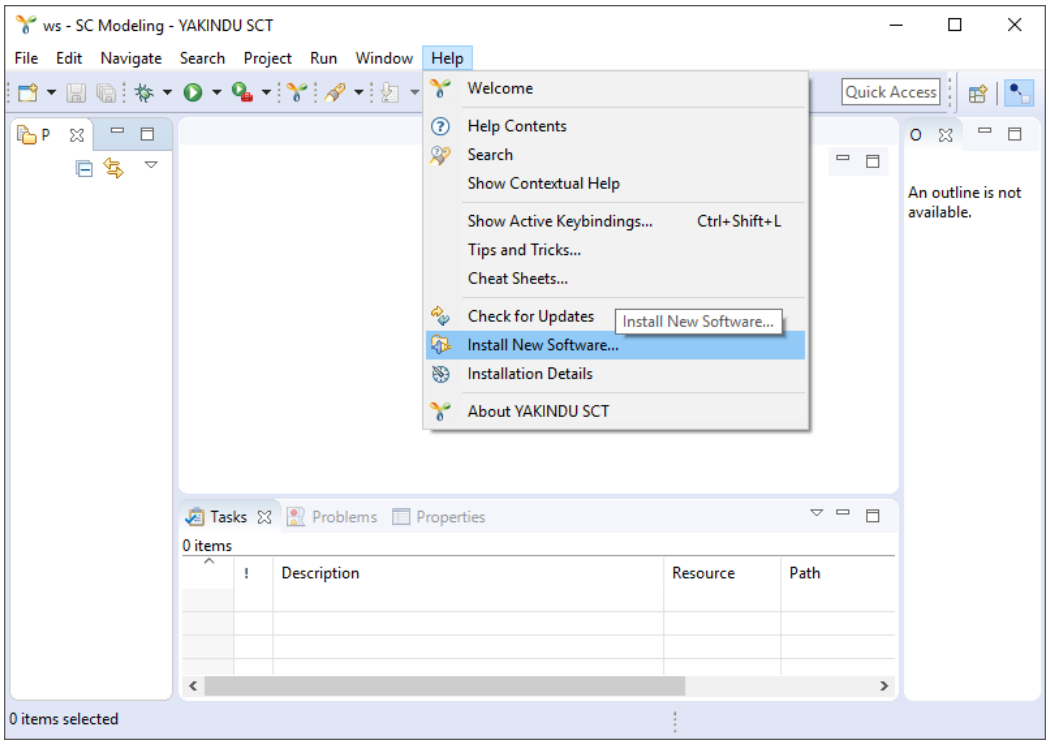
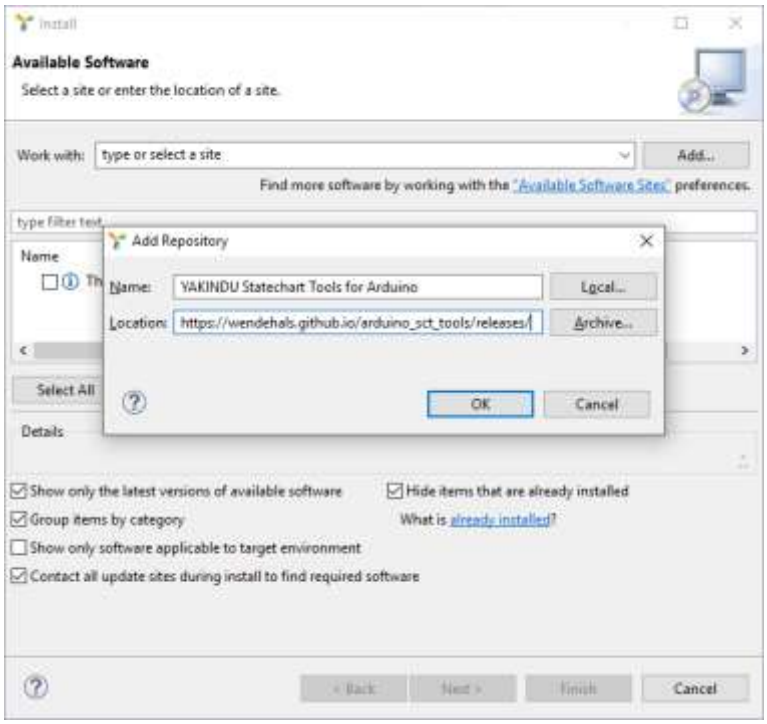
HOW TO INSTALL YAKINDU AND TRICKS ABOUT IT

1-Installation

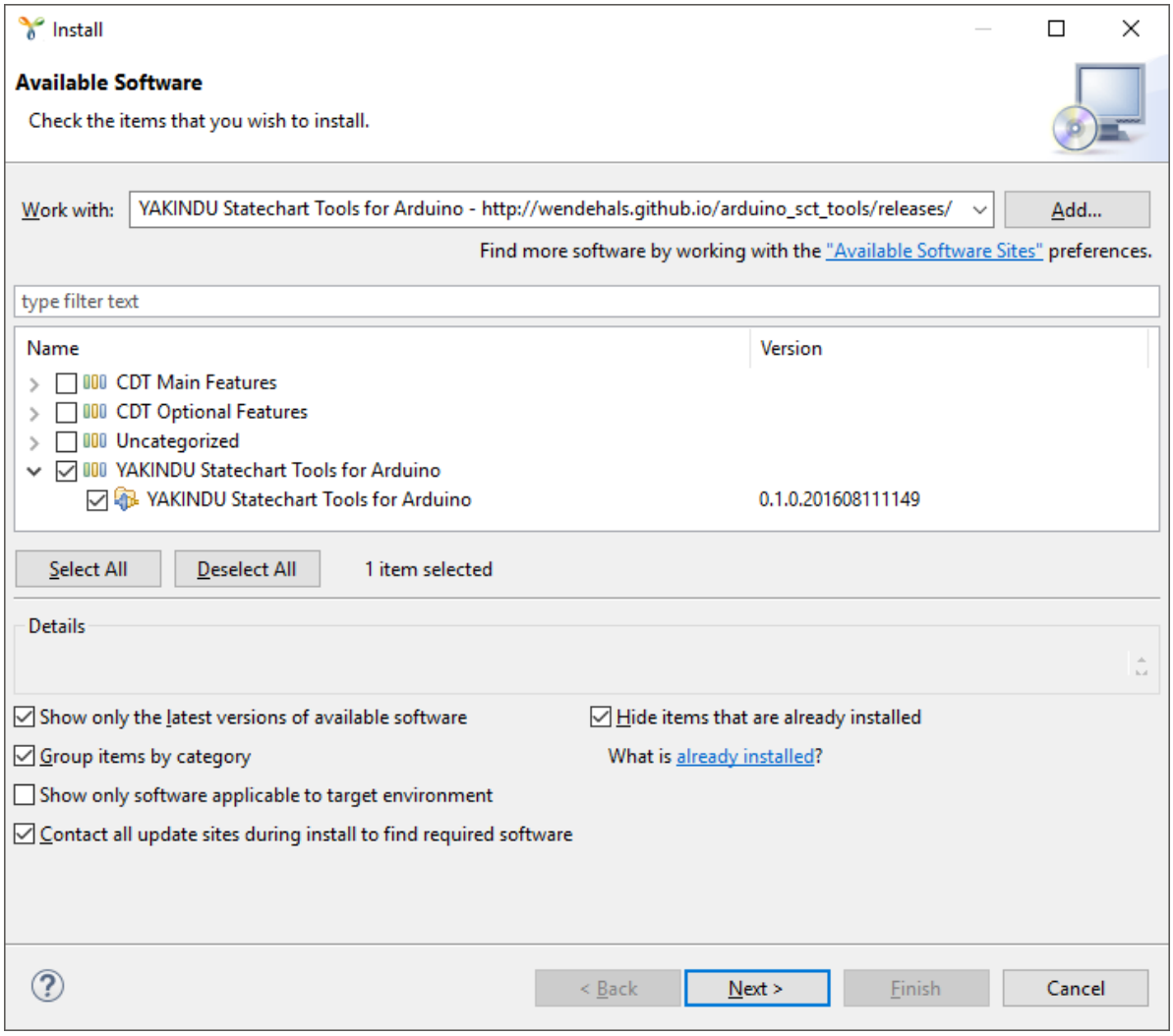
First you need to download the YAKINDU Statechart Tools. Unzip the archive to an arbitrary directory and start SCT. Choose "Install New Software..." from the "Help" menu.



Select the "Add..." button in the upper right corner to add a new update site. Copy the following URL to the "Location" text field: https://wendehals.github.io/arduino_sct_tools/releases/. Use any value for the "Name" field and select "OK".

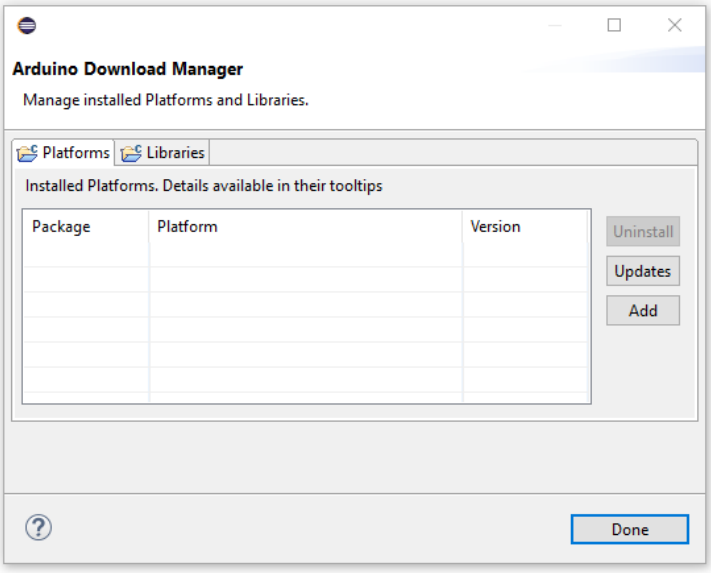


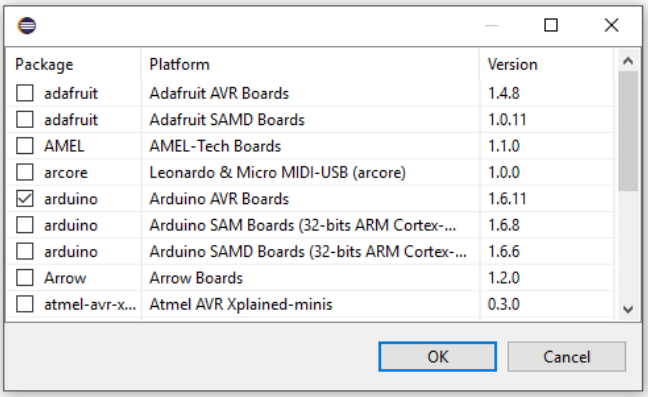
Select YAKINDU Statechart Tools for Arduino and press the "Next>" button to switch to the next page.



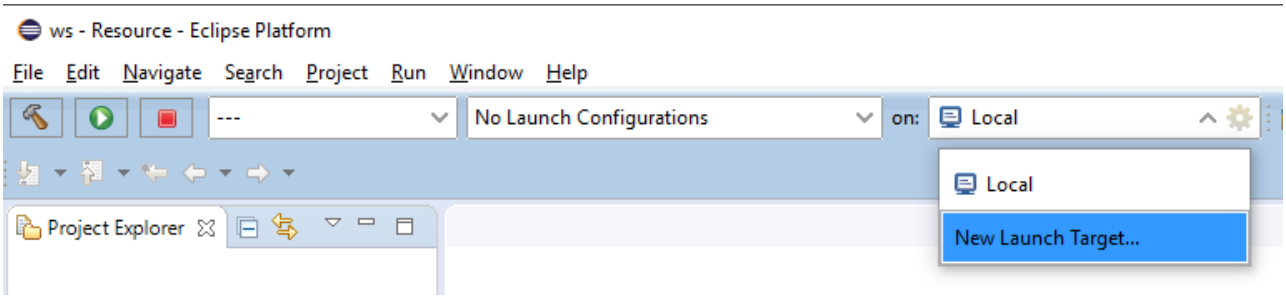
2-Arduino Toolchain Setup

Before starting development for your Arduino you need to install and setup the Arduino toolchain and libraries in your freshly installed Eclipse environment. Open the Arduino Downloads Manager from the Help menu. In the Platforms tab add a new toolchain by clicking the "Add" button and choosing the target platform. In our case it's the Arduino AVR Boards package.

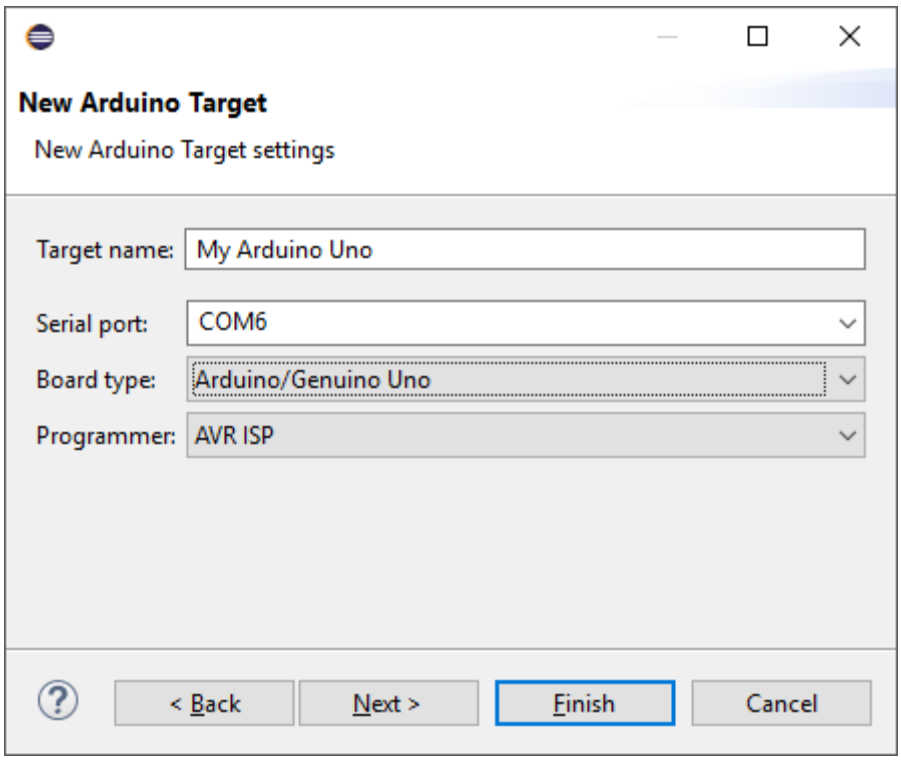




The Arduino toolchain and libraries are now installed. To upload the program to your Arduino you need to connect it to Eclipse. First, connect your Arduino via USB to your computer. If you have connected and used your Arduino with your computer before there should already be a driver installed. If not, make sure you have installed the Arduino IDE from arduino.cc, you need it for the USB driver.
There is a wizard that helps you to create a connection to your Arduino. You can open the wizard either by choosing "New Launch Target" from the toolbar or by clicking the "New Connection" button in the "Connections" view of the C/C++ perspective.



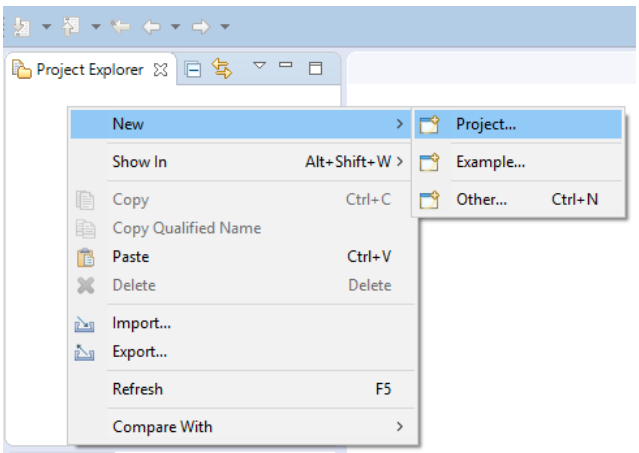
On the first wizard page select the Arduino list entry and click "Next>". On the second page provide a name for the connection, the serial port that, and the board type and press "Finish".



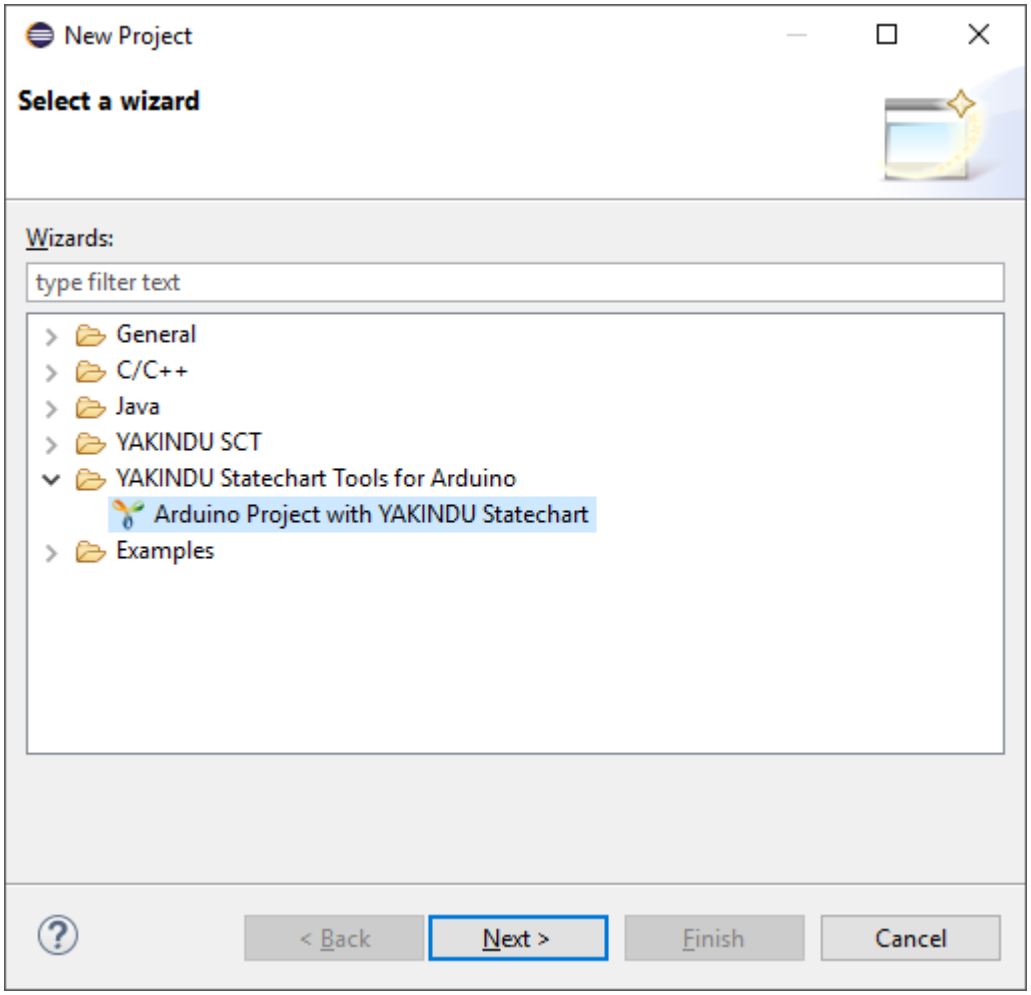
The environment is now ready to compile and upload programs to your Arduino.

3-Arduino SCT Project Setup

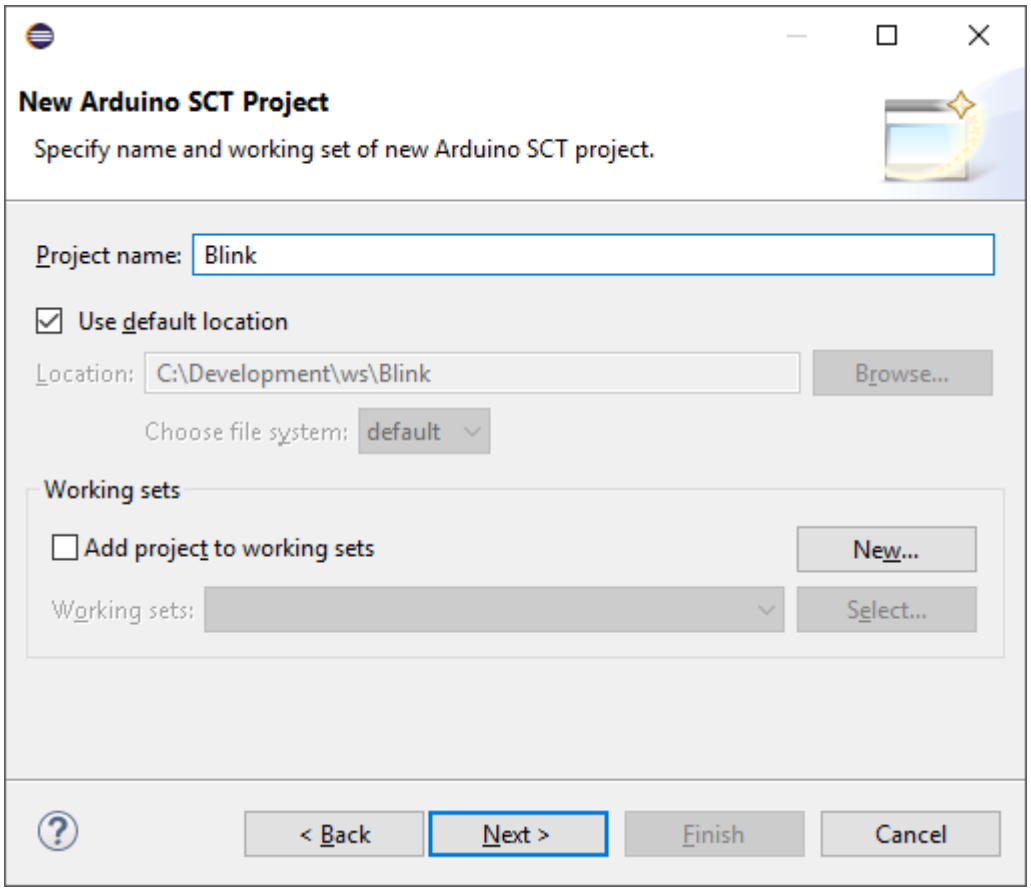
To start programming your Arduino, you need to create a project. After closing the Welcome page open the context menu of the Project Explorer view in the upper left corner of the Eclipse Window and select "New">"Project...".



Select "Arduino Project with YAKINDU Statechart" from the list on the first page of the New Project wizard and click "Next>".



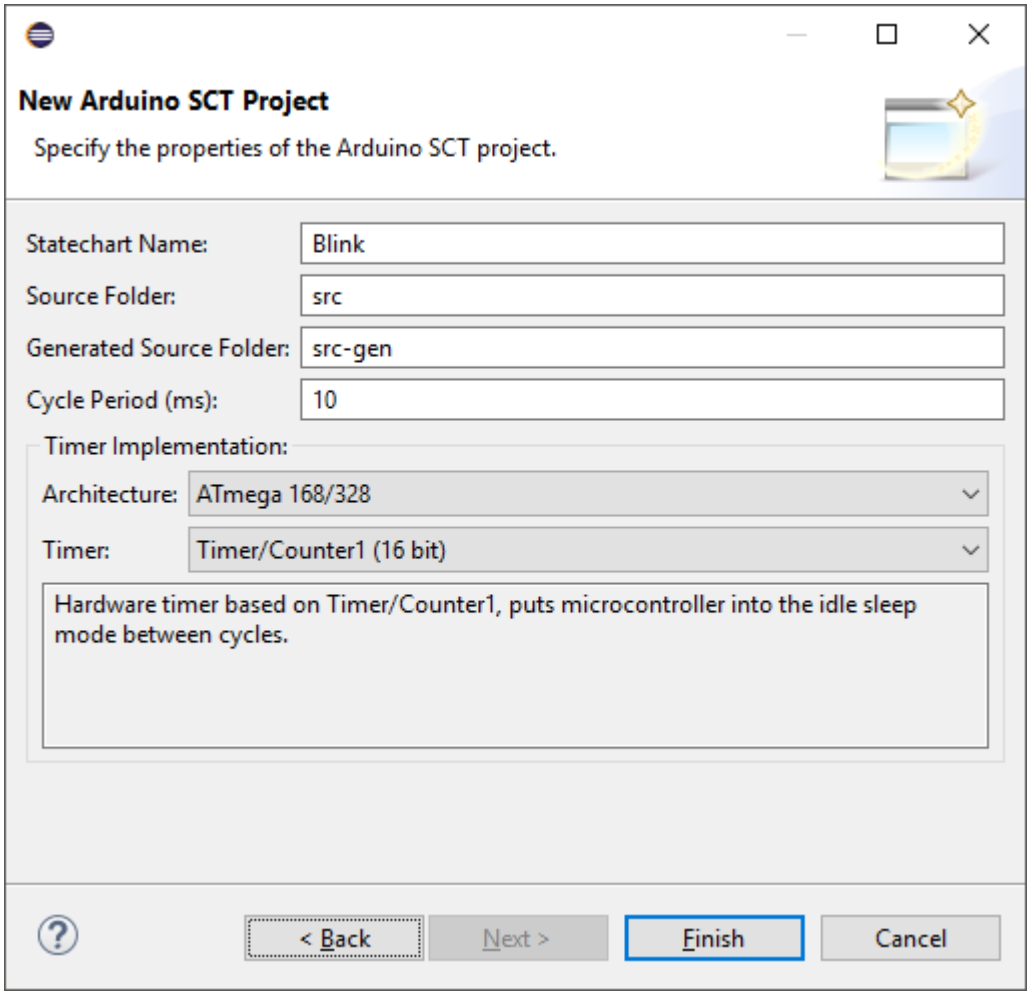
On the next page, choose a project name and click "Next>".



On the last page you can choose values for a set of properties of the Arduino SCT project. These are

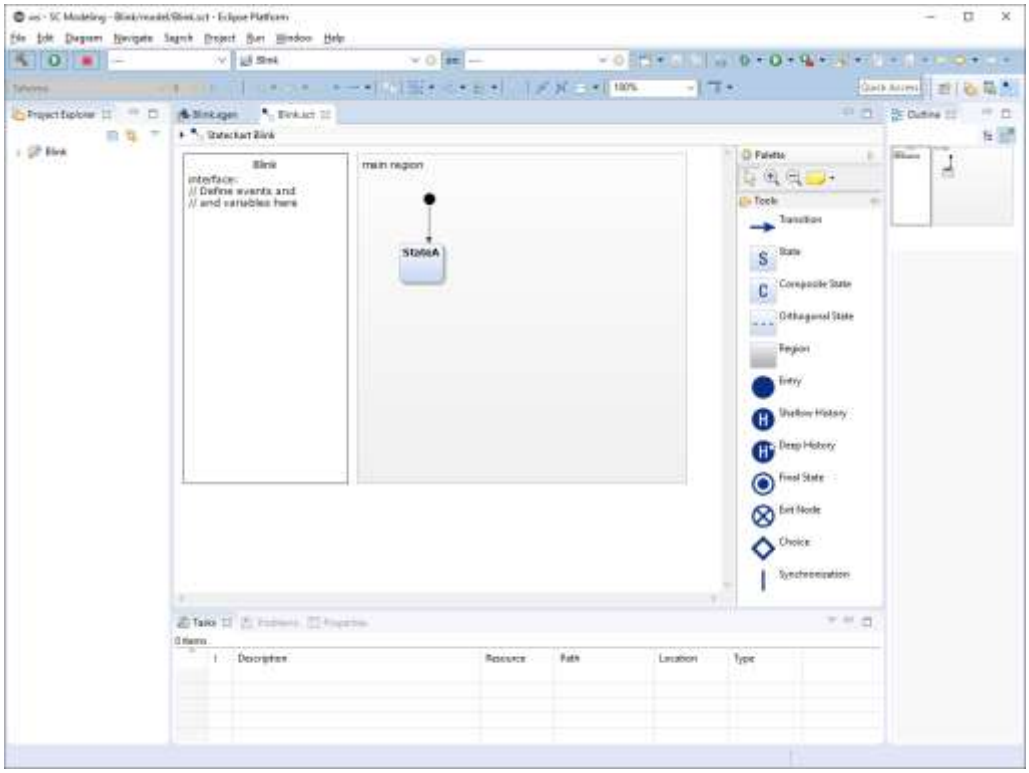
- a name for the statechart. The default is the project name. You can choose any valid C++ identifier, since it will be used as a class name. You can't use in the name characters as 2, -,+…it 's generating an error.
- a source code folder - it will contain the source code that will be edited by you. Initially, two files will be generated and put into this folder, <StatechartName>Connector.h and <StatechartName>Connector.cpp. After the first code generation these files will be kept untouched by the code generation.
- a generated source code folder - it will contain all the rest of the source code generated by the tool that is necessary to run your statechart on the Arduino. The source code in this folder should not be changed by the user and will be overwritten by the code generator.
- the timer implementation - currently, there are two timer implementations to run the cycles of the statechart. You can choose between a software based timer that is compatible to all kinds of Arduinos and a hardware based timer of the ATmega168 and ATmega328 microcontrollers.

After setting the properties, click "Finish".



You might be asked whether the Xtext nature should be added to the project. Please confirm this question.

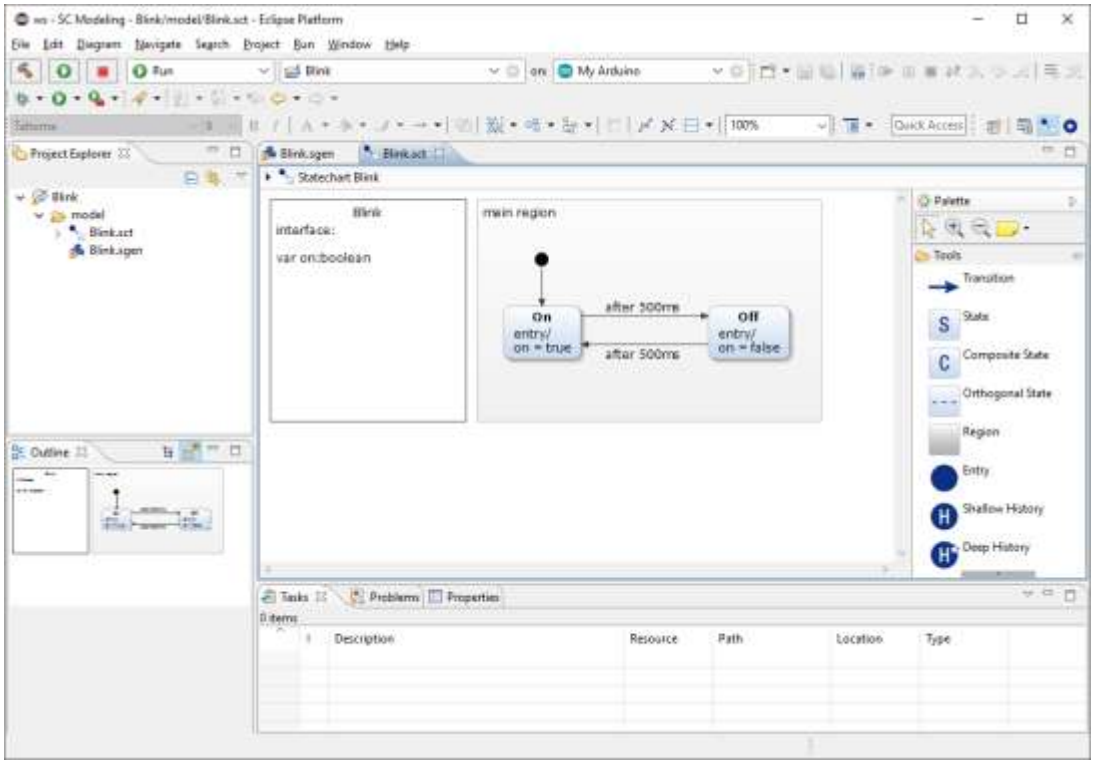
Your Eclipse should look like this:



4-Programming an Arduino with Statecharts

Modeling the Statechart

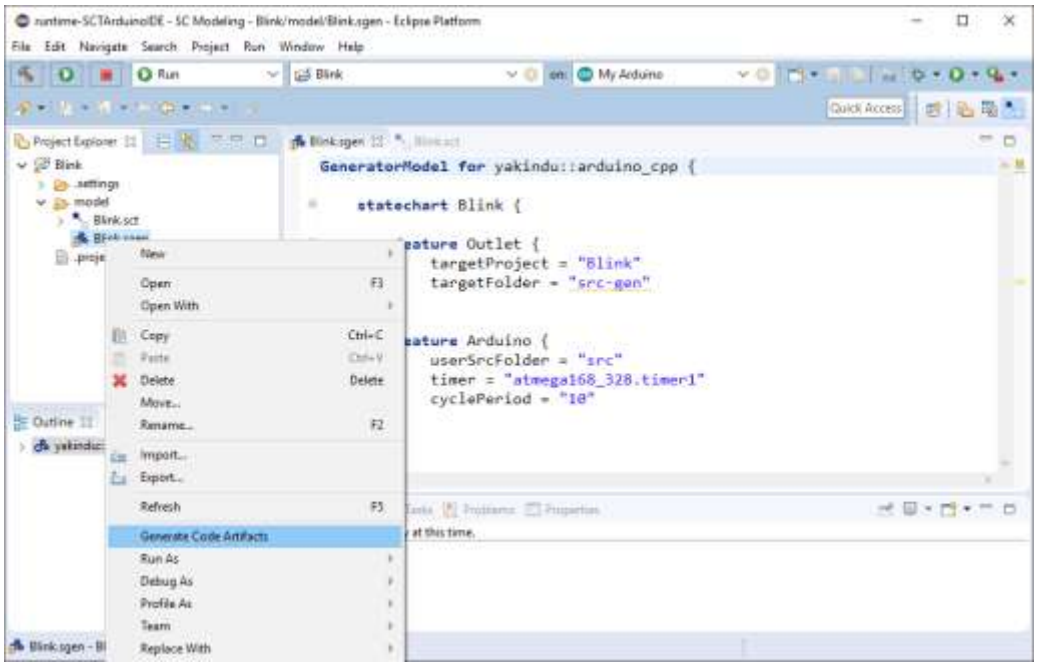
Now we are ready to model the statechart. We will use the "Hello World" example from the Arduino world, it's a blinking light. The Arduino Uno board has an LED built in that can be used for this purpose. In the "Blink" example, we will let the LED turn on and off every 500ms.
Model your statechart in the way that is shown in the figure below. Create two states, On and Off. Then create transitions between both of them and add time triggers to the transitions. The time trigger after 500ms lets the state automatically change after 500 milliseconds. Within the states, add an entry action that sets the boolean variable on to true or respectively to false. You need to declare the boolean variable on within the interface.



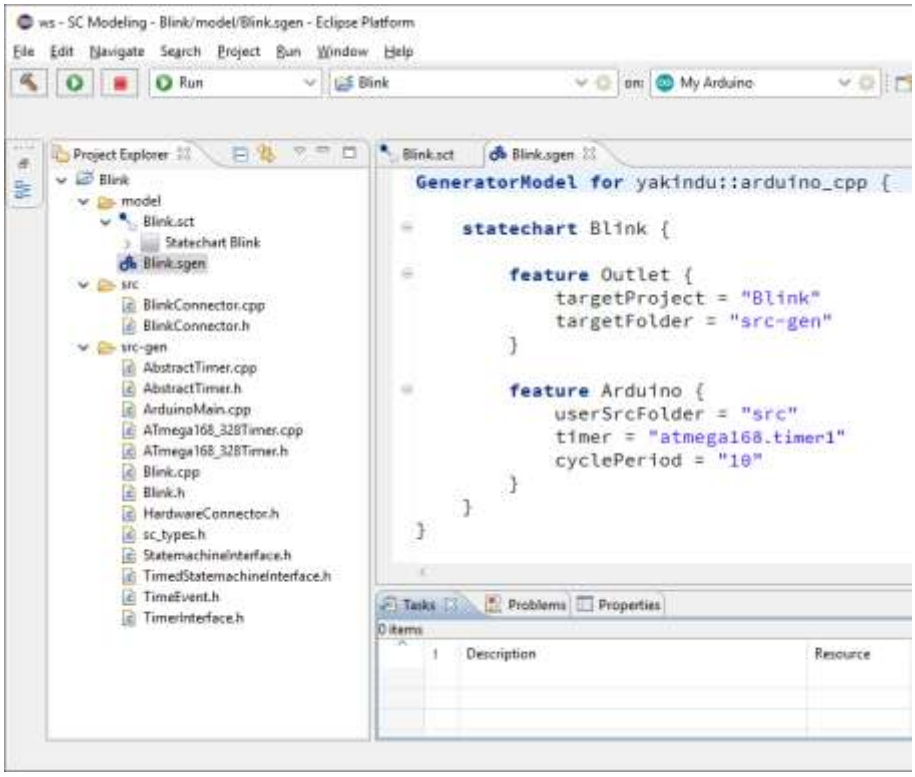
You will find a more detailed documentation of the YAKINDU Statechart Tools and the statechart syntax in the Eclipse help or online.

Source code generation

Our statechart model for the "Blink" example is now complete. Open the context menu on the "Blink.sgen" file in the Project Explorer and select "Generate Code Artifacts".



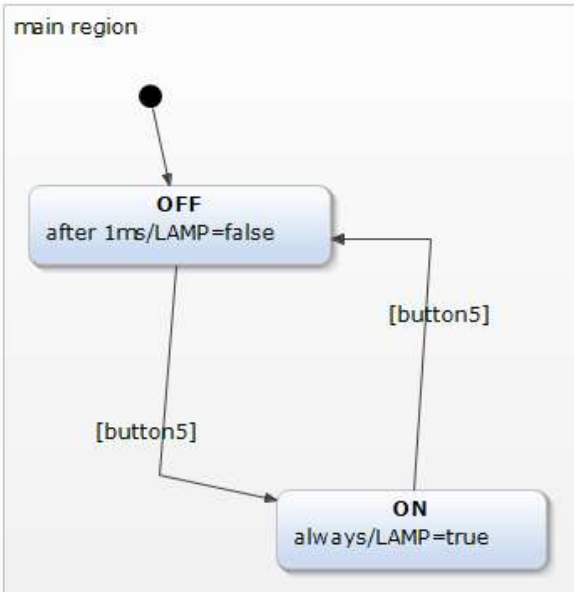
Your Eclipse environment should now look like depicted in the figure below. There are two new folders, "src" and "src-gen". The "src-gen" folder contains the statemachine generated from the statechart, interfaces to the statemachine, a timer implementation, and everything else that is needed to execute the statemachine on your Arduino. The "src" folder is meant to be changed by the developer. We will go into detail later.



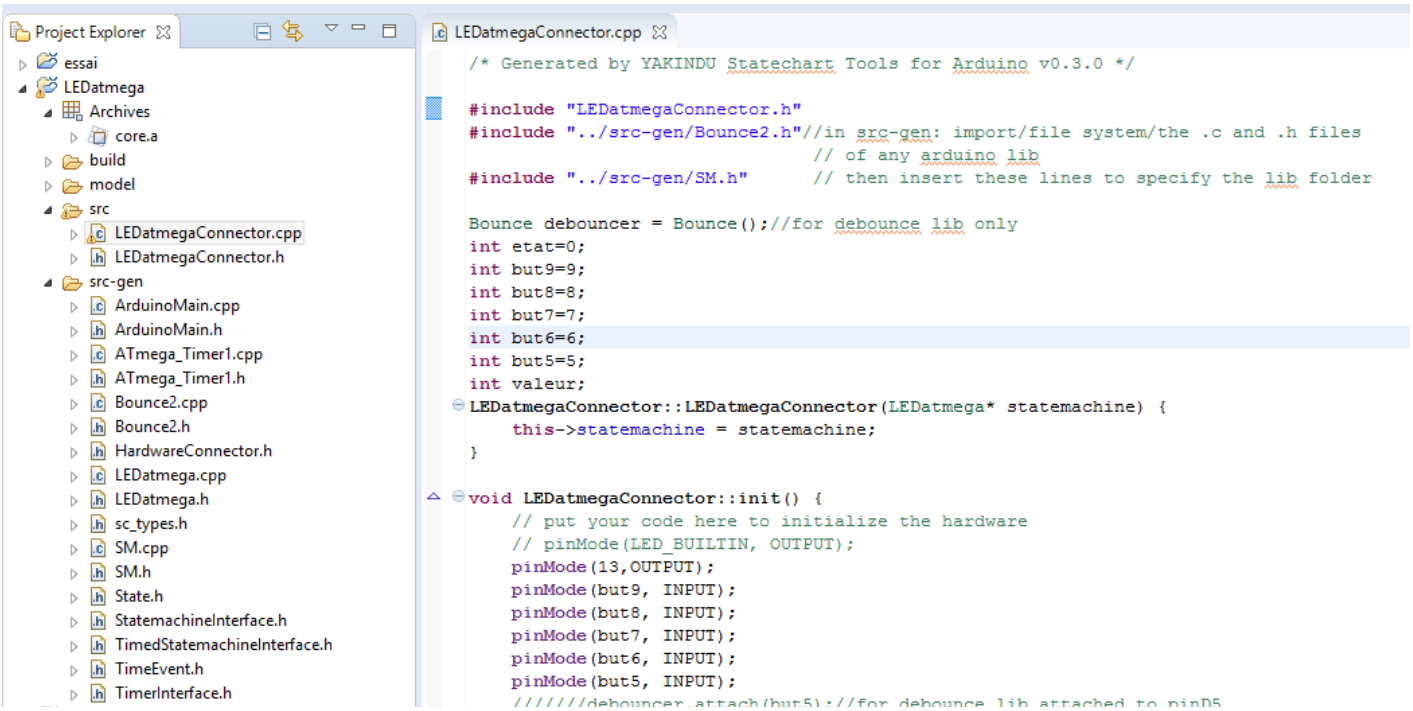
You don't have to care about the code that is put into the "src-gen" folder. It is completely generated and will be overwritten the next time when the statechart changes.

Some other tricks:

- An “after x ms” is necessary in your state chart not to have a code error in xxxxx.cpp



- Run/Run configuration/Statechart simulation/your state chart to change “cycle based” and cycle period
- Click on xxxx.sgen to change arduino cycle period, then don’t forget save/generate the code/build the project
- To insert library put it in the src-gen folder or right click on src-gen/import files and select the folder where is your library to add. Then call it in src/xxxConnector.cpp:
  `#include "../src-gen/thelib.h"`



If there are troubles in building with some imported library: correct `#include <myLib>` by `#include “myLib”`
If a library include folders, put all the fill in the same YAKINDU folder: src-gen and don’t forget: “save/clean/build” to correct each error.

- State to action/ transition to sensor link: modify the xxxxConnector.cpp in the src folder with a look on the file src-gen/xxx.h to get the good generated functions.

```
/* Generated by YAKINDU Statechart Tools for Arduino v0.3.0 */

#include "LEDatmegaConnector.h"
#include "../src-gen/Bounce2.h"//in src-gen: import/file system/the .c and .h files
                               // of any arduino lib
#include "../src-gen/SM.h"      // then insert these lines to specify the lib folder

Bounce debouncer = Bounce();//for debounce lib only
int etat=0;
int but9=9;
int but8=8;
int but7=7;
int but6=6;
int but5=5;
int valeur;
LEDatmegaConnector::LEDatmegaConnector(LEDatmega* statemachine) {
    this->statemachine = statemachine;
}

void LEDatmegaConnector::init() {
    // put your code here to initialize the hardware
    // pinMode(LED_BUILTIN, OUTPUT);
    pinMode(13,OUTPUT);
    pinMode(but9, INPUT);
    pinMode(but8, INPUT);
    pinMode(but7, INPUT);
    pinMode(but6, INPUT);
    pinMode(but5, INPUT);
    ///////debouncer.attach(but5);//for debounce lib attached to pinD5
    ///////debouncer.interval(10);//delay reaction

    // The state machine has already been initialized and started before.
    // If the cycle period is very high (let's say >> 1s), it takes some
    // time until runCycle() is called the first time. During that time,
    // the hardware is not in sync with the state machine. So it might be
    // better to call runCycle() once manually, to get in sync with the
    // initial state of the state machine.
}

void LEDatmegaConnector::runCycle() {
    // put your code here to update the hardware depending on the state machine's state
    // digitalWrite(LED_BUILTIN, statemachine->getXXX());

    //////debouncer.update();//launch debounce function

    digitalRead(but7);
    digitalRead(but6);

        //if (digitalRead(but5)==1) statemachine->raise_button5();//button5 is declared "in even

        //if (RE(digitalRead(but5), etat)==1) statemachine->set_button5(1);//button5 is declared
                                                                          //"var button5:integer"
                                                                          // or "var button5:boole

        //else statemachine->set_button5(0);


        //////if (debouncer.rose()) statemachine->raise_button5();//OK with bebounce lib
                                                              //rising edge detection: .rose()
                                                              //falling edge detection: .fell()
        //else statemachine->set_button5(0);

        if (RE(digitalRead(but5), etat)==1) statemachine->raise_button5();//OK with SMlib
        digitalWrite(13, statemachine->get_lAMP());

}
```

Example:

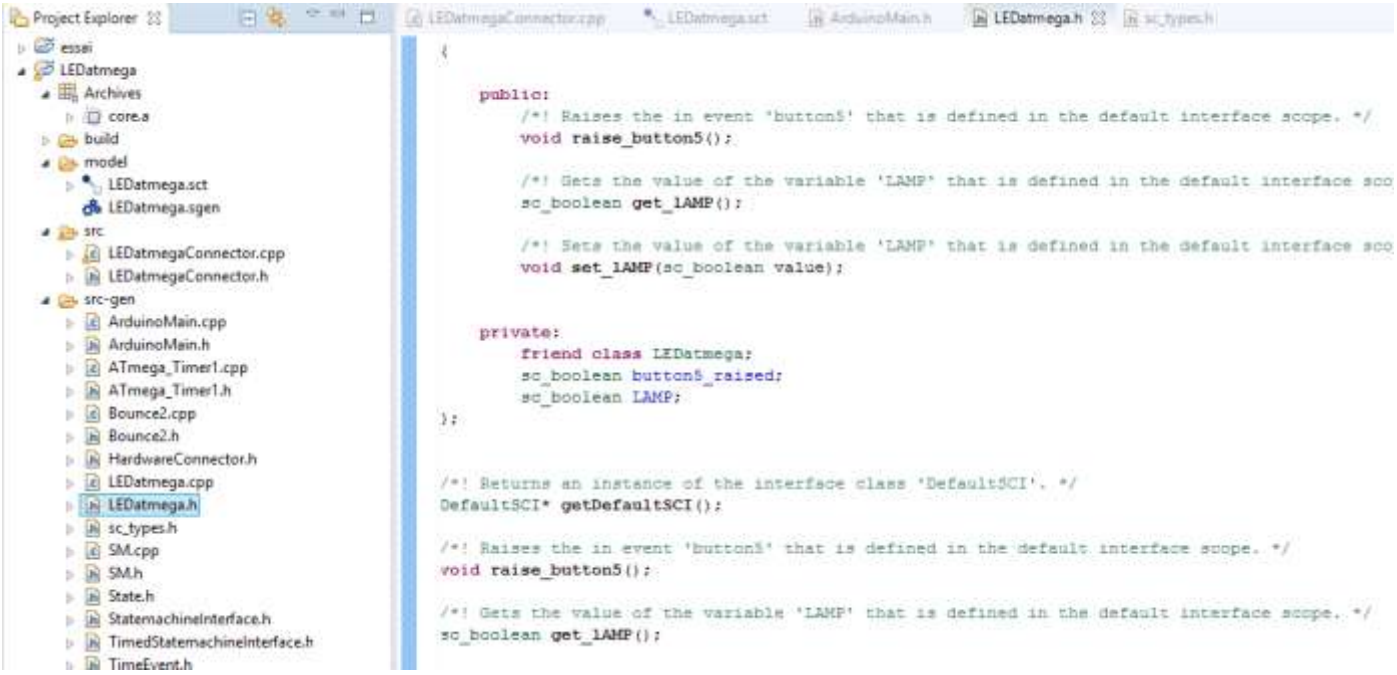
```
if (RE(digitalRead(but5), etat)==1) statemachine->raise_button5();//OK with SMlib
```

To put a rising edge on button5 owing to SM lib

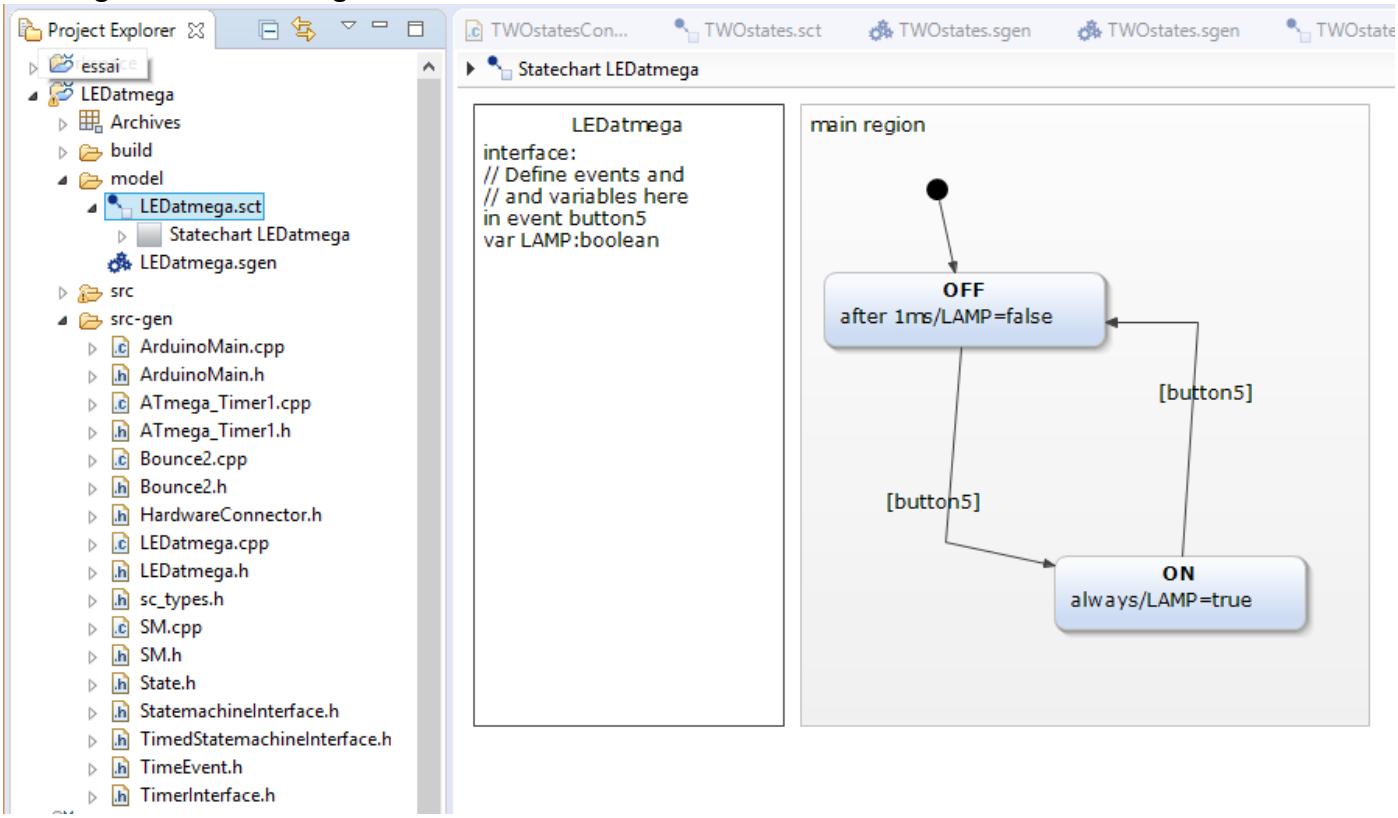
```
digitalWrite(13, statemachine->get_lAMP());
```

To link an arduino uno output to a variable in state

The file to look at:



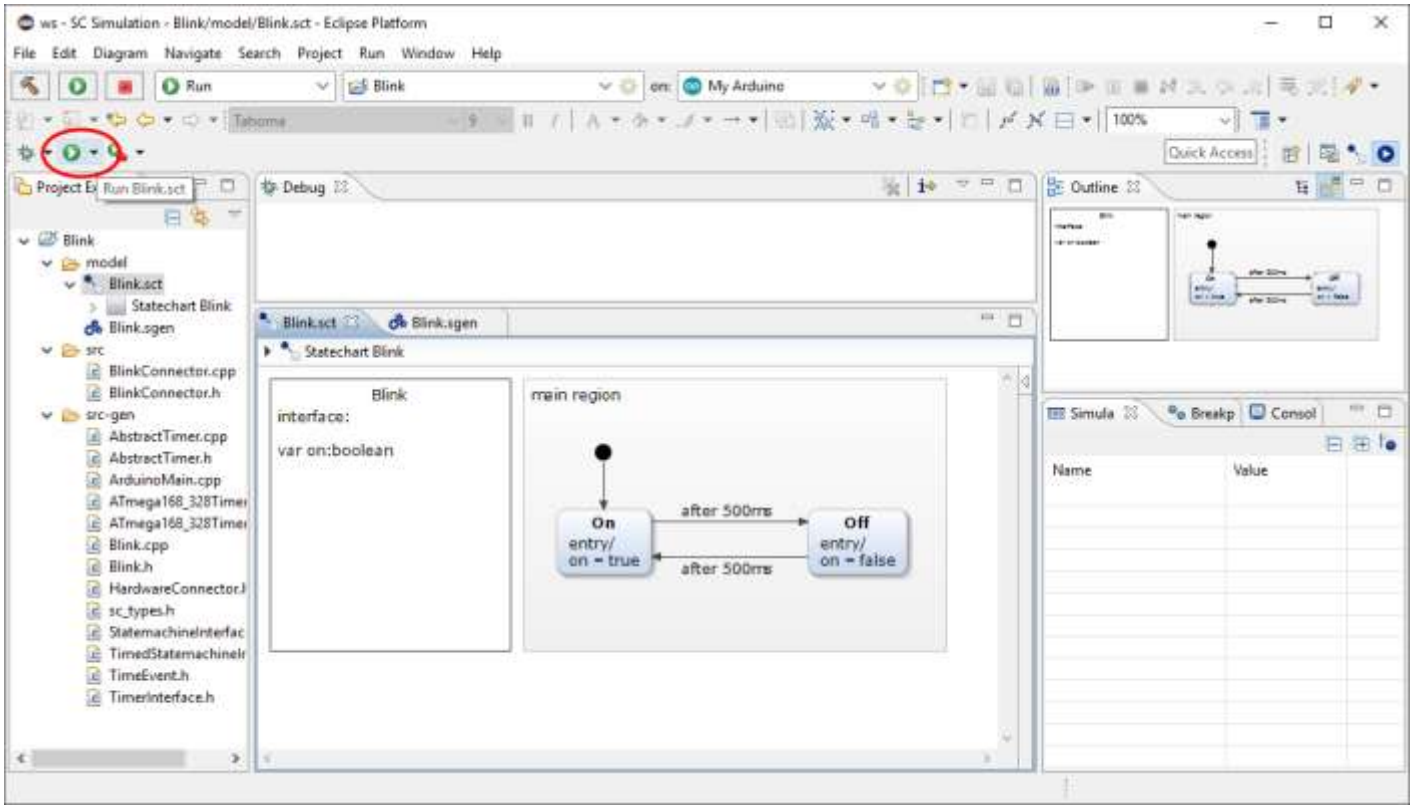
The diagram of which it's generated:



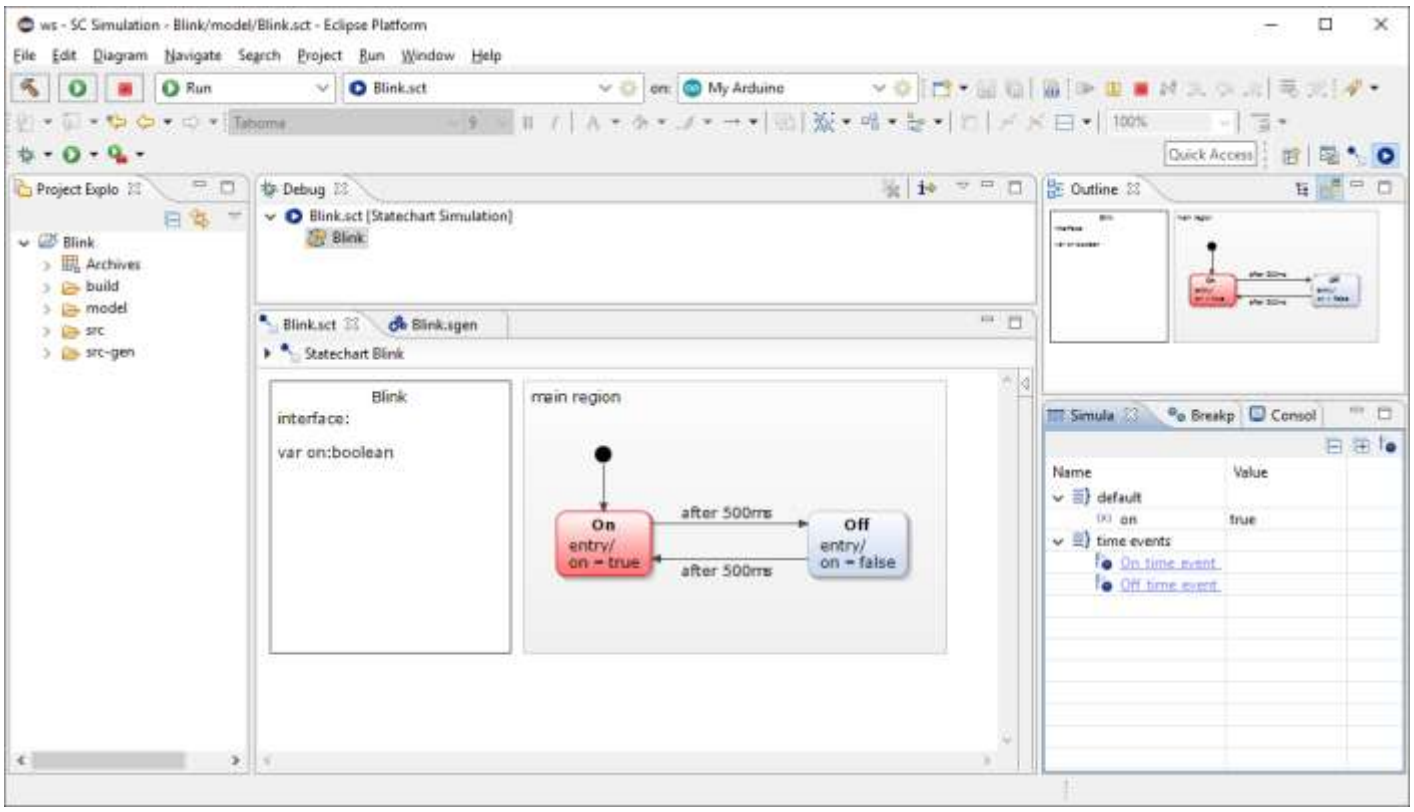
You can see in src-gen, SM.cpp and SM.h, coming from SM lib.

Simulation

This is a pretty simple statechart, you can easily check by hand whether it works as you expect. Once the statechart gets more complex, it's better to check its behavior by simulation. Switch to the SC Simulation perspective and start the simulation of the statechart by clicking on the "Run" button in the toolbar as depicted below.



Now the statechart will be simulated. The current state(s) of the statechart will be highlighted in red. On the right, variables and their values as well as events are shown in the Simulation view. You might even manipulate variables or send events manually in the Simulation view.

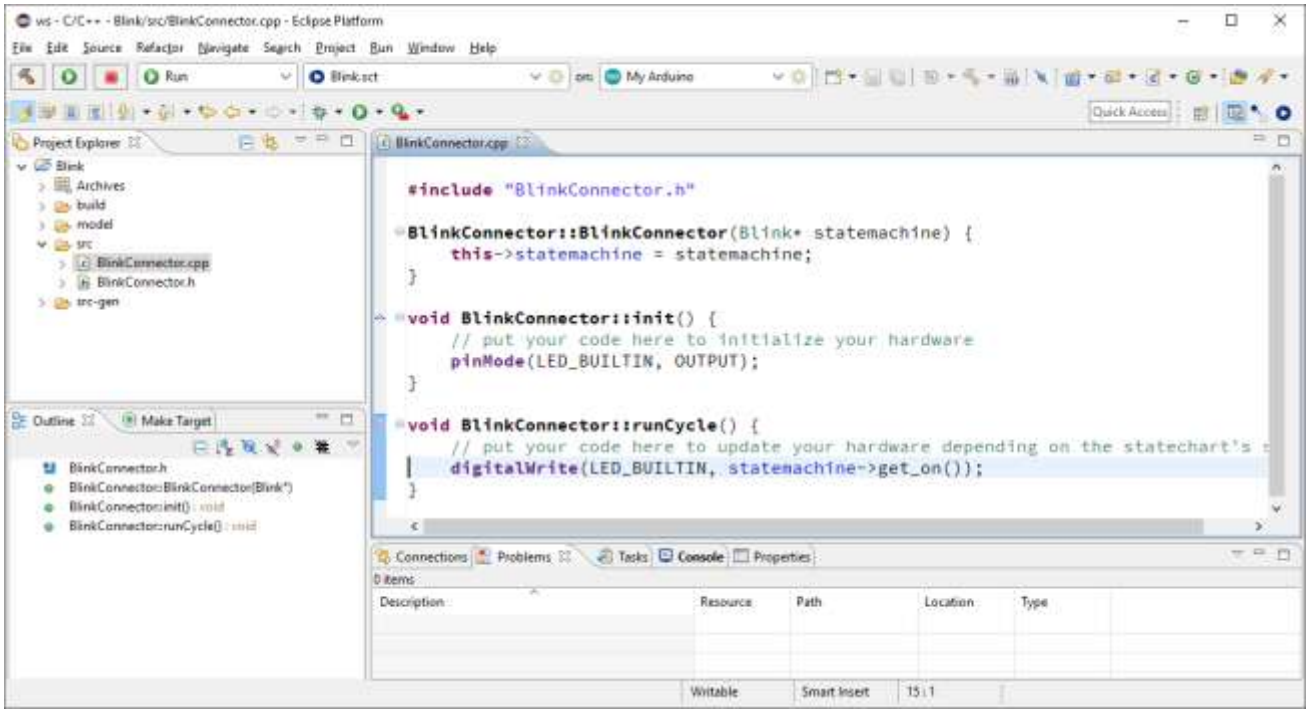


For more details about simulation refer to the YAKINDU Statechart Tools pages in the Eclipse help.

Connecting the Statechart Model with the Hardware

Once you are sure that your statechart works as expected you still need to do some programming. In the "src-gen" folder there is a file called "ArduinoMain.cpp". This file contains the `setup` and `loop` functions

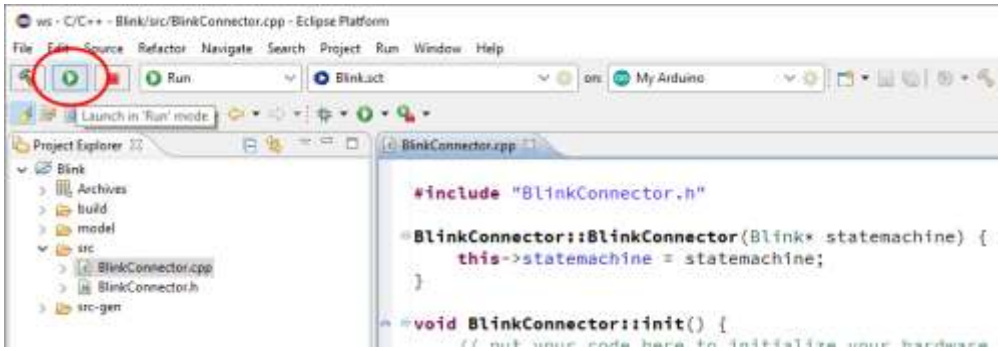
that have to be implemented in each Arduino sketch. These functions are already implemented and contain everything that is needed to run the statemachine on your Arduino.
Unfortunately, you still need to program the code that connects the statemachine to the Arduino hardware. In the "src" folder you will find two other files, "BlinkConnector.h" and "BlinkConnector.cpp" (the prefixes may differ, the class is named after the statechart). In the "BlinkConnector.cpp" file you will find two methods, `BlinkConnector::init()` and `BlinkConnector::runCycle()`. They correspond to the `setup` and `loop` functions in ordinary Arduino sketches.
Put everything that is needed to initialize your hardware into the `BlinkConnector::init()` function and everything that is needed to be executed regularly into the `BlinkConnector::runCycle()` method. In this example, you need to turn the built-in LED on the Arduino Uno board to on or off depending on the statechart's state. First of all you need to initialize the `LED_BUILTIN` pin as an `OUTPUT` pin. In the `runCycle` loop you then need to update the pin's level depending on the statechart's `on` variable to switch the LED on or off.



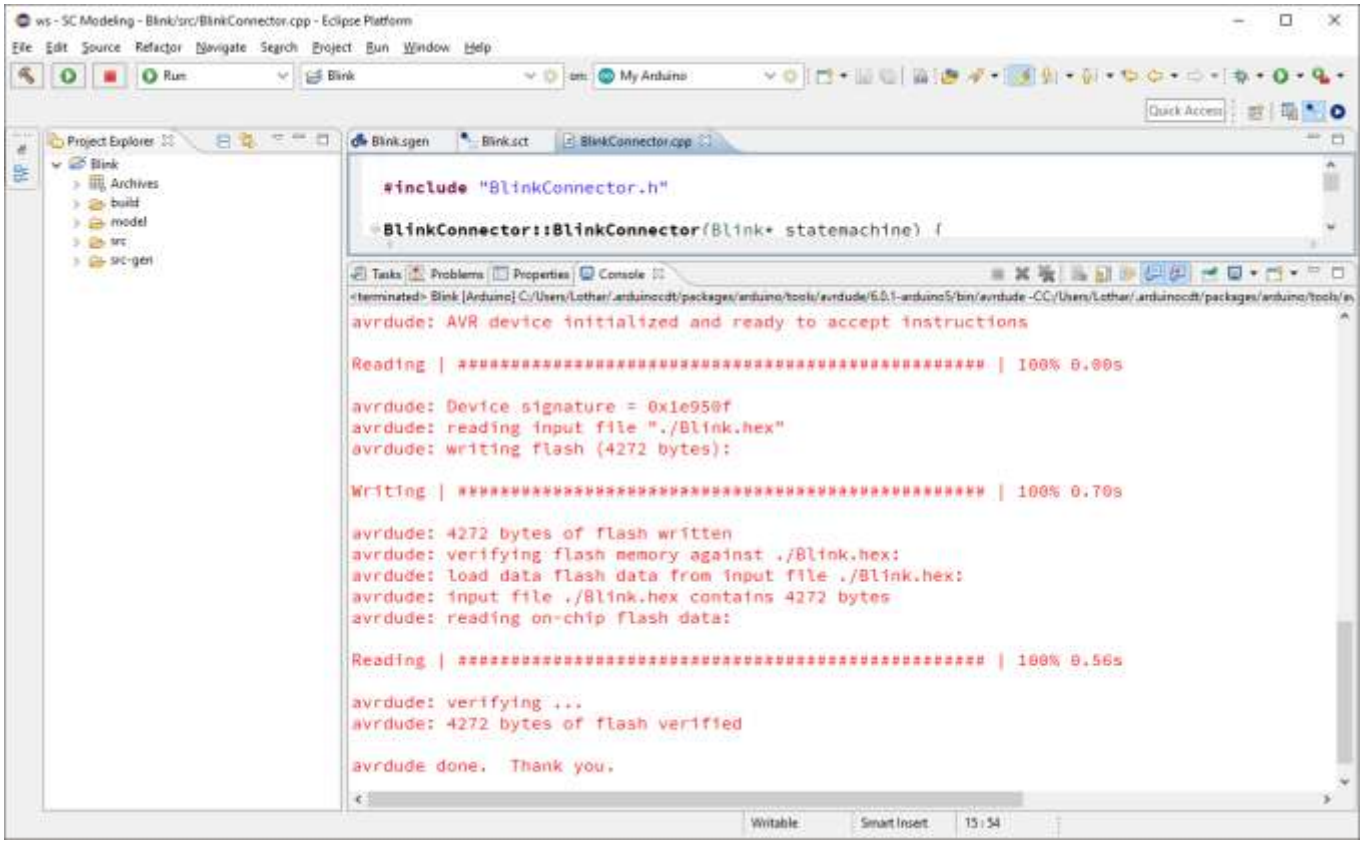
The Arduino hardware is now connected to the statemachine running on your Arduino. It's time to upload the program.

Uploading the Program

Click on the "Launch in 'Run' Mode" button to upload the code to your Arduino. Be sure to connect your Arduino to your computer and select the correct launch target in the toolbar.



If the upload was successfull the output should look like this:

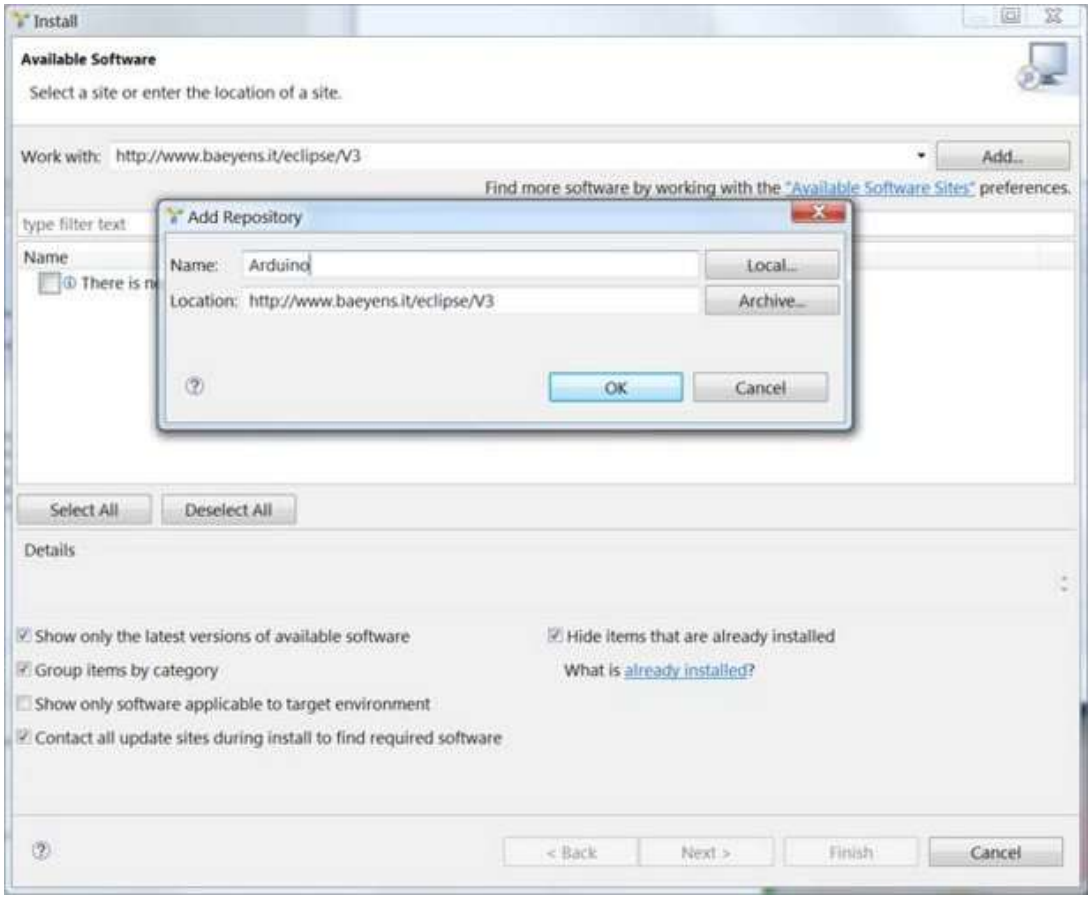


Now it's your turn to model more complex statecharts and run them on your Arduino. Have fun!

5-Creating an Example Project

There are also example projects shipped with YAKINDU Statechart Tools for Arduino. To create an example project open the context menu of the Project Explorer view in the upper left corner of the Eclipse Window and select "New">"Example...". In the wizard select one of the YAKINDU Statechart Tools for Arduino examples and click "Next>". On the next wizard page click "Finish". The example project will be created and the example statechart will be opened.

Step 3: Get the YAKINDU Statechart Tools

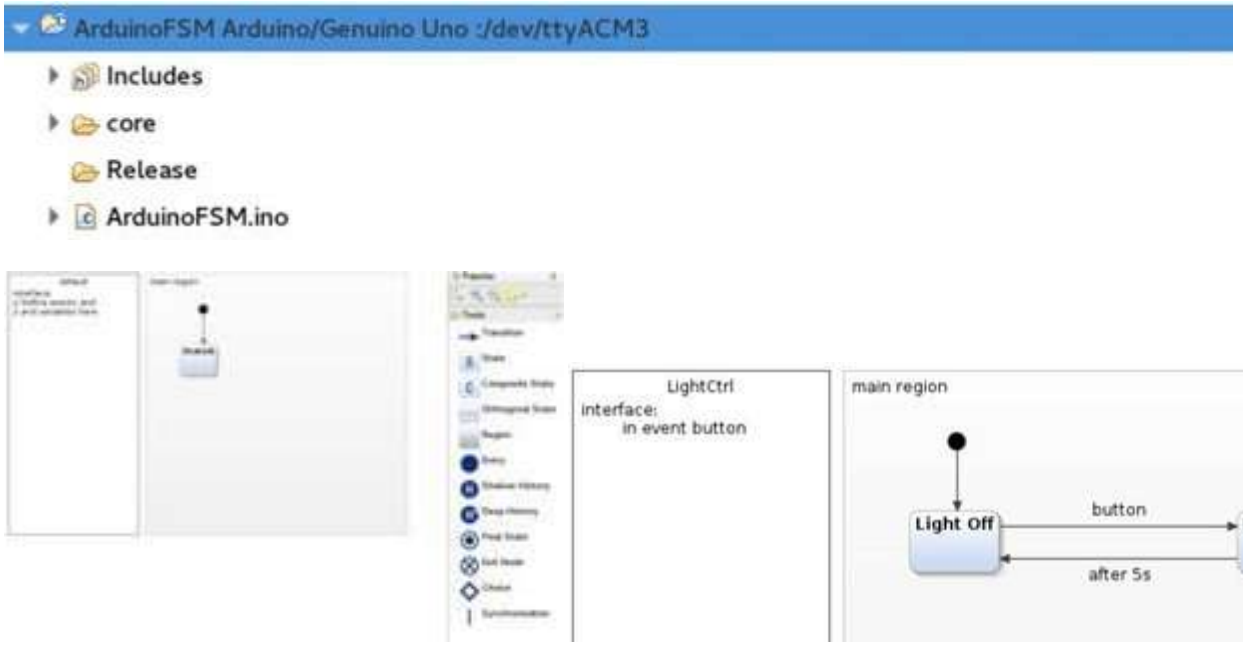


Yakindu SCT is a tool made for exactly that: Modeling your system and generating code from it. The modeling tools are far more advanced than simple finite-state machines, because they are based on the statechart theory by Harel. They extend the normal automaton theory by some further concepts – for example, a history state, where leaving a statechart saves the active state, so you can come back later, and much more. We won't need these extra functions for this 'Ible though.
Yakindu SCT is based on Eclipse, one of the most often used IDEs. So, we can use all the Eclipse plugins on the market and have an already known environment. And it is open source which means it is for free! At first, go to statecharts.org and select „Download SCT". You will need to put in your name, your email and your profession. After you downloaded the tool, just unzip it (right click -> Extract All, or similar). Inside, you will find „SCT". Start it. (No, a real installation is not needed.)

After you installed Yakindu SCT, you have the tools to model a FSM, but we will want to get the code to work on an Arduino. There is an excellent plugin for Eclipse to do that, you can find more about it on http://www.baeyens.it/eclipse/. It gives you the full Arduino toolchain inside of Eclipse, so you have the ease of use of the Arduino IDE as well as the intelligent code management and coding assistants of Eclipse. In SCT, go to *Help -> Install new Software*. The install wizard opens. Click on the *Add...* button near the upper right corner of the wizard. A dialog opens, asking you to specify the update repository you want to install the new software from. Enter some text into the Name field. This text is abitrary in principle, but you should choose something that makes it easier for you to identify this particular update repository among other update repositories. After entering name and location (http://www.baeyens.it/eclipse/V3/) of the update repository, click OK. Eclipse establishes a network connection to the update repository, asks it for available software items and shows them in the install wizard. Here, you simply accept the choice „Arduino". Clicking Next a few more times and accepting the license agreements later, it will ask you to restart the tool. After you have done that, the plugin downloads all the needed libraries, so you don't have to copy them from an existing Arduino project. And here you go, having the Arduino tools installed in your Yakindu SCT installation. Now it is time to combine the possibilities of both.

Note: If you are on Windows and haven't already, install the official Arduino IDE as well. It comes with the required drivers. I'm not sure about the situation on Mac. Linux already contains the drivers, so an installation of the Arduino IDE is not neccessary.

Step 4: Start creating a statechart



We will now start to model the statechart together. At first, we will create a new project. You should be on the welcome page of SCT / Eclipse. *Go to File -> New -> Project...* and choose *Arduino -> New Arduino Sketch* in the main menu. The normal wizard for new Eclipse projects will appear. You have to give your project a name. Let's name it ArduinoFSM. In the next window, you can specify the port your arduino is connected to. If you don't know it and don't know how to find out, ignore this. You can now click Finish.
If you instead selected *New -> Arduino Sketch*, you will not be asked where your arduino is connected. Use *Project -> Properties* to do that later then. If you don't know how to figure out your Arduino's port, the last step of this instructable will help you out.

In case the welcome screen does not close after you created the project, just close it on your own, using the X in the tab. You should now have something similar to the first picture in the Project Explorer on the left.

We will now want to create a new folder called „model". Right click your project and select *New -> Folder*. Type in the name and click Finish.
Right click that new folder, go to New again. Depending on your installation, you might be able to directly add a new Statechart Model, or maybe you have to use Other, select Yakindu, and choose Statechart Model. What you have now should look like the second picture: One entry state and one generic first state named **StateA**.

The text box on the left allows you to declare events and variables related to the statechart, while the area on the right is the graphical statechart editor.
We will need one event: the pushbutton. Double-click the textbox on the left, and under interface, insert the text

```
in event button
```

With that, you declare that there is an incoming event named „button". Also, double click the word „default" in that text box, and give the statechart a better name – how about „LightCtrl"? Now, add another state: just click on *State* in the palette on the right, and then somewhere in the graphical statechart editor. Double click on both states' names, and name the one with the black entry state attached to it **Light Off**, and the new state **Light On**. Now, we need the transitions: Select Transition from the palette, click on one state,

hold, and drag to the other one. This should form the transition. It goes from the state you clicked first to the second state. Add the second transition by clicking the state you dragged to first now and drag to the other one, so that you have transitions in both directions. Now, click on a transition. A text field will appear. Here, you can input the events and outputs that you want to give that transition. On the transition from **Light Off** to **Light On**, type *button*, on the other one, type *after 5s* (that's faster than 30 seconds for testing). You should now have something that looks like the third picture!

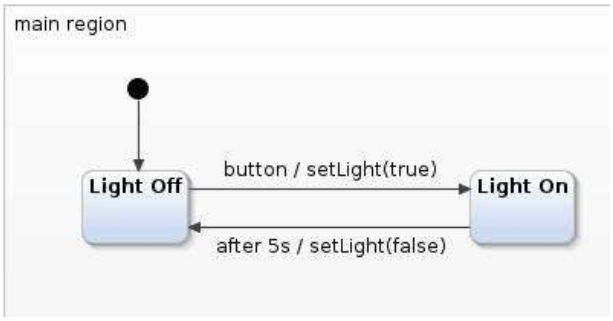
That's it for now. You have a working model of a staircase light!

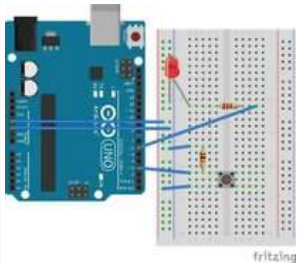
Step 5: Simulate the statechart

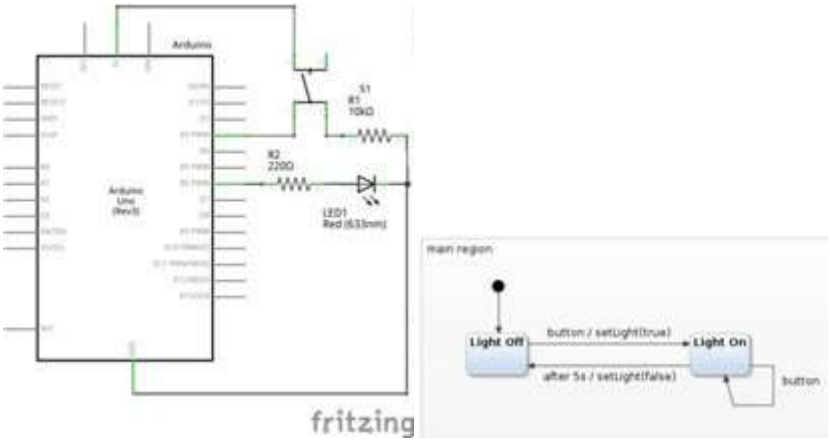
Another nice feature of Yakindu SCT is that you can simulate the statechart without writing any code beforehand. You can try if your state machine does exactly what you wanted it to do.

Simulating a statechart is extremely easy. Right click the .sct-file in Eclipse / SCT, select *Run As* and then *Statechart Simulation*.
A new perspective opens. You should be able to see that the first state is red, this is the active state. (Take a look at the picture) On the right, there should be the Simulation View open. You can simulate the button push event by clicking the word *button* in the simulation view on the bottom right. The active state should change from Light Off to Light On. After five seconds, or after you click on the time event *Light_On_timer_event_0*, the active state changes back to **Light Off**. Fantastic! Now let's check how to get this to work on an Arduino.

Step 6: Bring your system to the real world







Okay, we clicked around, used a graphical editor (normally associated with low-level-languages), let's get that thing to life. First, we need a code generator that transforms our statechart into C Code. That's surprisingly simple, though it might look like black magic at first.

Right click your model-folder and select *New -> Code Generator Model*. Click yourself through the wizard, and attach the code generator to the statechart you made before. Attention: In that same window, there is a selector in the top that is easily overseen. Use it to select the C Code generator instead of the Java one, and click finish after you have checked the checkbox next to your statechart. Normally, the generator should now work directly and automatically all the time. Check if two folders were created, src and src-gen. If that is not the case, go to Project in the main menu and check if Build automatically is activated. Do so if it isn't, and right click your project and select Build Project. A progress bar should appear as well as both the mentioned folders. When you made any changes, you can also right-click the generator file and select *Generate Code Artifacts*.

The content of the folder src-gen is quite interesting. The file *LightCtrl.c* contains the implementation of the statechart. When you inspect it, you will find a function *LightCtrlIface_raise_button(LightCtrl* handle)*. You can call this function to raise the button-event we declared earlier – for example, when you check your hardware button's pin and see it has a HIGH-level. Then there is the file *LightCtrlRequired.h*, at which you need to take a look. It declares functions you need to implement. For this statechart, there's only two functions: *lightCtrl_setTimer* and *lightCtrl_unsetTimer*. We need these functions because our statechart uses the construct *after 5s*. This is a pretty convenient function, but our statechart code generator does not deliver a timing service, because that is highly platform dependent – your computer can handle timers differently than the tiny Arduino, and timer handling on Mac & Linux works differently than on Windows.

Luckily, I will give you a timing service, so you don't need to implement your own. In your project, create a new folder, let's call it *scutils* for **s**tate**c**hart **util**ities. You can name it whatever you want or choose not to create that folder, it's just a matter of organization. We will create two files in there, *sc_timer_service.c* and *sc_timer_service.h*. Copy the

code from GitHub inside there:

sc_timer_service.h

sc_timer_service.c

With YAKINDU SCT 2.7.0, there is a new option to obtain the project for this instructable:

In SCT, go to File -> New -> Example..., select "YAKINDU Statechart Examples", and click next. In the new Example Wizard, click "Download" to obtain the newest set of examples as indicated. Select "Basic Finite State Machine For Arduino" from the arduino category, and click Finish. The project will be copied into your workspace. Right-click it and hit 'Refresh' - just to be sure.

Now, we can start to work on the Arduino code in the *.ino-file the wizard generated.

Additionally to the *Arduino.h*, also include *avr/sleep.h*, and of course our state machine and the timer service: *LightCtrl.h*, *LightCtrlRequired.h* and *sc_timer_service.h*. Now, the regular Arduino stuff is needed: we define the pins for the button and the LED, and set these up inside of the setup-function (that's what it was made for). Then, we need to define the functions the statechart expects us to define - *lightCtrl_setTimer* and *lightCtrl_unsetTimer*, as explained earlier. Here, we just use the timer service, and we are done. Now, we should spare a thought on how we actually want to activate the LED when we reach the state **Light On**. Basically, we have three options:

1. We can check if the statemachine is in the state Light On, and activate / deactivate the LED based on that information
2. We can go to our statechart and set a variable when we reach the states, that we can poll

3. We could add an operation that manages the light that is called by the statechart on a transition.

The first solution is really bad. We would have logic concerning the statechart outside of it. If we would rename our states, it would cease to work correctly; but those names are meant to be prosaic and not logic related. Using variables is okay, especially when working with desktop applications. We could sync with them every x milliseconds or so. Here, we want to use an operation. Add the following to the statechart's interface declaration:

```
operation setLight(LightOn: boolean): void<br>
```

This declares a function that accepts a boolean value as argument and returns nothing (void). This should not be new for you, only the syntax here is different. Remember – statecharts are not bound to a specific language, so the syntax is generic. This function appears in *LightCtrlRequired.h* automatically. If it does not, save your statechart, right click on your project and build it.

The function declared here looks like this:

```
extern void lightCtrlIface_setLight(const LightCtrl* handle, const sc_boolean
lightOn);
```

The input parameter handle is of the type LightCtrl, it is the referrer to the statechart. If you are not that experienced in C: the star denotes a so-called pointer, so the variable contains the address of the statechart variable. This helps us because we can operate on the original object and don't have to create a copy of it. So, let's implement this function:

```
void lightCtrlIface_setLight(const LightCtrl* handle, const sc_boolean lightOn) {<br>
if(lightOn)
        digitalWrite(LED_PIN, HIGH);
    else
        digitalWrite(LED_PIN, LOW);
}
```

As you can see, this function is bloody simple – we don't even use the handle to the statechart, we only write a HIGH on the LED pin if the operation's argument is true, and LOW otherwise.

We change the statechart itself so that it looks like in the first picture.

Remember step 1? Left to the slash is the needed input for the transition, right is the output of the state machine if this transition is taken. The output here is to call the specified operation with these arguments.

```
#include "Arduino.h"<br>#include "avr/sleep.h"
#include "src-gen/LightCtrl.h"
#include "src-gen/LightCtrlRequired.h"
#include "scutil/sc_timer_service.h"<br>
#define BUTTON_PIN 3
#define LED_PIN 6
#define MAX_TIMERS 20 //number of timers our timer service can use
#define CYCLE_PERIOD 10 //number of milliseconds that pass between each statechart
cycle<br>
static unsigned long cycle_count = 0L; //number of passed cycles
static unsigned long last_cycle_time = 0L; //timestamp of last cycle
static LightCtrl lightctrl;
static sc_timer_service_t timer_service;
static sc_timer_t timers[MAX_TIMERS];<br>
//! callback implementation for the setting up time events
void lightCtrl_setTimer(LightCtrl* handle, const sc_eventid evid, const sc_integer
time_ms, const sc_boolean periodic){
    sc_timer_start(&timer_service, (void*) handle, evid, time_ms, periodic);
```

```
}<br>
//! callback implementation for canceling time events.
void lightCtrl_unsetTimer(LightCtrl* handle, const sc_eventid evid) {
    sc_timer_cancel(&timer_service, evid);
}<br>
void lightCtrlIface_setLight(const LightCtrl* handle, const sc_boolean lightOn) {
    if(lightOn)
        digitalWrite(LED_PIN, HIGH);
    else
        digitalWrite(LED_PIN, LOW);
}<br>
//The setup function is called once at startup of the sketch
void setup()
{
    pinMode(BUTTON_PIN, INPUT);
    pinMode(LED_PIN, OUTPUT);<br>
    sc_timer_service_init(
            &timer_service,
            timers,
            MAX_TIMERS,
            (sc_raise_time_event_fp) &lightCtrl_raiseTimeEvent
            );<br>
    lightCtrl_init(&lightctrl); //initialize statechart
    lightCtrl_enter(&lightctrl); //enter the statechart
}<br>
// The loop function is called in an endless loop
void loop()
{
    unsigned long current_millies = millis();<br>
    if(digitalRead(BUTTON_PIN))
        lightCtrlIface_raise_button(&lightctrl);<br>
    if ( cycle_count == 0L || (current_millies >= last_cycle_time + CYCLE_PERIOD) ) {
            sc_timer_service_proceed(&timer_service, current_millies -
last_cycle_time);
            lightCtrl_runCycle(&lightctrl);
            last_cycle_time = current_millies;
            cycle_count++;
        }<br><p>}<br></p>
```

Also, check out the code [in this gist with line numbers](#).

- Lines 1-6 contain includes as discussed earlier.
- Lines 8 and 9 define hardware pins we will want to use for our arduino.
- Lines 11 and 12 define how many timers our statechart can use, and how many milliseconds should pass between each computing cycle of the statechart.
- Lines 15 and 16 declare variables that we can use to count the cycles and to manage the time of the last cycle.
- Lines 17, 19 and 21 declare important variables for the usage of the statechart: The statechart itself, the timer service, and an array of timers.
- Lines 24 and 33 define functions that the statechart needs for the timer, and line 33 is the function to set the LED we discussed earlier.
- In line 41, void setup() is a standard function of the Arduino. It's called once at startup. We use it to initialize stuff – our LED and button pins get their direction configured (INPUT is standard, we do that for clarity), the timer service is initialized, the statechart is initialized and entered. Entering means to start the state machine, so the first state is activated – this is the state the entry state points at. So, at startup, the light is off.
- In line 59, the loop function follows, which is called all the time by the Arduino.
- In line 61, we capture the current time with the millis() function, which is defined by the Arduino library.
- In line 63, we check if our button is pressed, and raise the button-event if it is.
- In line 66, we check if more than CYCLE_PERIOD milliseconds passed since we last cycled our statechart.
- This takes some load from our arduino and means we can reliably use up to 10 milliseconds for our own functions.

- In line 68, we tell the timer service how much time has passed since the last invocation, tell the statechart to run one cycle in line 70, save the current time in line 72, and increment the cycle count in line 73.

Using the arduino plugin, you can now connect the arduino with the LED and the button attached to your computer, and use the button in the top toolbar to upload the program to your arduino.

The circuit is shown in pictures two and three.

The LED is connected to a digital pin (6) with a resistor of roundabout 200 Ohms. The cathode is attached to GND.

Pushbuttons have four pins, please check which of these are always connected and which are connected when you push the button. Then, you attach the digital pin (3 is used here) to one side, and a pulldown resistor to GND. This stops the pin from „floating“, an undefined state, and keeps it on 0 Volts. When the button is pressed and the other side is attached to VCC, that side is „stronger“ because it has no resistor and the voltage goes up to 5 Volts – basically a voltage divider where one resistor is 0 Ohms. Please use a fairly high resistor here, because it limits the current going through the button. 1 kR is the minimum.

As you can see, this program’s logic is completely independent from the actual size of our statechart. It does not matter if our statechart has 2 or 20 states – of course, if we want to do something, we need to implement a function here and there. But the main code inside of void loop() always stays pretty small and allows for a modular program architecture. We only have to take care of the interfaces from the statechart to our Arduino's hardware inside of our code, the auto-generated statechart handles its internal logic. Remember how we discussed to reset the timer when the button is pressed again? You could now add a transition from the Light On state to itself with „button“ as guard event, and you would not need to change or add a single line in your code. Try it out!

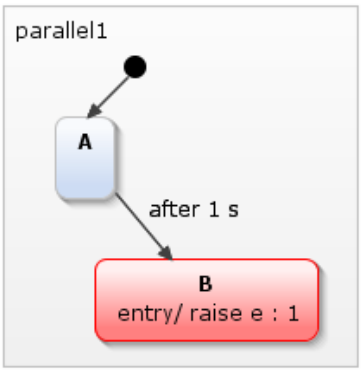
Step 7: Additionally: Find out your Arduino's port

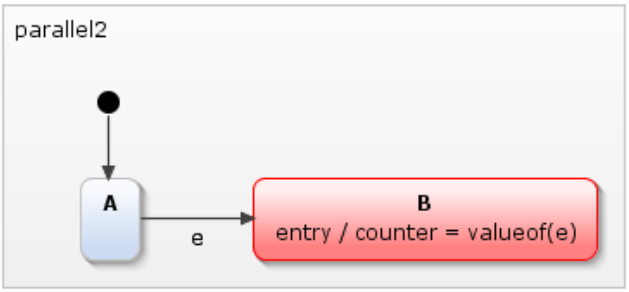
So, you got stuck because you can't figure out which serial / USB-port your Arduino is connected to. Alright, you'll find instructions below for Windows and Linux.

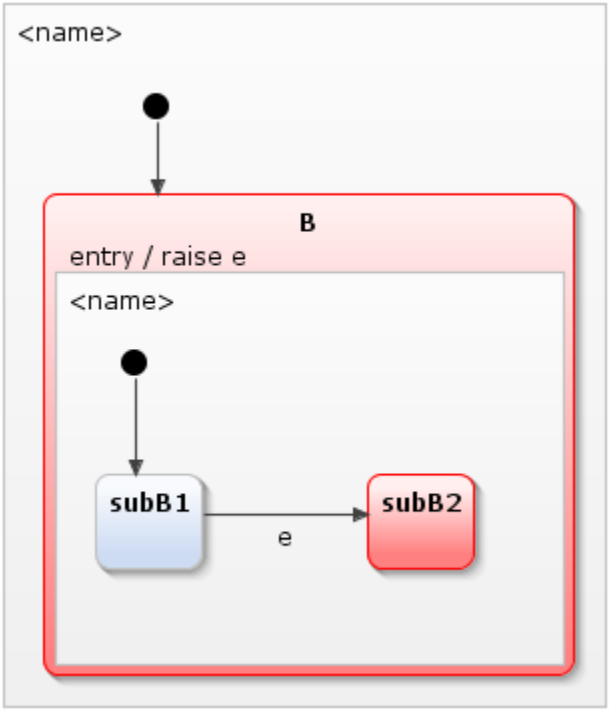
Windows

Plug the arduino in your computer, go to "Devices and Printers" (from the start menu or the system control panel). Your arduino should show up there, as you can see in the image - for me, the port would be COM12. This can change, for example when you use another USB port, reboot your system... If something does not work, check if this is still correct.


default
internal:
event e : integer
var counter : integer


parallel1
A
after 1 s
B
entry/ raise e : 1


parallel2
A
e
B
entry / counter = valueof(e)


<name>
B
entry / raise e
<name>
subB1
e
subB2