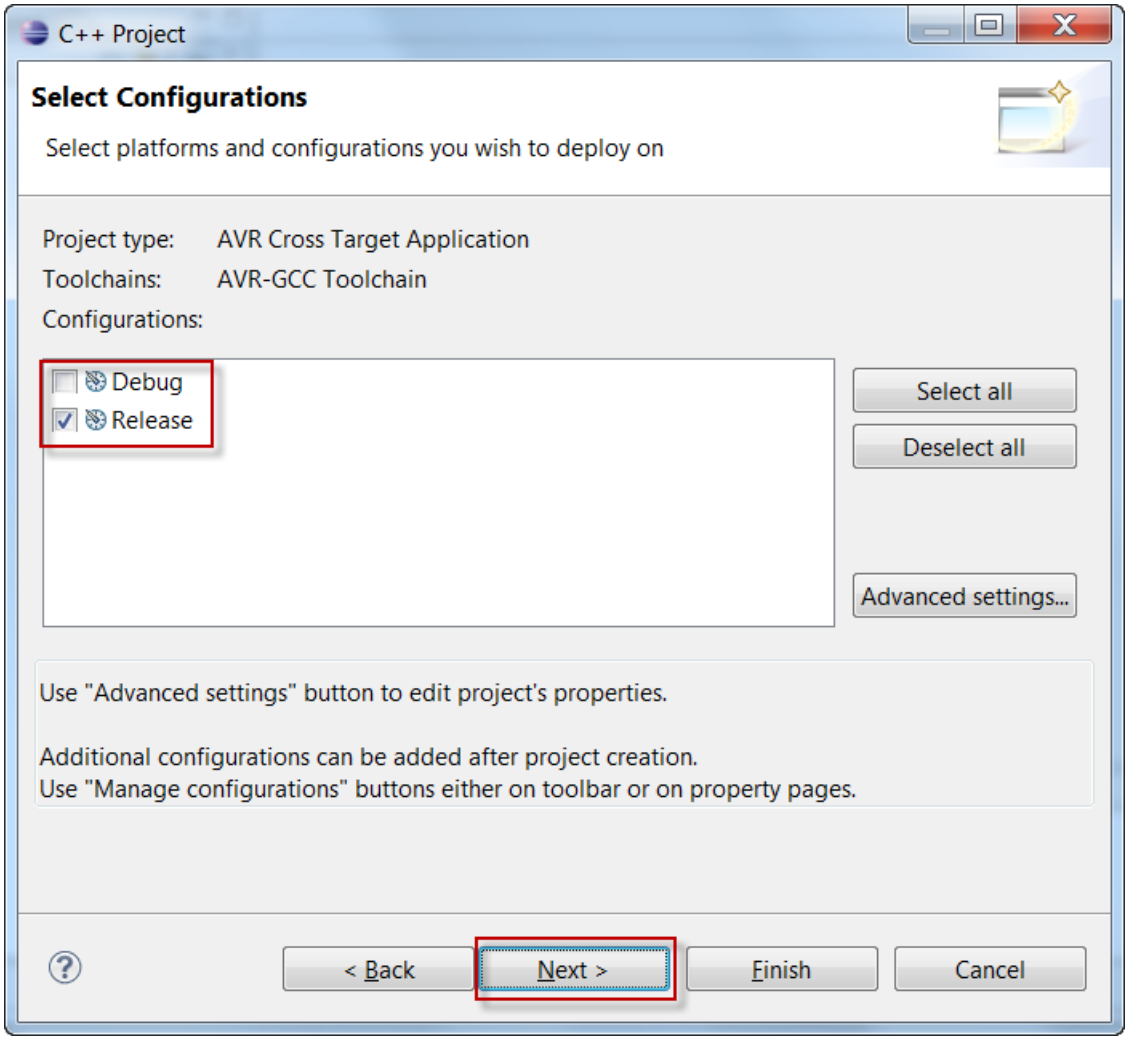
- Setup a new C/C++ project
- Write a very simple code which switches an LED on and off
- Compile the project including the Arduino core
- Download to the Arduino target and run



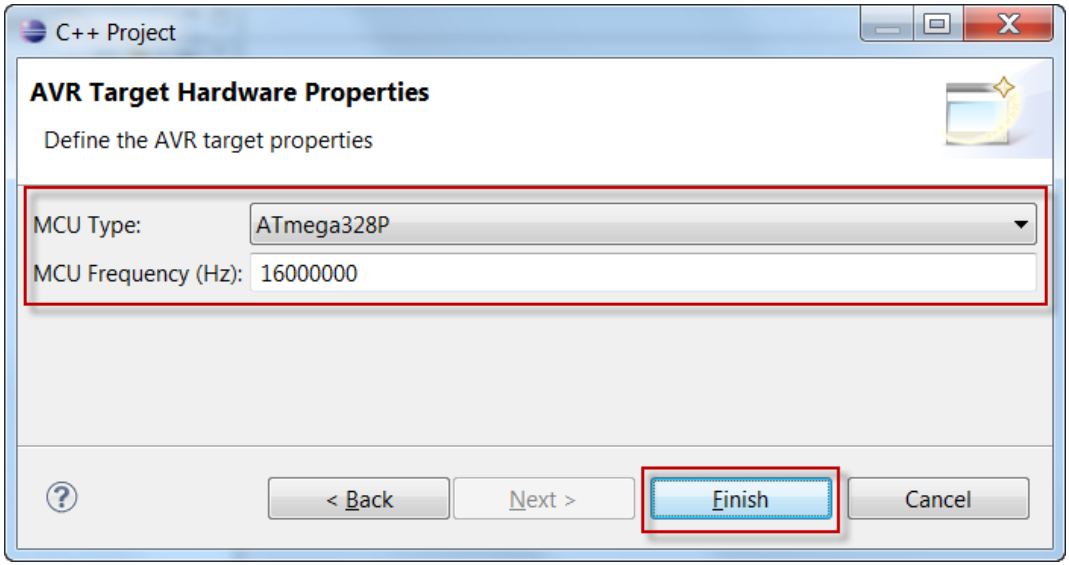
- Create a new C++ Project
- ⓘ You have to create a C++ project even if you plan to write C code since we have to compile the Arduino Core
- ⓘ If you can't create a C/C++ Project you have probably not downloaded the Eclipse for C/C++ Developers
  - Install the CDT plugin from http://www.eclipse.org/cdt/



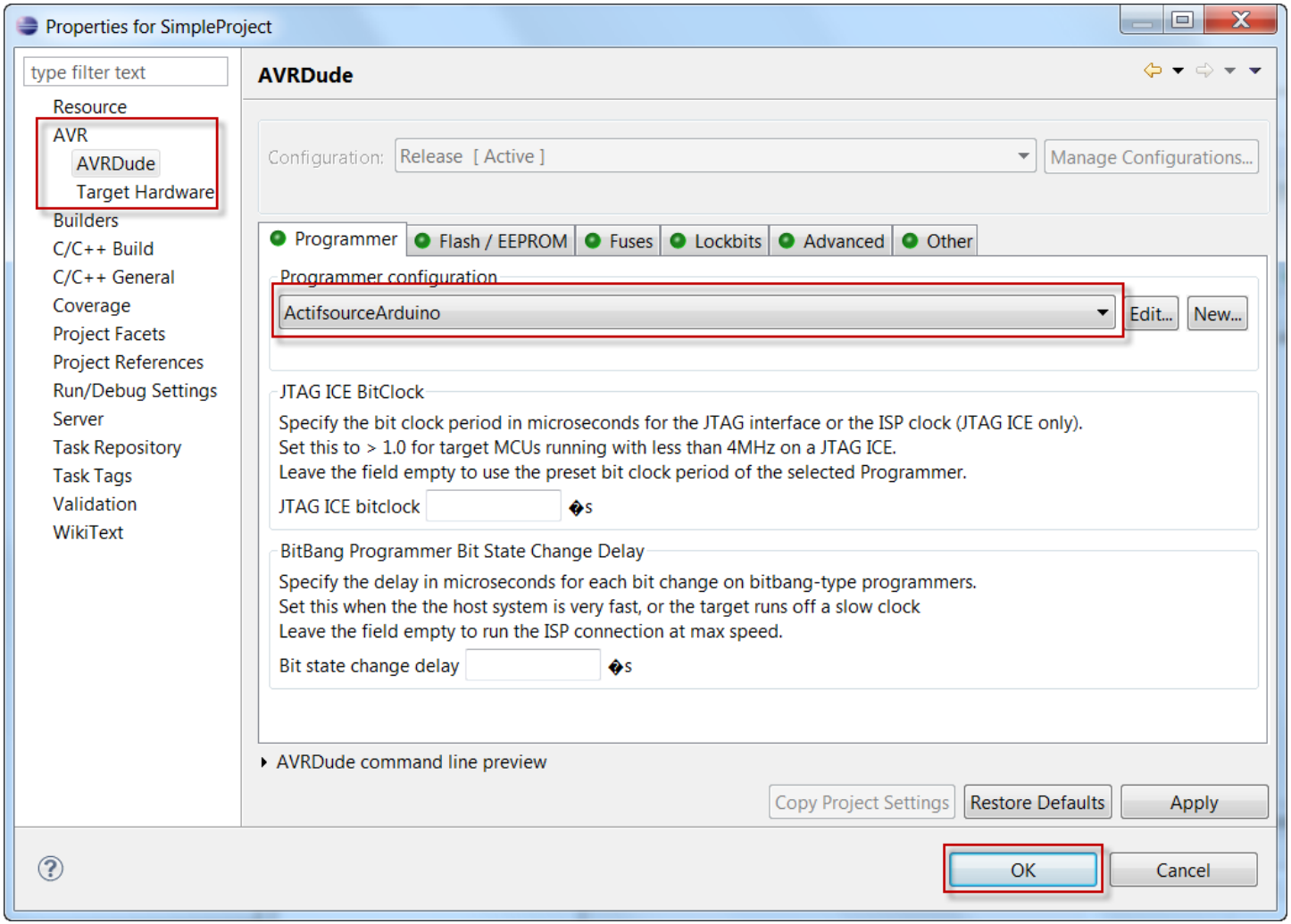
- Create a new C++ project named `SimpleProject`
- Choose AVR Cross Target Application
- Choose the AVR-GCC Toolchain
- Click **Next**



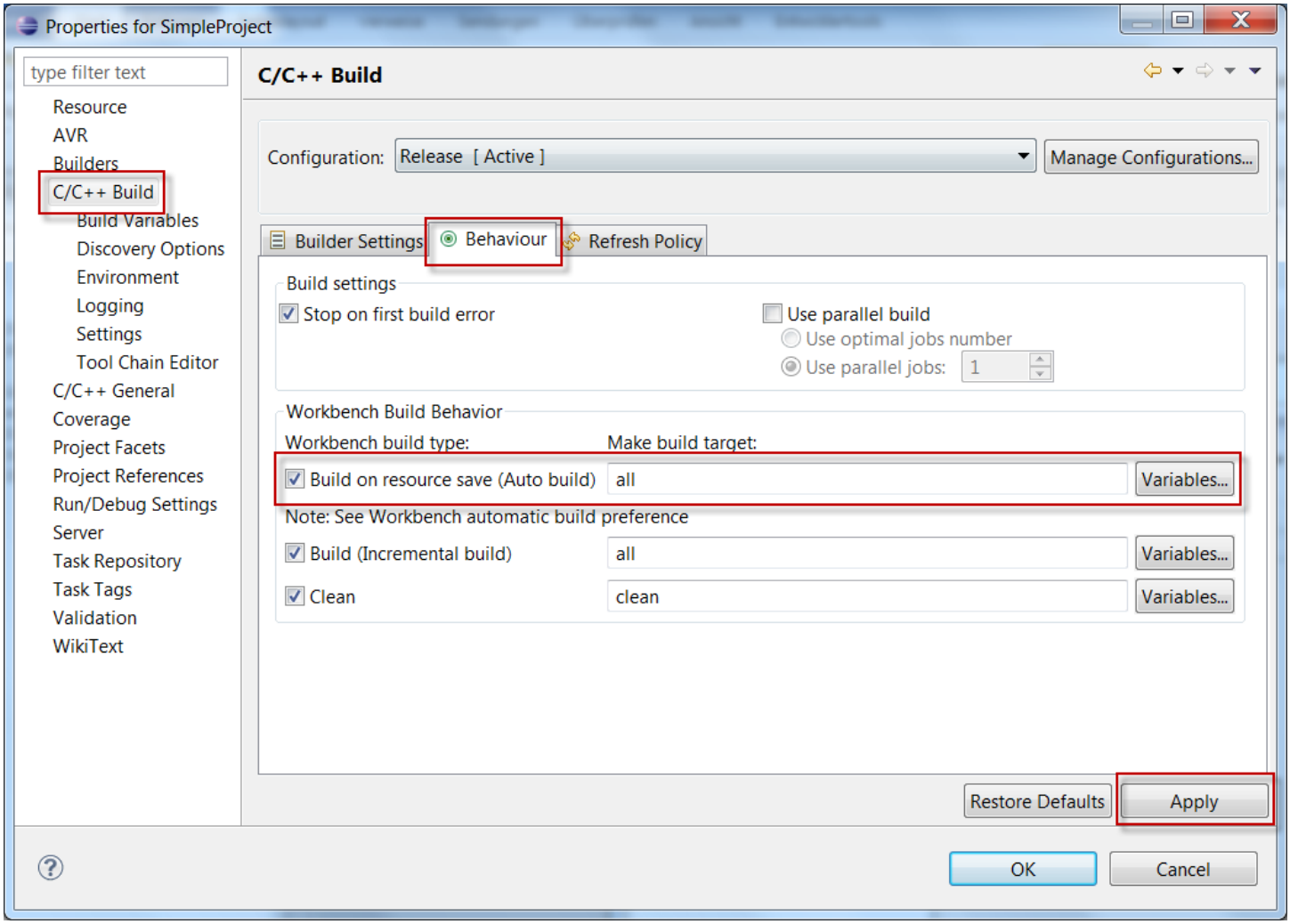
- Do not create a Debug Version
- ⓘ The debug code is too large to fit on the Arduino Uno
- ⓘ See boards.txt for the maximum code size
  - `uno.upload.maximum_size=32256` for this example
- Click **Next**



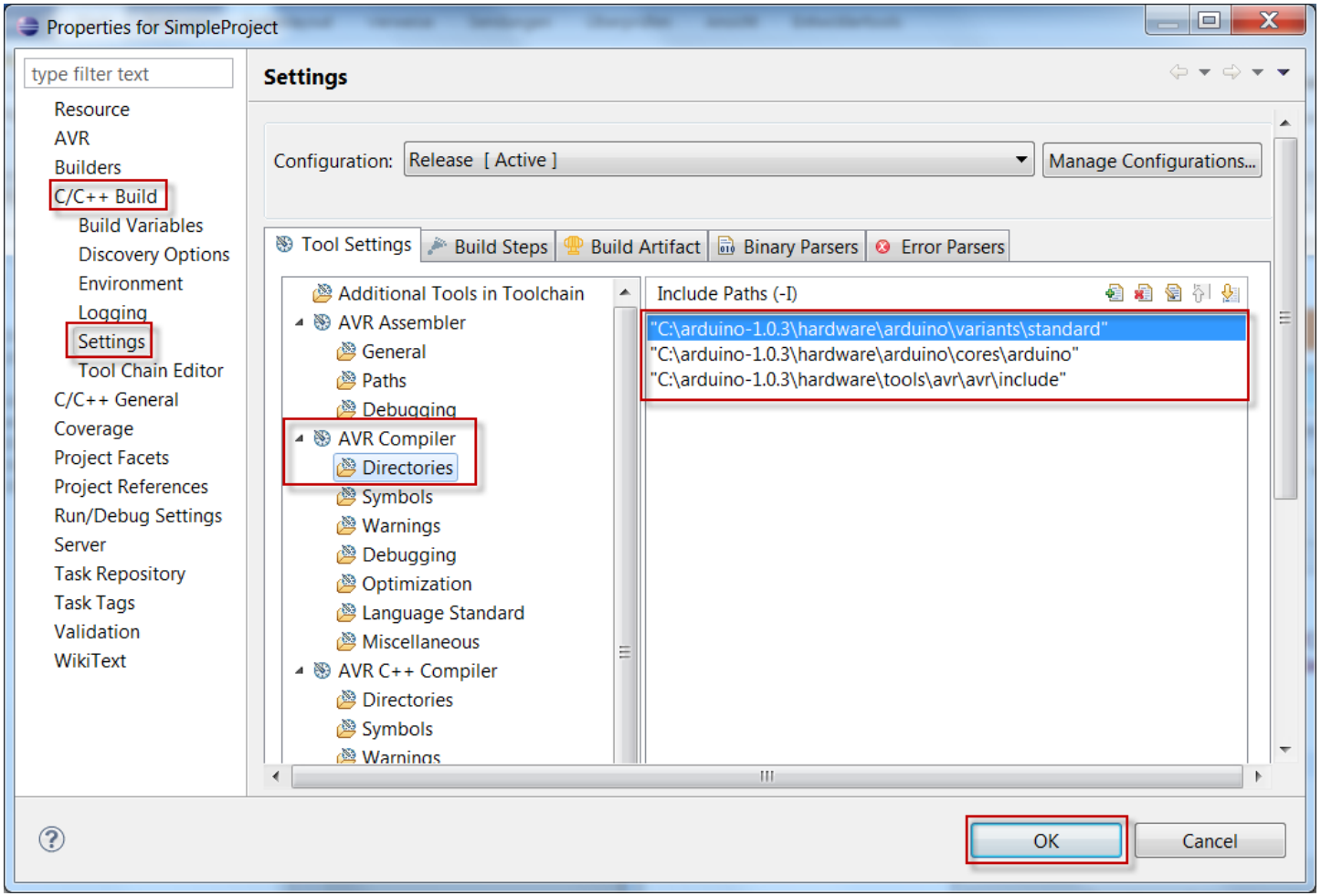
- Specify the MCU Type as found in your boards.txt (`uno.build.mcu`)
  - `atmega328p` for this example
- Specify the MCU Frequency as found in your boards.txt (`uno.build.f_cpu`)
  - `16000000L` for this example
  - Enter the number without the L
- Click **Finish**
- Open the C/C++ Perspective if you are asked or open it manually in the upper right corner 



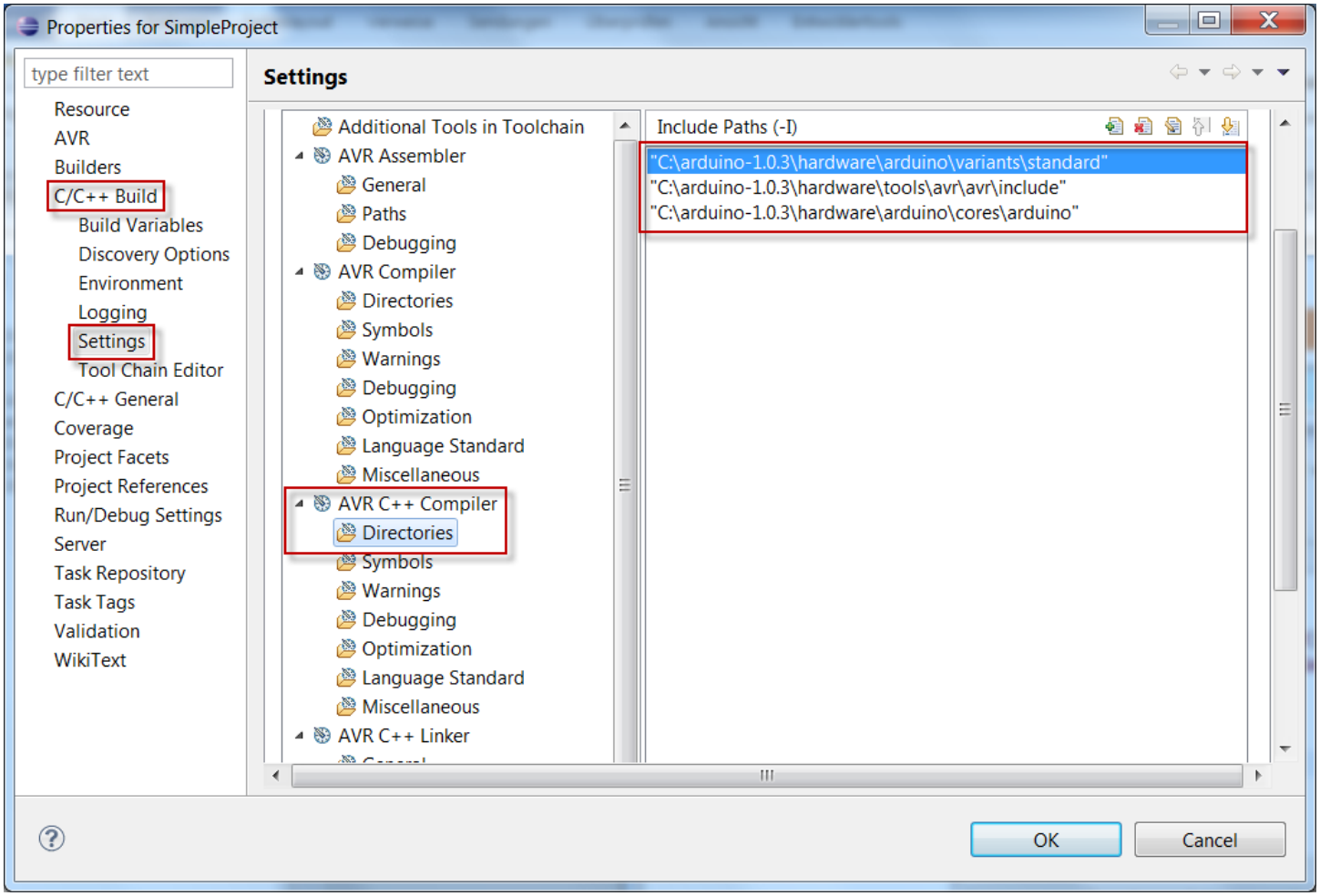
- For the newly created `SimpleProject`, choose Properties (Mouse-Click-Right on the project)
- Select AVR/AVRDude
- In the Programmer Tab, select the previously created Programmer Configuration `ActifsourceArduino`
- Click **OK**



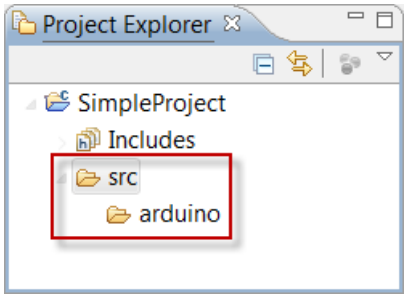
- SimpleProject/Properties/C/C++Build/Behaviour
- Click **Build on resource save (Auto build)** for automatic rebuild
- Click **Apply**



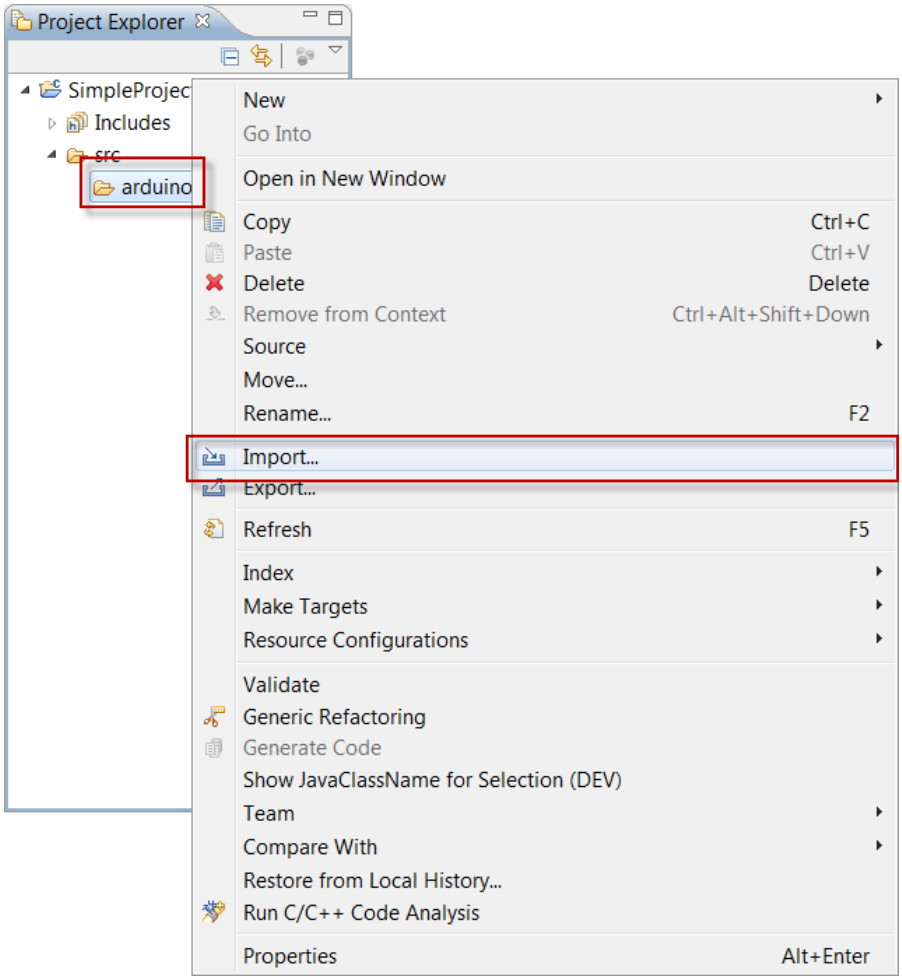
- SimpleProject/Properties/C/C++Build/Settings/AVR Compiler/Directories
- Set Include Path `C:\arduino-1.0.3\hardware\tools\avr\avr\include`
- Set Include Path `C:\arduino-1.0.3\hardware\arduino\cores\arduino`
- Set Include Path `C:\arduino-1.0.3\hardware\arduino\variants\`***`standard`***
  - See boards.txt (`uno.build.variant`) for your variant (standard for this example)
- Click **OK**



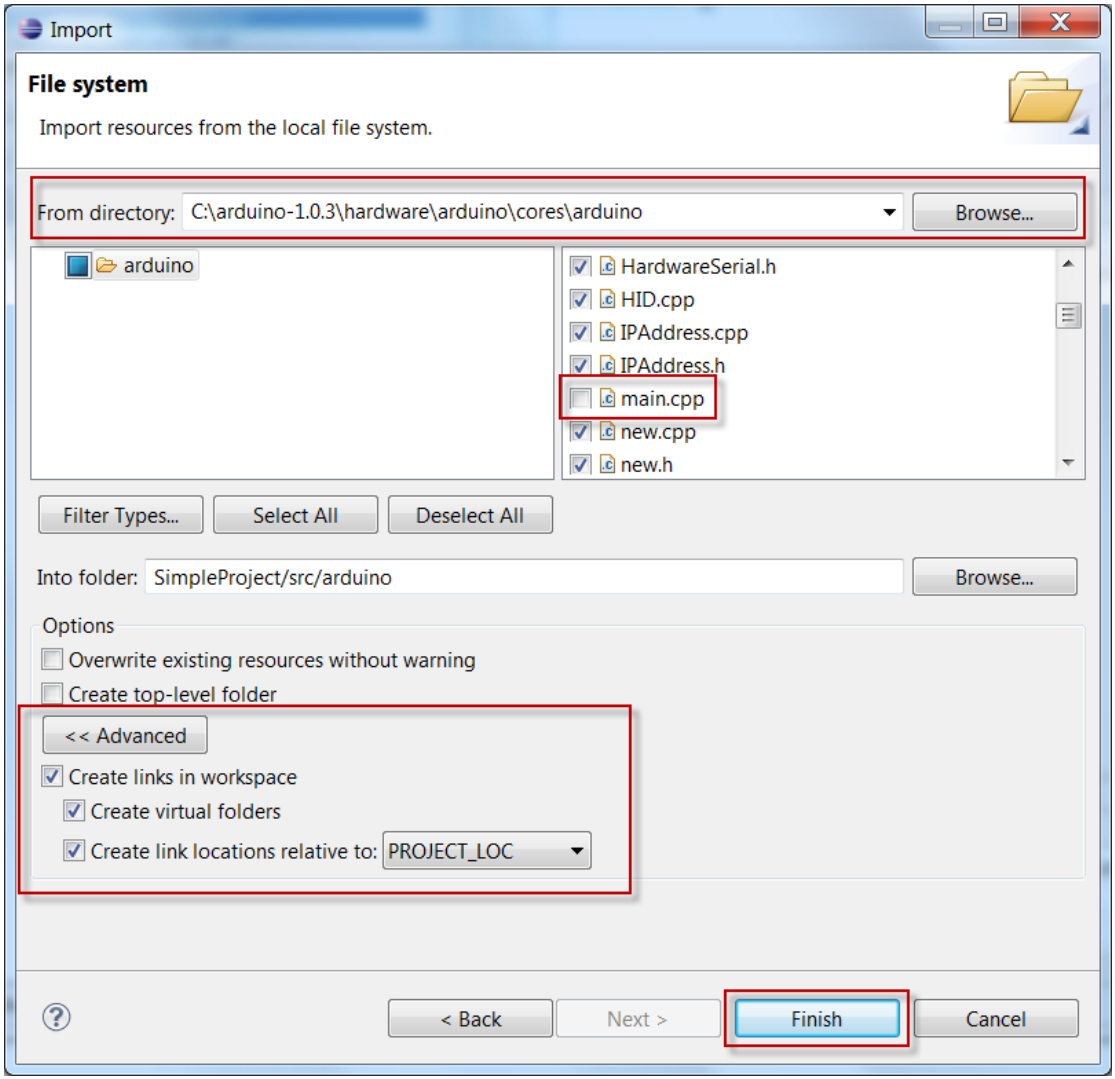
- Do the same again for SimpleProject/Properties/C/C++Build/Settings/AVR C++ Compiler/Directories
- Set Include Path `C:\arduino-1.0.3\hardware\tools\avr\avr\include`
- Set Include Path `C:\arduino-1.0.3\hardware\arduino\cores\arduino`
- Set Include Path `C:\arduino-1.0.3\hardware\arduino\variants\`***`standard`***
  - See boards.txt (`uno.build.variant`) for your variant (standard for this example)
- Click **OK**



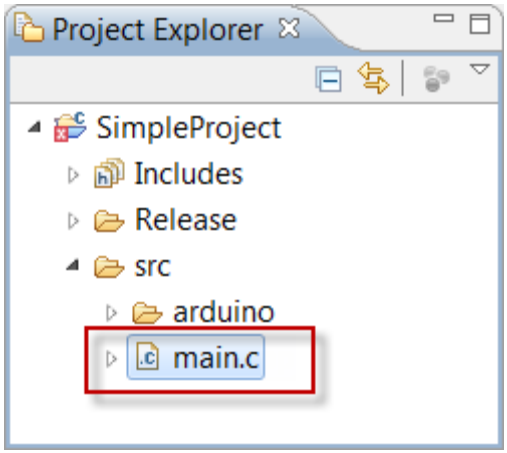
- Create a new folder `src` in the project
- Create a new folder `src/arduino` in the project



- For the folder `src/arduino` choose Import…
- Choose Import/General/File System



- Select directory `C:\arduino-1.0.3\hardware\arduino\cores\arduino`
- Select **All files**
- Important: Deselect `main.cpp`
- Click **Advanced**
- Select **Create links in workspace**
- Click **Finish**



- Create a new file `main.c` in the folder `src`

```
#include <Arduino.h>                // include the arduino header file

int ledPin =  13;                   // LED connected to digital pin 13

int main(void)
{
  init();                           // initialize the arduino hardware
  pinMode(ledPin, OUTPUT);          // initialize the digital pin as an output:

  while(true)                       // loop forever
  {
    digitalWrite(ledPin, HIGH);     // set the LED on
    delay(1000);                    // wait for a second
    digitalWrite(ledPin, LOW);      // set the LED off
    delay(1000);                    // wait for a second
  }
  return 0;
}
```

- Write a very simple Arduino program in the file `main.c` as shown above
- ⓘ If you enabled **Build on resource save (Auto build)** in SimpleProject/Properties/C/C++Build/Behaviour before, the code should now be built
- ⓘ You can also build manually by pressing Project/Build All (Ctrl+B) or the build button in the Eclipse toolbar
- ⓘ Your code is probably not linking if you named your file `main.cpp` instead if `main.c`

C/C++ - Eclipse

File Edit Source Refactor Navigate Search Project Run AVR Window Help

EditorTools

C/C++

actifsource

Project E

SimpleProject
- Binaries
- Includes
- Release
- src
  - arduino
  - main.c

Arduino.h | main.c

```
#include <Arduino.h>

int ledPin =  13;    // LED connected to digital pin 13

// The setup() method runs once, when the sketch starts
void setup()   {
  // initialize the digital pin as an output:
```

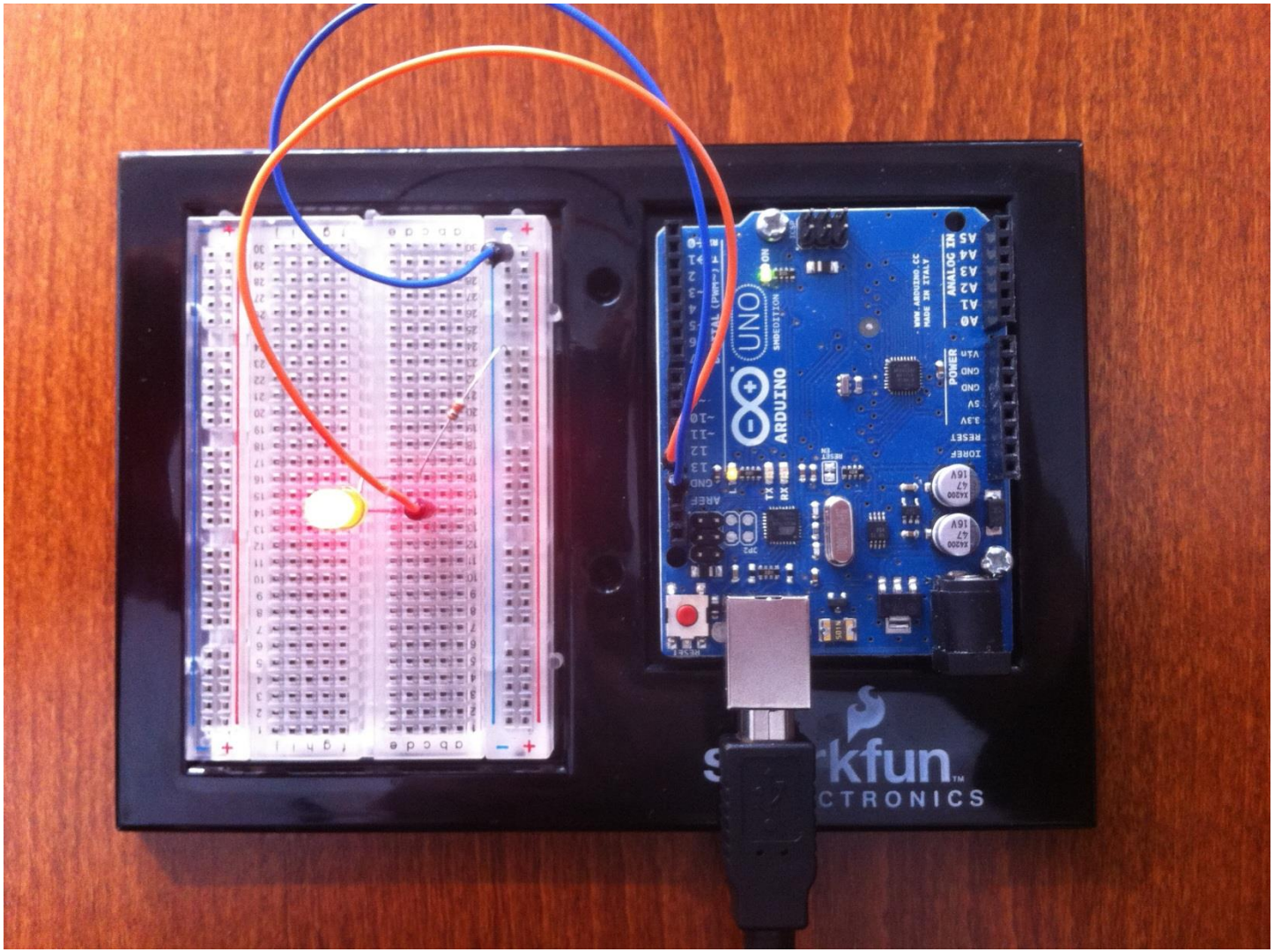
Problems | Tasks | Console | Properties | AVR Device Explorer | AVR Supported MCUs

AVRDude

```
Writing | ################################################## | 100% 3.37s

avrdude: 20970 bytes of flash written
avrdude: verifying flash memory against SimpleProject.hex:
avrdude: load data flash data from input file SimpleProject.hex:
avrdude: input file SimpleProject.hex auto detected as Intel Hex
avrdude: input file SimpleProject.hex contains 20970 bytes
avrdude: reading on-chip flash data:
```

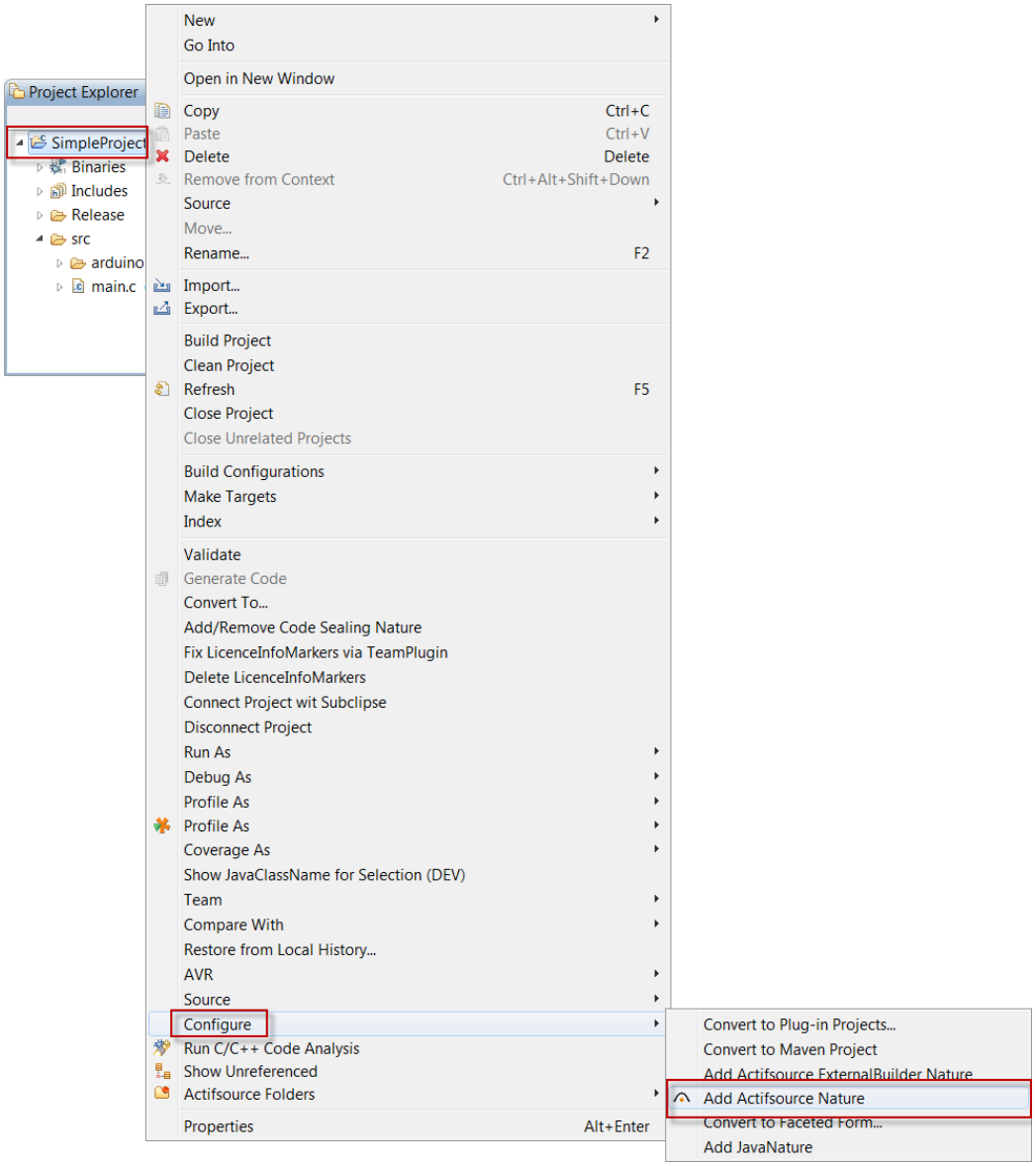
- Download your code to the Arduino
  - Select your project SimpleProject in the Project Explorer
  - Press the AVR download button in the Eclipse toolbar 



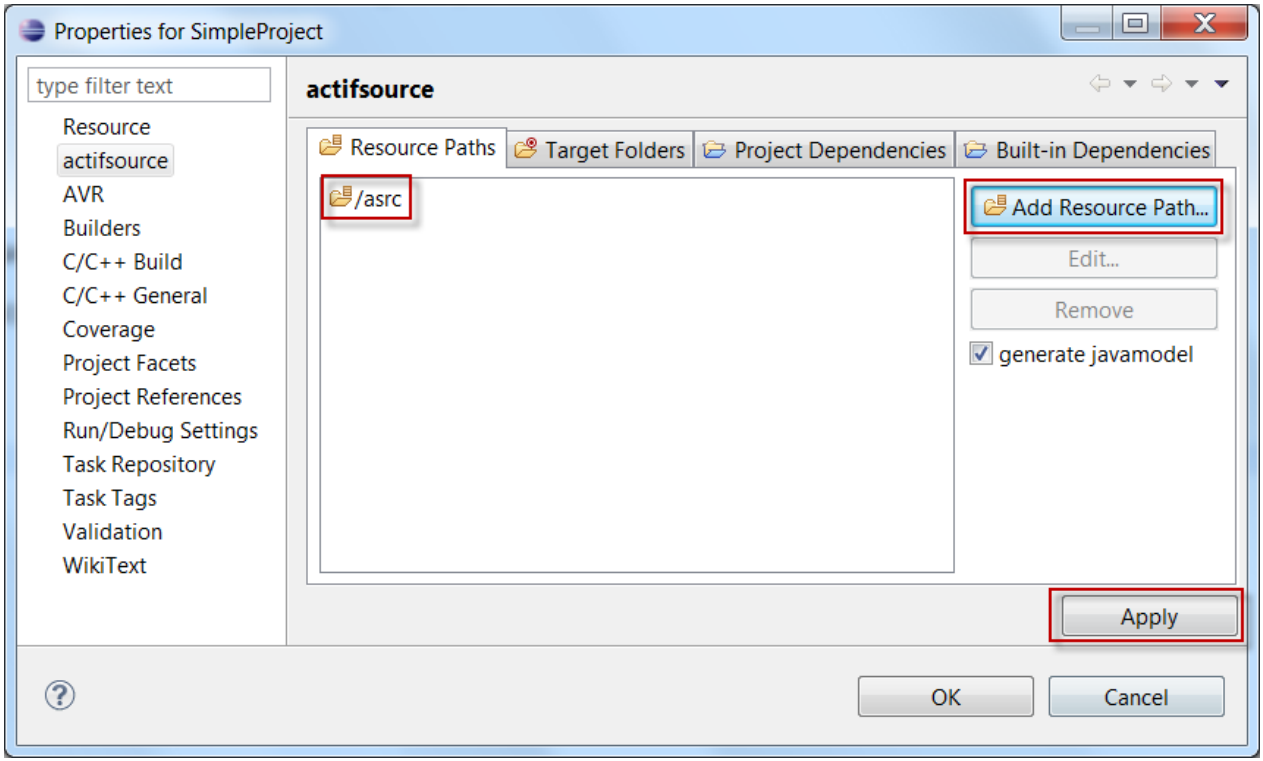
- Mount your LED between Pin 13 and GND
  - Don't forget the 330Ω resistor for a 1.7V LED with 10mA → (5V-1.7V)/0.01A = 330Ω
- ⓘ The program starts automatically after the download has been completed
  - The LED should switch on and off once per second

Part 4: CIP Arduino project

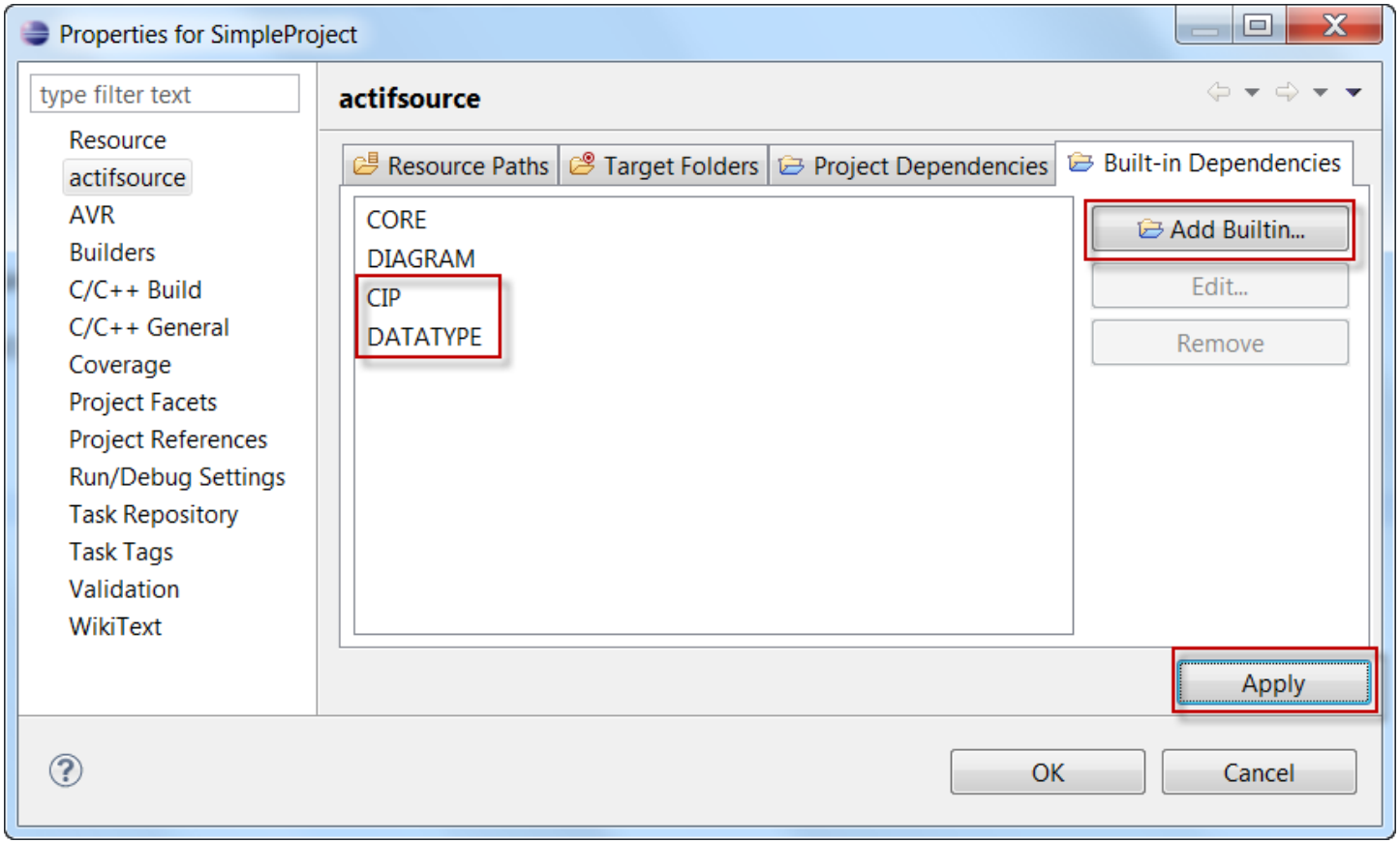
- Modeling a reactive state machine using the CIP method
- Generating C code for the state machine
- Connecting the generated state machine code to the Arduino I/O
- Downloading and testing to the Arduino board



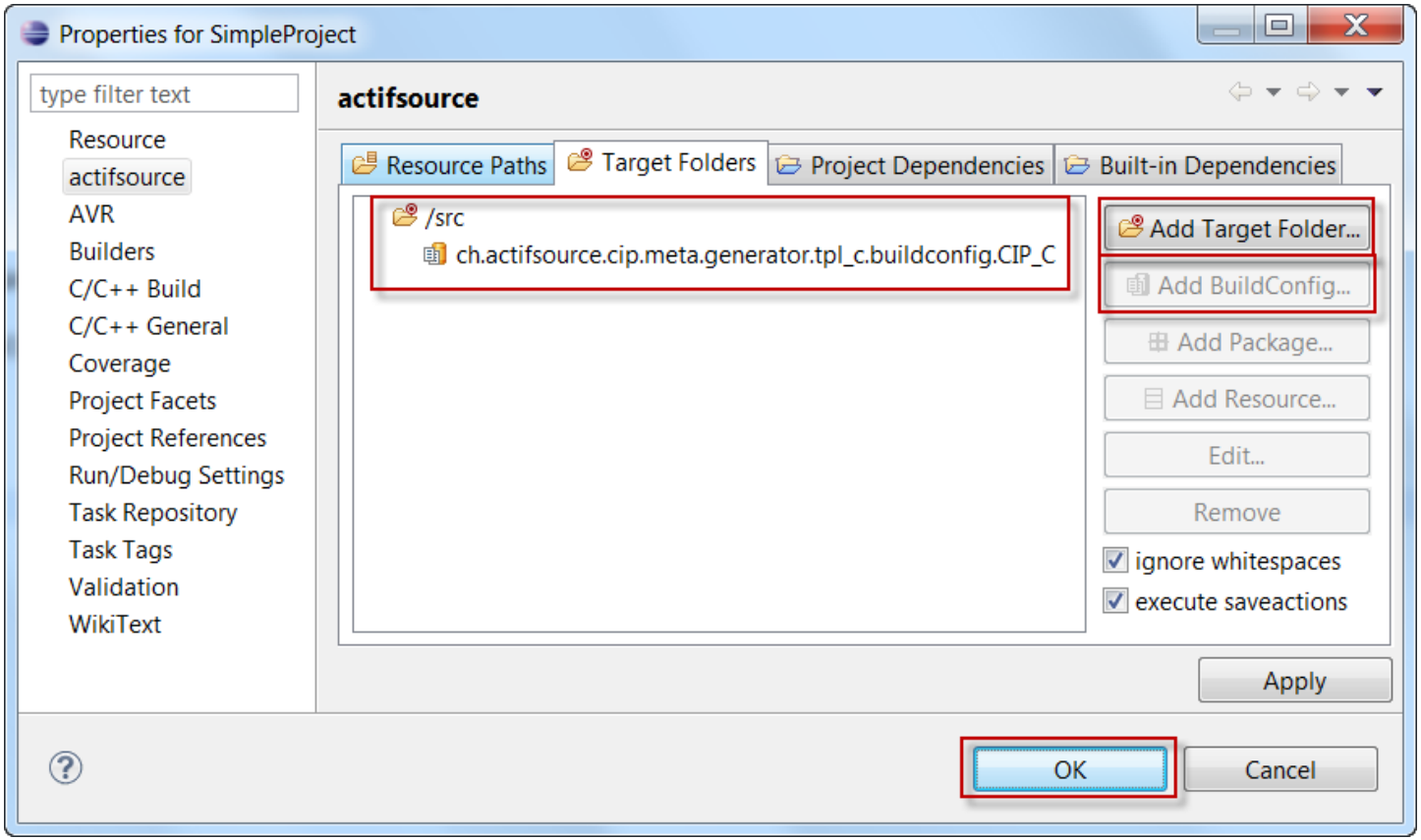
- Switch to the Actifsource Perspective 
- Select your project SimpleProject in the Project Explorer
- Select **Configure/Add Actifsource Nature**



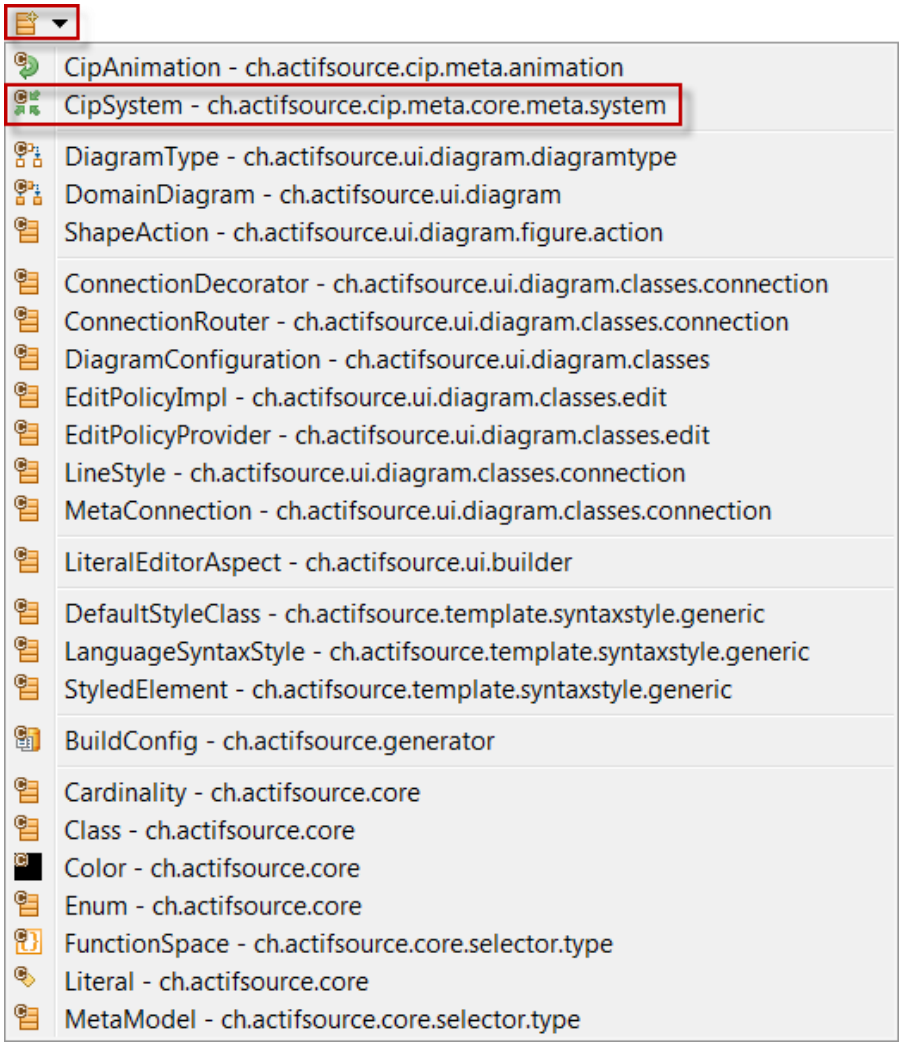
- Open SimpleProject/Properties/actifsource/Resource Paths
- Add the Resource Path `asrc`
- Press **Apply**



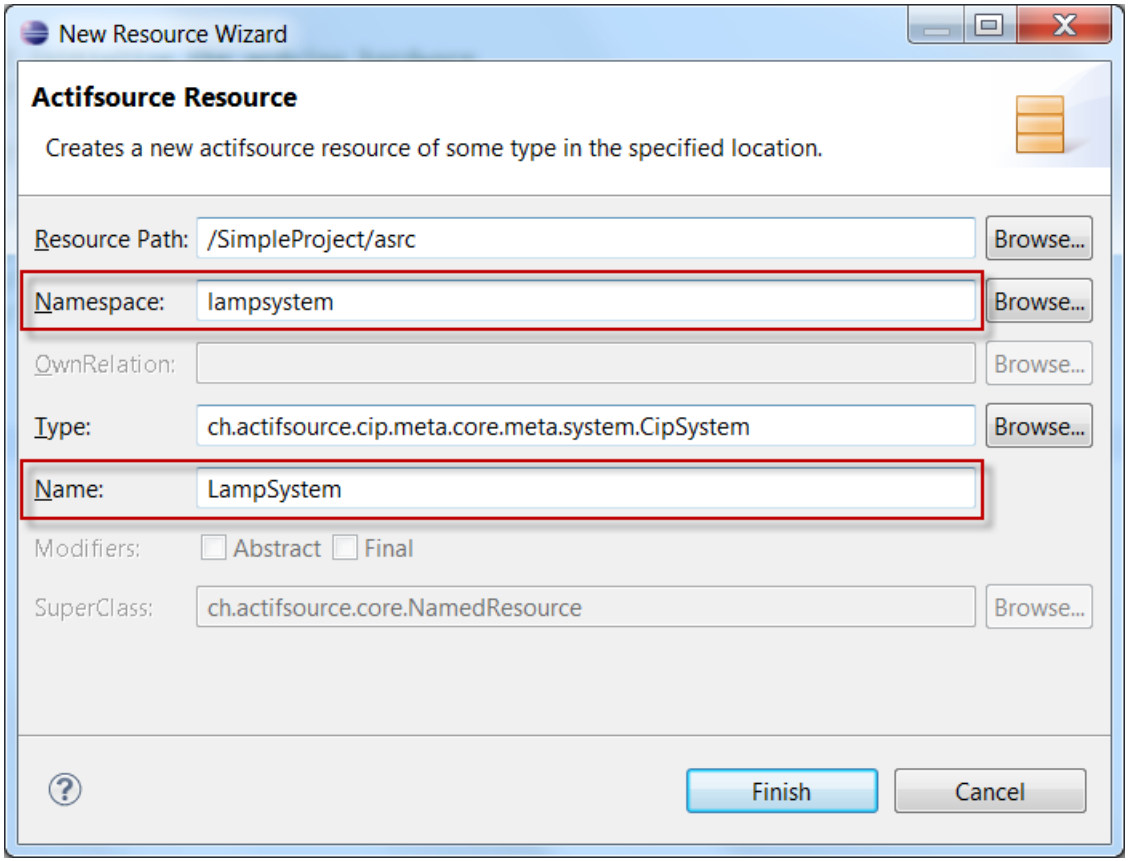
- Open SimpleProject/Properties/actifsource/Built-in Dependencies
- Add Builtin **CIP**
- Add Builtin **DATATYPE**
- Important: Press **Apply**



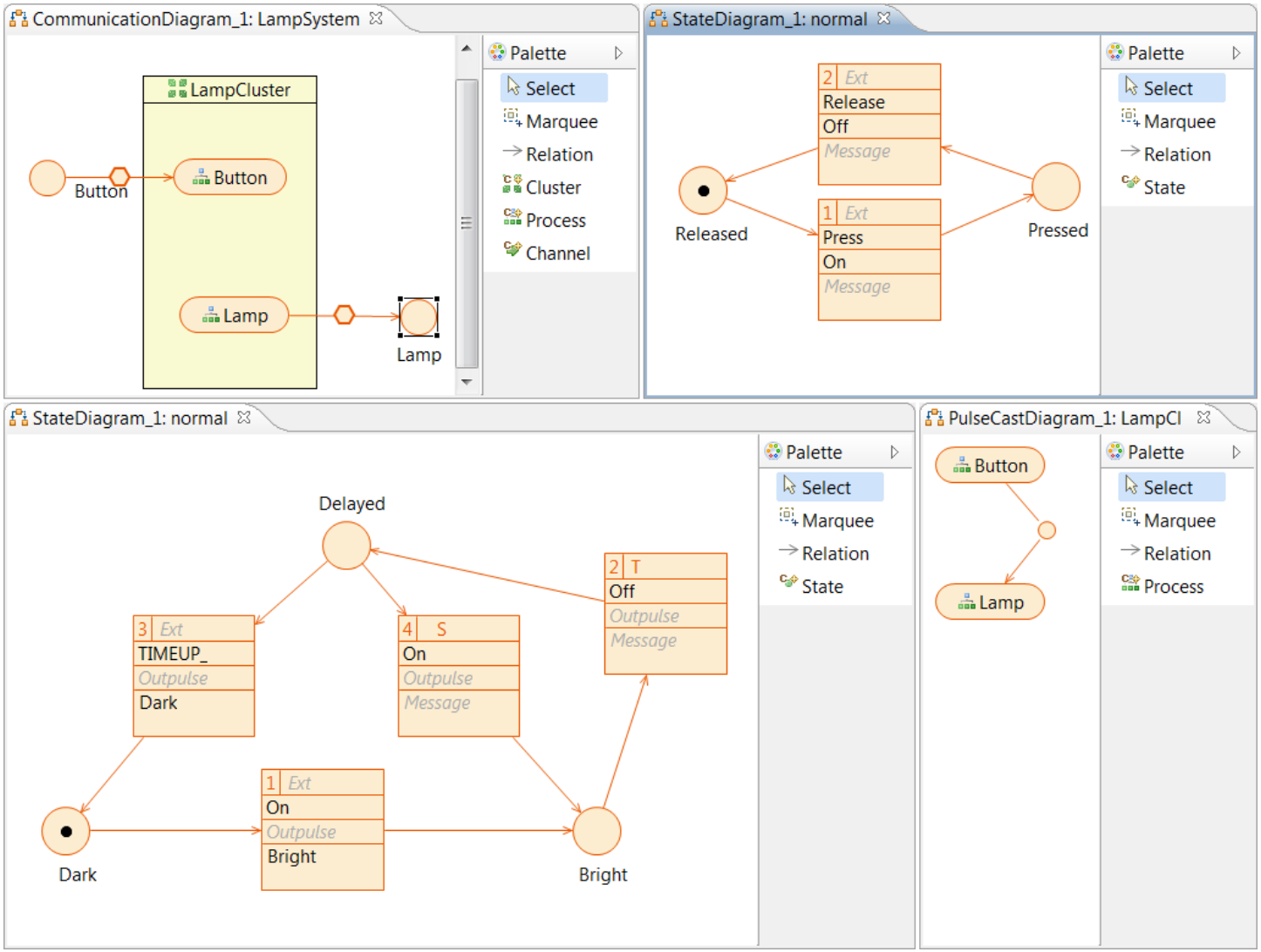
- Open SimpleProject/Properties/actifsource/Target Folders
- Add the existing folder `src` as target folder
- Add the BuildConfig **CIP_C** to the target `src`
- Press **OK**



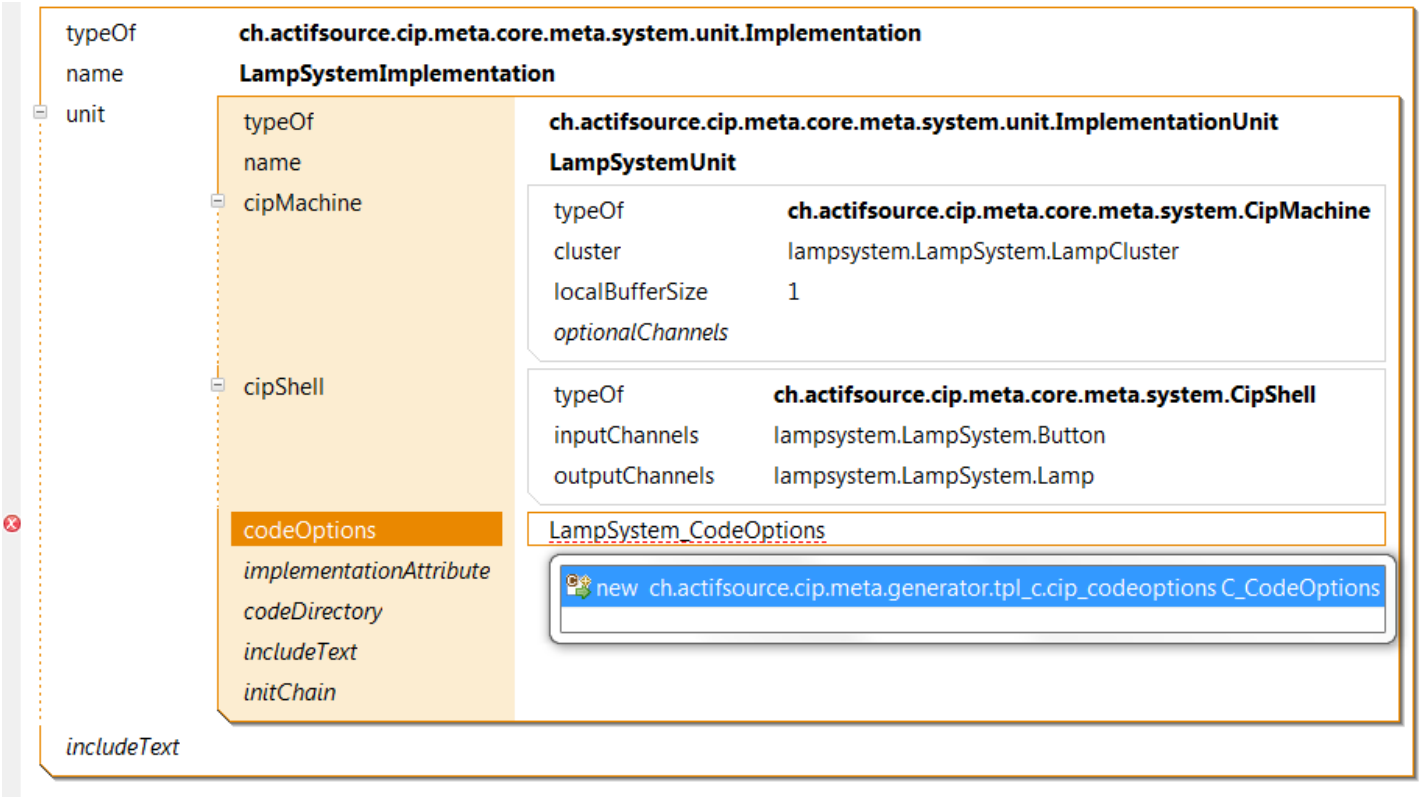
- Select the Resource Folder `asrc` in the Project Explorer
- Select **CipSystem** from the New Resource Tool in the Eclipse toolbar



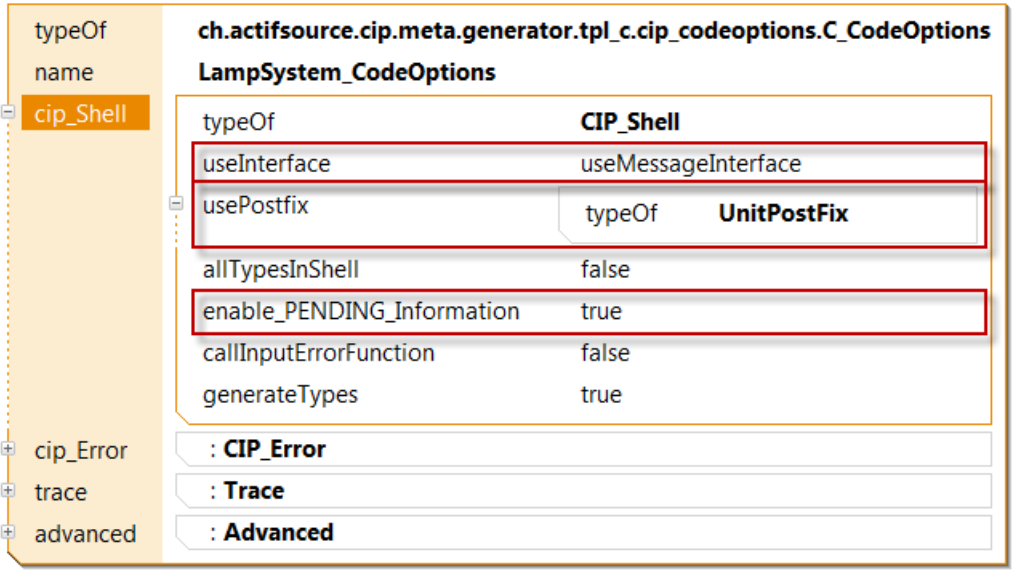
- Enter `lampsystem` as Namespace
- Enter `LampSystem` as Name
- Press **Finish**



Create a LampSystem as shown in the Actifsource Tutorial **CIP Statemachine – Lamp**



- Create a new **Implementation** named LampSystemImplementation in your LampSystem
- Create a new **ImplementationUnit** named LampSystemUnit in your LampSystemImplementation
- Create a new **CipMachine** in your LampSystemUnit
- Reference the **Cluster** LampCluster in your **CipMachine**
- Create a new **CipShell** in your LampSystemUnit
- Reference the input channel Button
- Reference the output channel Lamp
- Create a new **C_CodeOption** named LampSystem_CodeOption using Content Assist (Ctrl+Space)



- Configure the newly created LampSystem_CodeOption
- Set **useInterface** to **useMessageInterface**
- Set **usePostFix** to **InitPostFix**
- Set **enable_PENDING_Information** to **true**
- The code for your CIP system is now generated in the target folder src

```
#include "arduino.h"
#include "sLampSystemUnit.h"

int buttonPin = 2;    // button connected to digital pin 2
int lampPin =  13;    // lamp connected to digital pin 13

void AI_LampSystemUnit_C2_Dark()   {digitalWrite(lampPin, LOW);}
void AI_LampSystemUnit_C2_Bright() {digitalWrite(lampPin, HIGH);}

void iMSG_LampSystemUnit()
{
  OUT_LampSystemUnit.C2_Dark   = AI_LampSystemUnit_C2_Dark;
  OUT_LampSystemUnit.C2_Bright = AI_LampSystemUnit_C2_Bright;
}

int main()
{
  if (!fINIT_LampSystemUnit()) {return 1;}       // init cip
  init();                                        // init arduino
  pinMode(buttonPin, INPUT);                     // init button pin
  pinMode(lampPin, OUTPUT);                      // init lamp pin

  int oldbuttonState = 0;
  while (1)
  {
    int newButtonState = digitalRead(buttonPin); // read button pin
    TRG_LampSystemUnit.TICK_();                  // ticking time
    if (oldbuttonState!=newButtonState && newButtonState==1) {IN_LampSystemUnit.C1_Press();}
    if (oldbuttonState!=newButtonState && newButtonState==0) {IN_LampSystemUnit.C1_Release();}
    oldbuttonState=newButtonState;
    while (TRG_LampSystemUnit.PENDING_.ANY_) {TRG_LampSystemUnit.STEP_();}
    delay(3);
  }
  return 0;
}
```

↳ Rewrite your `main.c` as shown above

ⓘ Your code is probably not linking if you named your file `main.cpp` instead if `main.c`

C/C++ - SimpleProject/src/main.c - Eclipse

File Edit Source Refactor Navigate Search AVR Project Run Window Help

EditorTools

Project Explorer

- SimpleProject
  - asrc
  - Actifsource
  - bin
  - Binaries
  - Includes
  - JRE System Library [JavaSE-1.6]
  - Release
  - src
    - arduino
    - Button.c
    - eLampSystemUnit.c
    - eLampSystemUnit.h
    - Lamp.c
    - main.c
    - mLampSystemUnit.c
    - mLampSystemUnit.h
    - sLampSystemUnit.c
    - sLampSystemUnit.h
    - tLampSystemUnit.c
    - tLampSystemUnit.h
  - src-gen

main.c

```
int oldbuttonState = 0;
while (1)
{
  int newButtonState = digitalRead(buttonPin); // read button pin
  TRG_LampSystemUnit.TICK_();                  // ticking time
  if (oldbuttonState!=newButtonState && newButtonState==1) {IN_LampSy
  if (oldbuttonState!=newButtonState && newButtonState==0) {IN_LampSy
  oldbuttonState=newButtonState;
  while (TRG_LampSystemUnit.PENDING_.ANY_) {TRG_LampSystemUnit.STEP_(
  delay(3);
}
return 0;
```

Problems | Tasks | Console | Properties | AVR Device Expl | AVR Supported

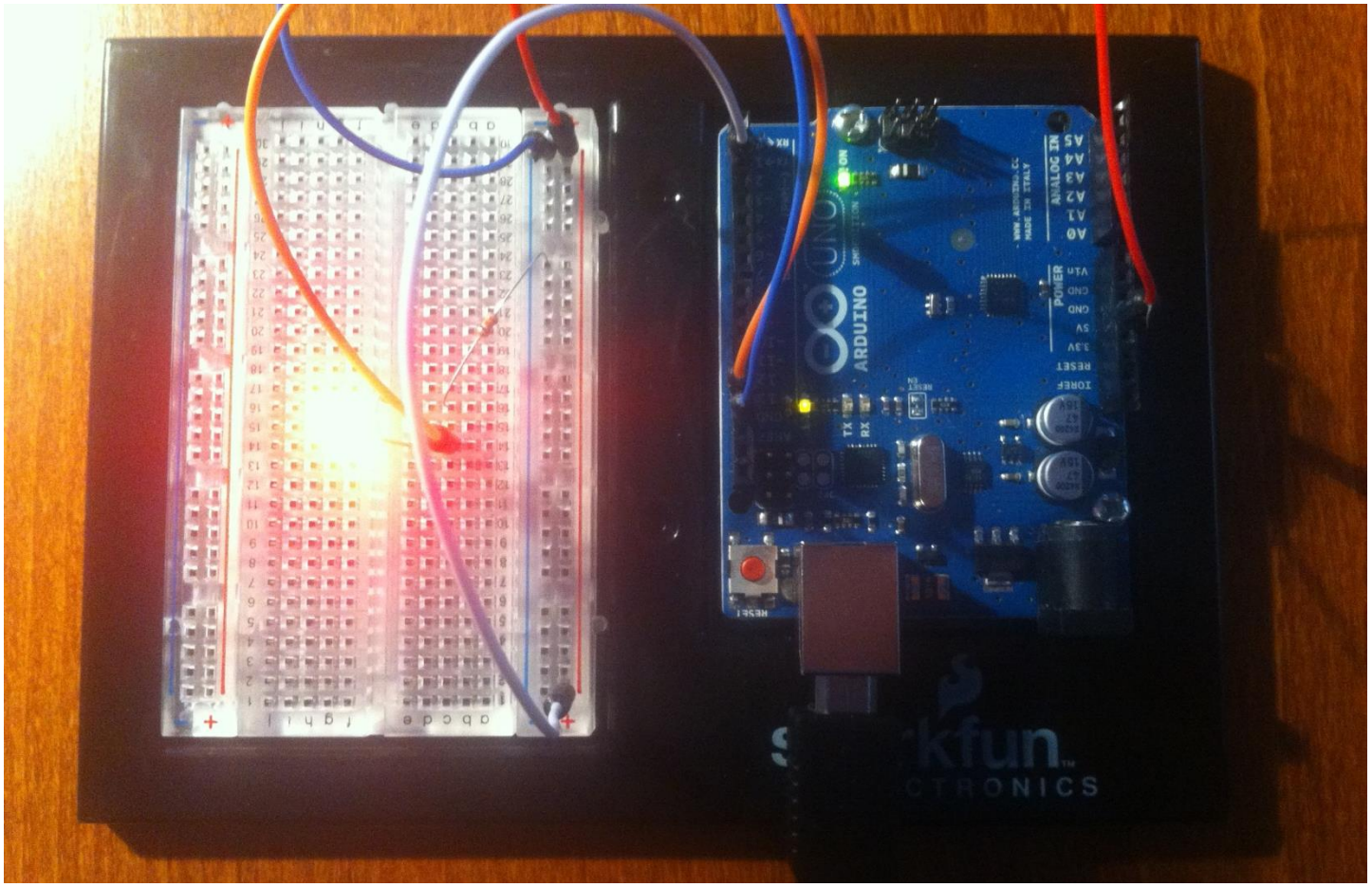
AVRDude

```
Writing | ################################################## | 100% 3.60s

avrdude: 22452 bytes of flash written
avrdude: verifying flash memory against SimpleProject.hex:
avrdude: load data flash data from input file SimpleProject.hex:
avrdude: input file SimpleProject.hex auto detected as Intel Hex
avrdude: input file SimpleProject.hex contains 22452 bytes
avrdude: reading on-chip flash data:
```

SimpleProject

- Download your code to the Arduino
  - Select your project SimpleProject in the Project Explorer
  - Press the AVR download button in the Eclipse toolbar 



- ⓘ We built a lamp with a delayed off switch
  - Pin 2 on GND means Button is released
  - Pin 2 on 5V means Button is pressed
- Start while Button is released
- Press Button → Light should switch on
- Release Button → Light should switch off with delay
  - Tune with the delay function of your timer and the delay() call in the main.c