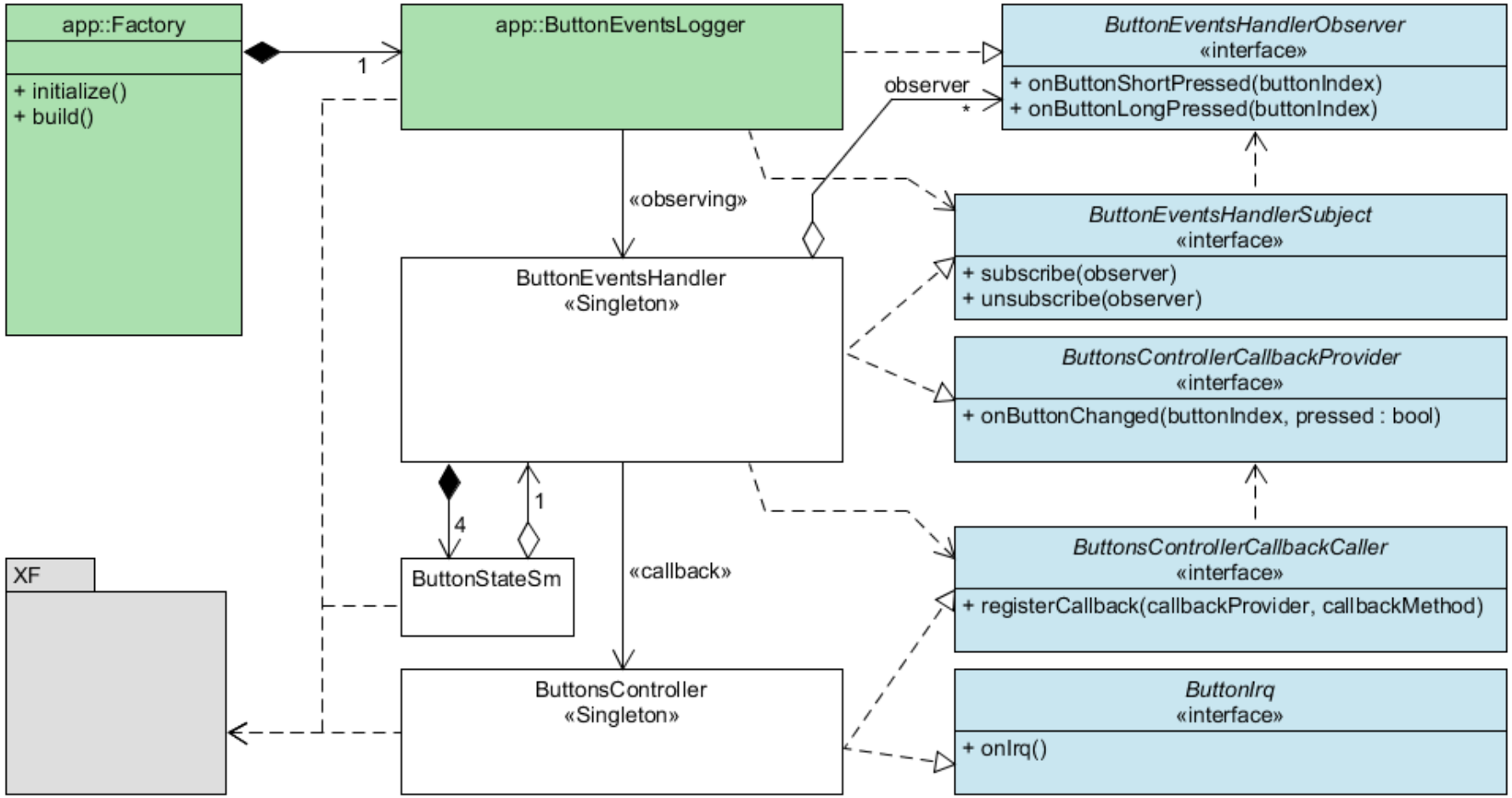
Button Manager – Laboratory Guide

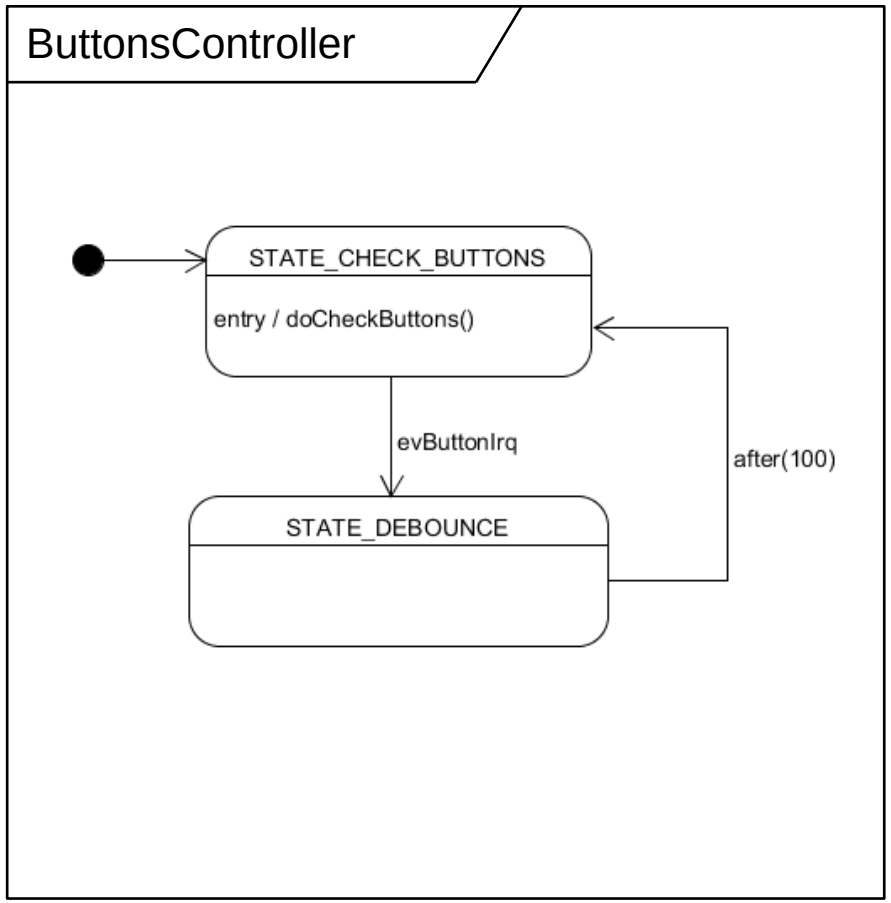
Medard Rieder
Thomas Sterren

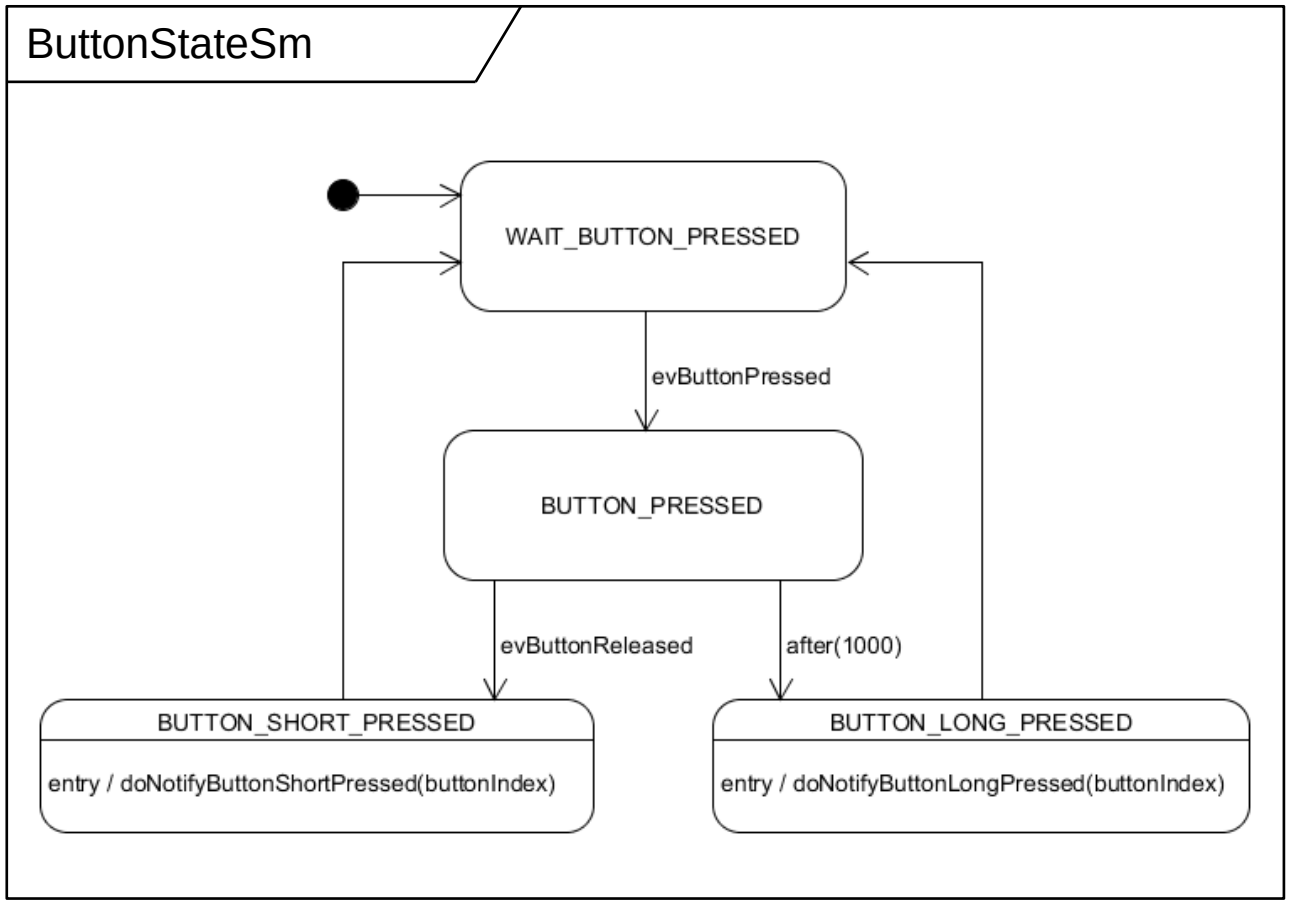
v22.01

ButtonManager Class Diagram



State-Machine Diagrams

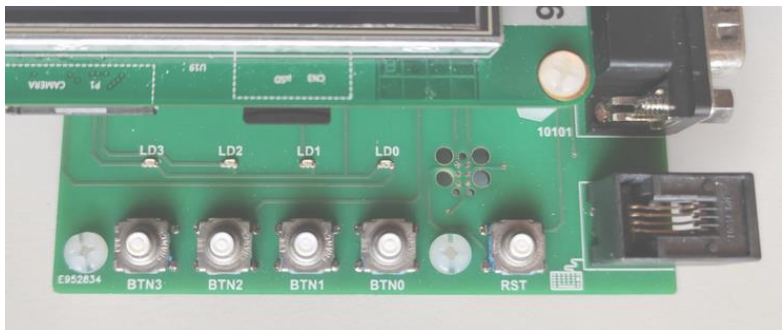


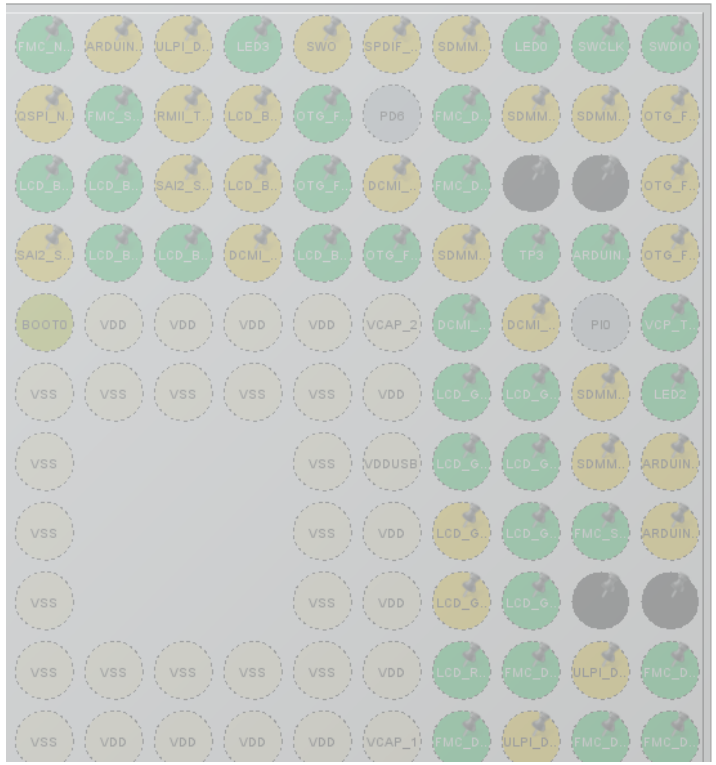


Open STM32CubeIDE Peripheral Configuration

- In the STM32CubeIDE project double-click on the ***ButtonManager.ioc*** file
- Configure the 4 GPIO for the buttons on the extension board:

GPIO	User Label
PI2	BUTTON0
PI3	BUTTON1
PG7	BUTTON2
PG6	BUTTON3





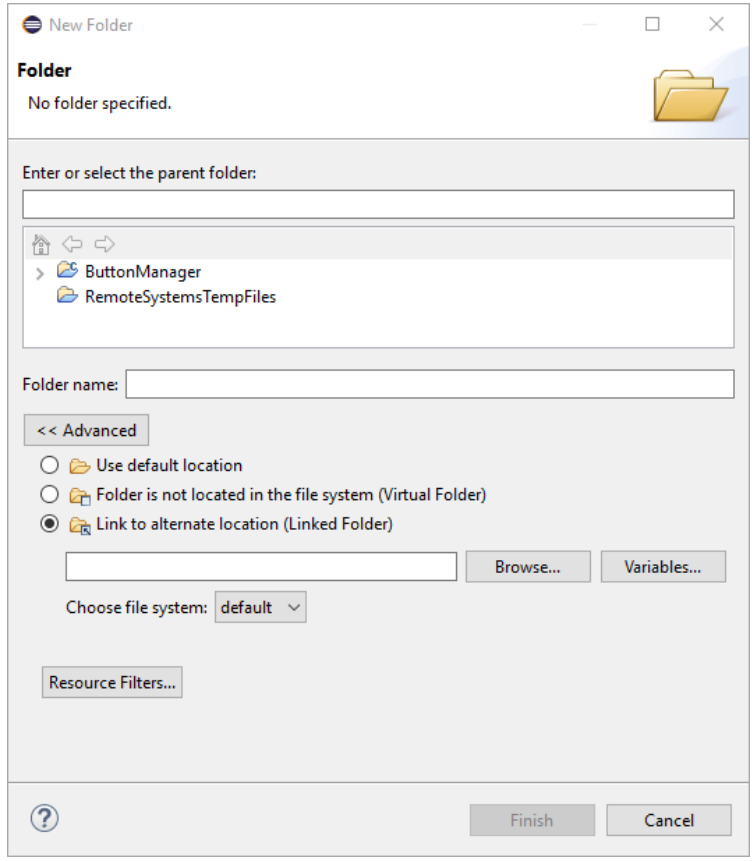
Add Packages from *src* Folder

Add following *src* folders to the Eclipse project:

- *app*
- *mdw*
- *platform*

Create a ‘linked folder’ to add a folder outside of the project

Use variable PROJECT_LOC to change absolute path to a relative path



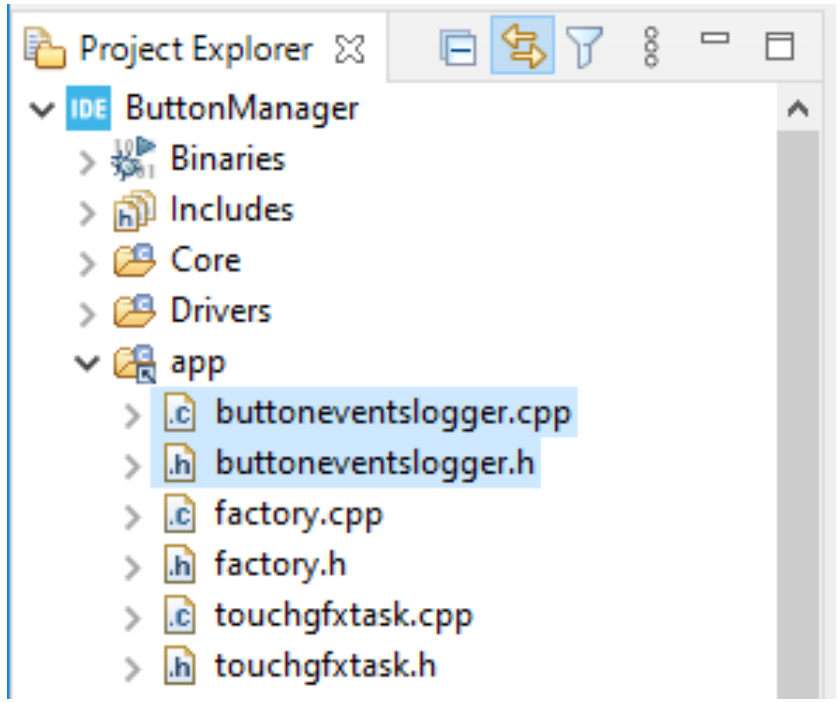
Start Coding – Add *app* Classes

Now you are ready to start development. Its on you to provide the missing classes as specified in the laboratory document:

Add the following class to the *app* package:

- `ButtonEventsLogger`

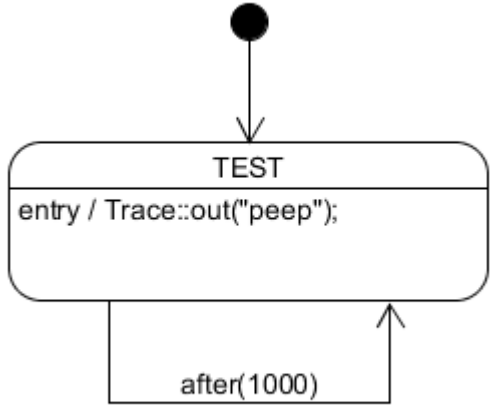
Implement `ButtonEventsHandlerObserver` interface on `ButtonEventsLogger`



Add State-Machine to `ButtonEventsLogger`

Check using a state-machine that the `XF` is working correctly

Check that the `Trace` functionality is working correctly



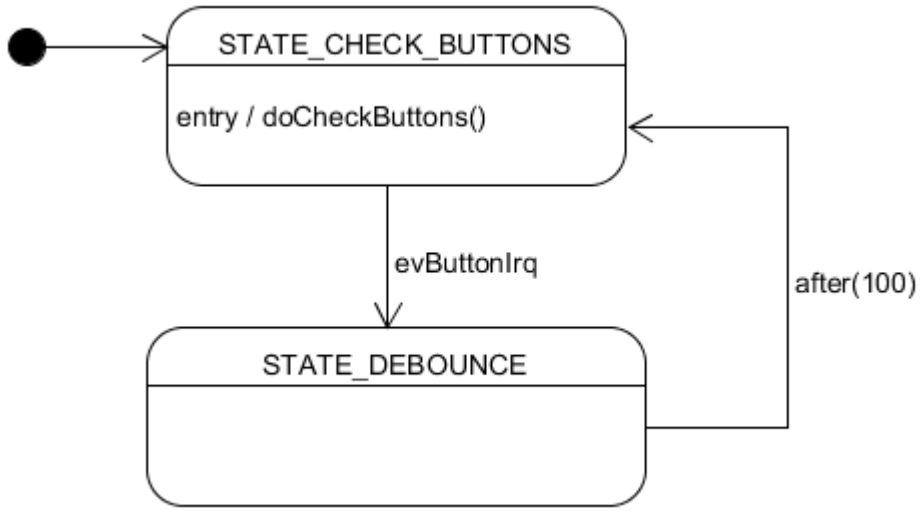
Add `ButtonsController` Class

Add `ButtonsController` class in 'platform/f7-disco-gcc/board/‘

Implement `ButtonIrq` interface on `ButtonsController`

Implement `ButtonsControllerCallbackCaller` interface on `ButtonsController`

Implement state-machine for `ButtonsController`



Add ‘mdw/button’ Package Classes

Add `ButtonEventsHandler` class to 'mdw/button/‘

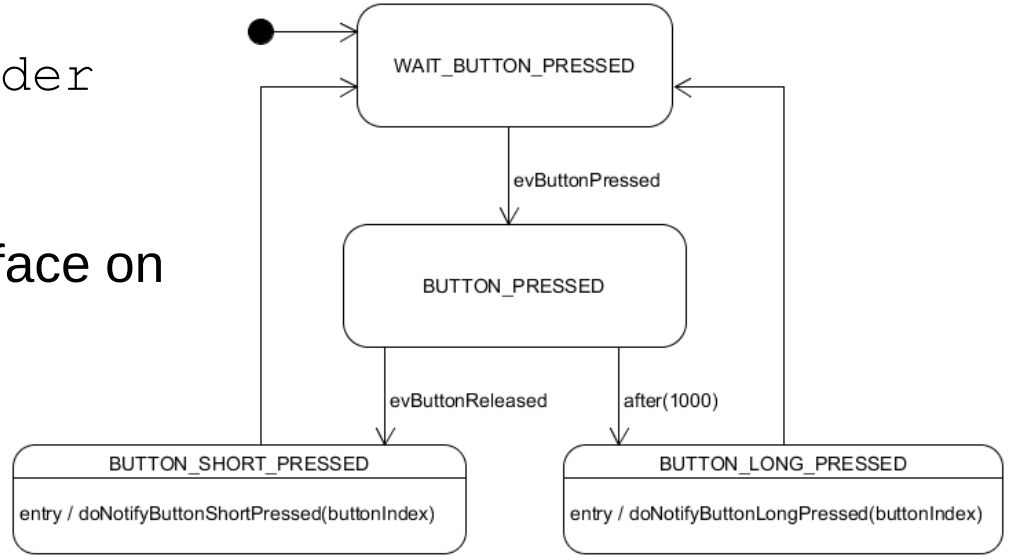
Implement `ButtonsControllerCallbackProvider` interface on `ButtonEventsHandler`

Implement `ButtonEventsHandlerSubject` interface on `ButtonEventsHandler`

Add `ButtonStateSm` class to 'mdw/button/‘

Implement state-machine for `ButtonStateSm`

Add `ButtonStateSm` instances to `ButtonEventsHandler`



Decouple calls to `ButtonEventsLogger::on…()` Methods

The underlying `ButtonEventsHandler` class must not call the application code located in the `ButtonEventsLogger` class.

This can be achieved by generating events inside of the `ButtonEventsLogger::on…()` methods.

The internal events are causing finally the `XF` to asynchronously execute the application code of the `ButtonEventsLogger`.