GOLDi - Step by Step

Pierre Helbing

September 2020

1 How to use ECP

Step 1:
At first you have to create an account or sign in with an existing one, before you can start an experiment.

Figure 1: Login interface

Before you start your selected experiment, you have to prepare the according files for the Physical System. You can find more information on how to adapt the files from the Physical Systems at the tap **Control Unit**. The available Control Units are:

- Manual Control
- Digital Demo Board
- PLD
- Finite State Machine

- MicroController

Step 2:
1. Click on the tab Start Experiment
2. Select a location:

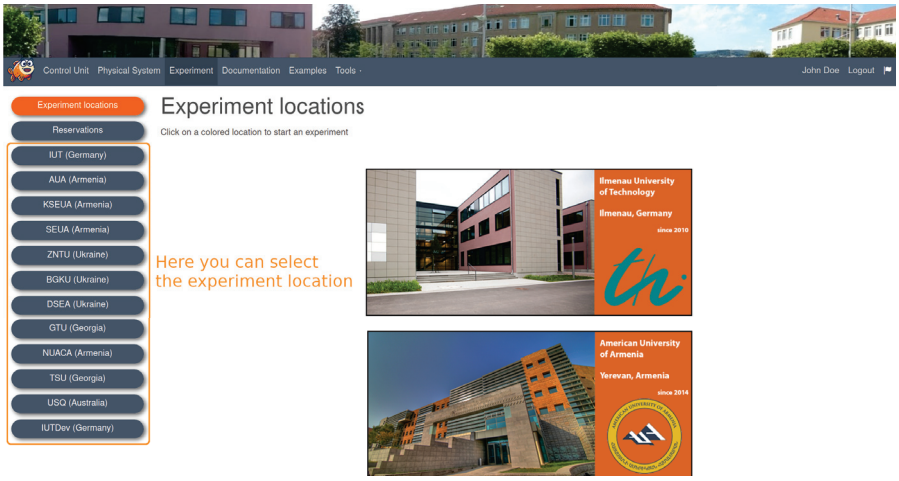


Figure 2: Select a location

3. Select the Physical System and the Control Unit which you want to use:

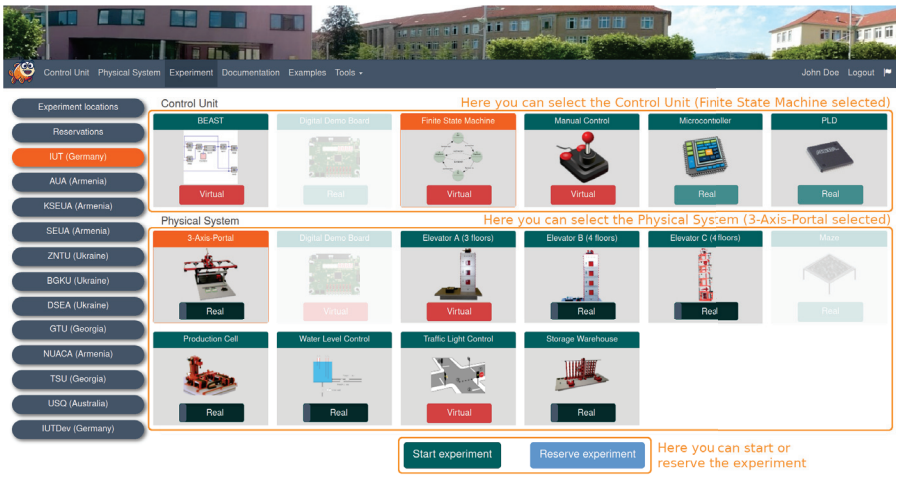


Figure 3: Select the Physical System and Control Unit

- You can select between **Real** and **Virtual** when you choose the Physical System and Control Unit.
- When you choose **Real** you can try out your experiment on the real system.
- When you choose **Virtual** you can try out your experiment on a simulation.
- You can combine each Physical System with each Control Unit except the digital demo board. You can combine the digital demo board only with the digital demo board.

4. When you selected the Physical System and Control Unit, you can click on the button **Start experiment** at the end of the page

Step 3:
Now the experiment will be opened in a new tab.

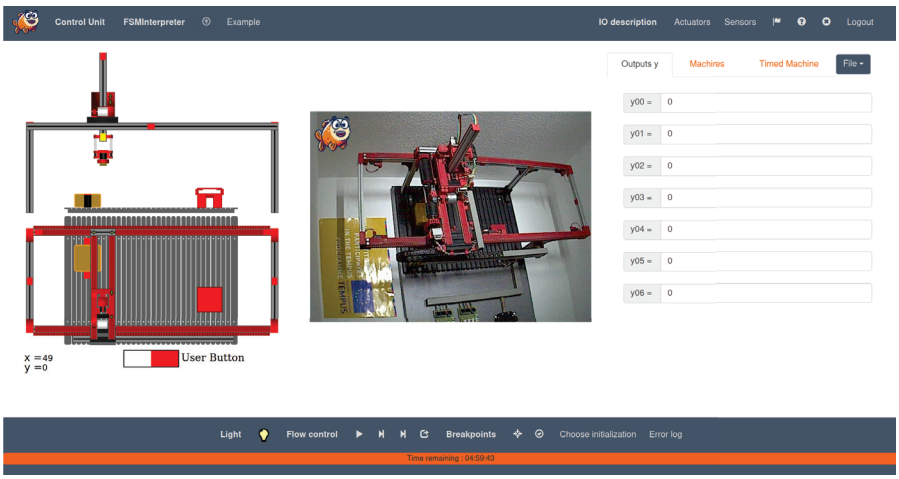


Figure 4: Experiment interface

The experiment interface is explained below:
There is are toolbars on the top and on the end of the site. In the top toolbar of the site you can find buttons to change settings, to load files and to show the IO description.

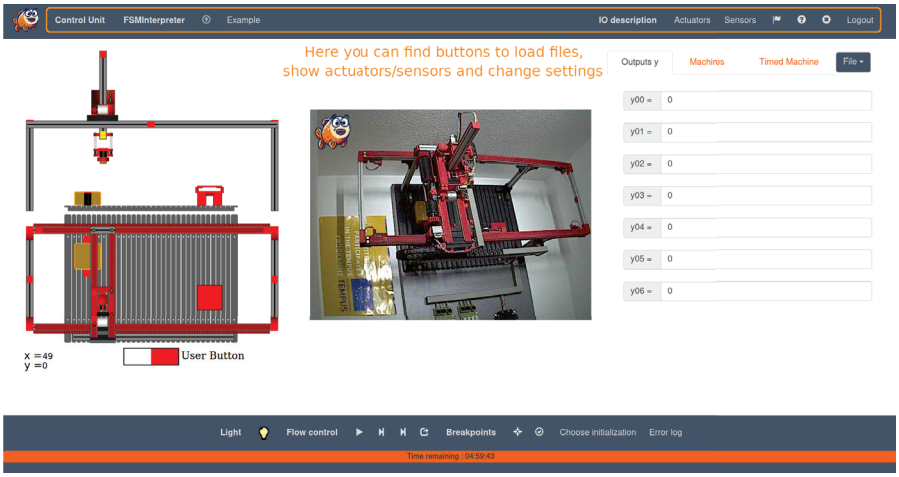


Figure 5: Top toolbar

	With this button you can upload your stored file. Notice the file format for each Control Unit
Example	With this button you can load a defined example for the selected Control Unit
Open WIDE	With this button you can open the Integrated Development Environment of GOLDi (only available for Microcontroller and PLD)
Actuators	Click on this button to see an overview of the actuators. The actuator-names are either green or red. Green indicates an actuator value of 1. Red indicates an actuator value of 0. When you go over one of the listed actuators with your mouse the according actuator will be highlighted in the virtual system.
Sensors	Click on this button to see an overview of the sensors. The sensor-names are either green or red. Green indicates a sensor value of 1. Red indicates a sensor value of 0. When you go over one of the listed sensors with your mouse the according sensor will be highlighted in the virtual system.
	With this button you can change the language of the whole website. It is possible, that not everything will be translated into your selected language. These parts will remain in English.
	With this button you can open a help-guide.
	Click on this button to close the current experiment.
Logout	Click on this button and you will be logged out of GOLDi-website. The experiment will be closed as well.

Table 1: Top toolbar buttons

In the middle of the page you can find the simulation of the Physical System on the left, the real model of the Physical System in the middle and the control settings (here: equations) for the experiment on the right.

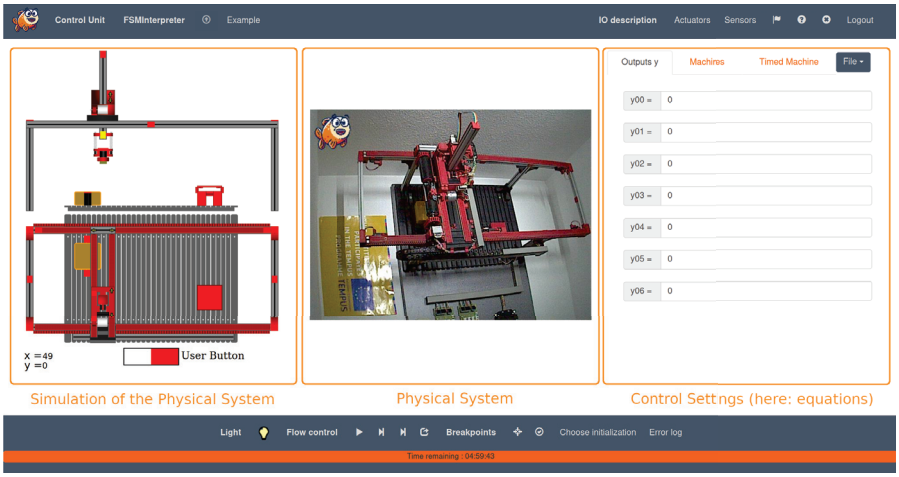


Figure 6: Experiment overview

In the bottom toolbar you can find buttons to control the experiment for the Physical System.

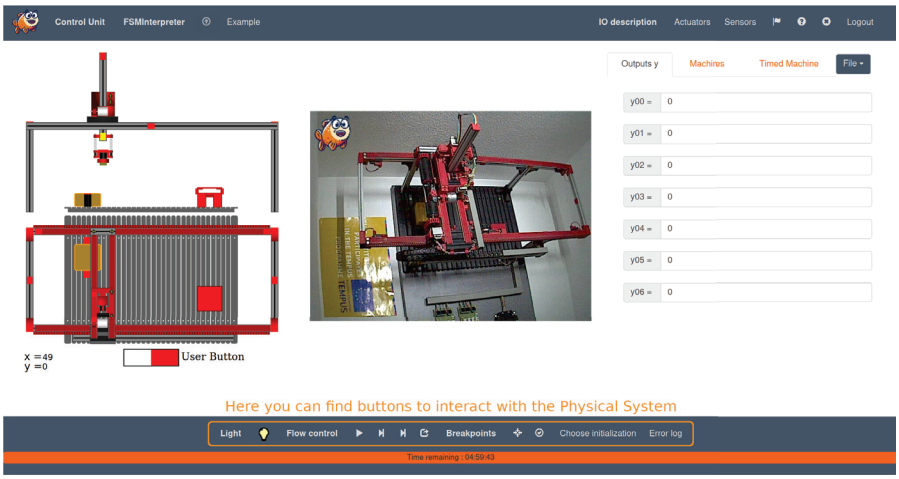


Figure 7: Bottom toolbar

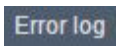
	With this button you can switch on and off the light from the real system.
	With this button you can conduct your uploaded program on the Physical System.
	With this button the simulation goes step for step forward through your uploaded program. It stops every time when it reaches a sensor.
	With this button the simulation goes step for step forward through your uploaded program. It stops every time when it reaches an actuator.
	With this button you can reset the controller (FPGA or MicroController).
	With this button you can set breakpoints at which the program stops every time when the system reaches them.
	With this button you can activate or deactivate your chosen breakpoints.
	With this button you can choose the initial state for the Physical System. Each Physical System has its own set of possible initial states.
	Click on this button and you will see an overview with the occurred errors. There are two different types of errors which can occur: • Control Error: Error based on the design of your experiment • Time Out: Occurs when the system does not react for a certain time. This can happen if your internet is too slow or your design has got a looping error.

Table 2: Bottom toolbar buttons

2 Manual Control

With the **Manual Control** it is possible to control the Physical Systems by clicking on buttons. You can use these buttons to control the movement of system elements in different directions and to control several specific features of a system.

2.1 Example for the Physical System 3-Axis-Portal

If you choose the 3-Axis-Portal as the Physical System, you can see the animation of the system on the left side and the control buttons on the right side. The portal robot can be moved on the x-axis and on the y-axis, its electromagnetic gripper can be moved in z-direction. Furthermore the magnet of the gripper can be switched on and off. There are control buttons for every possible movement. If they have been clicked, you can see the corresponding movement in the animation.

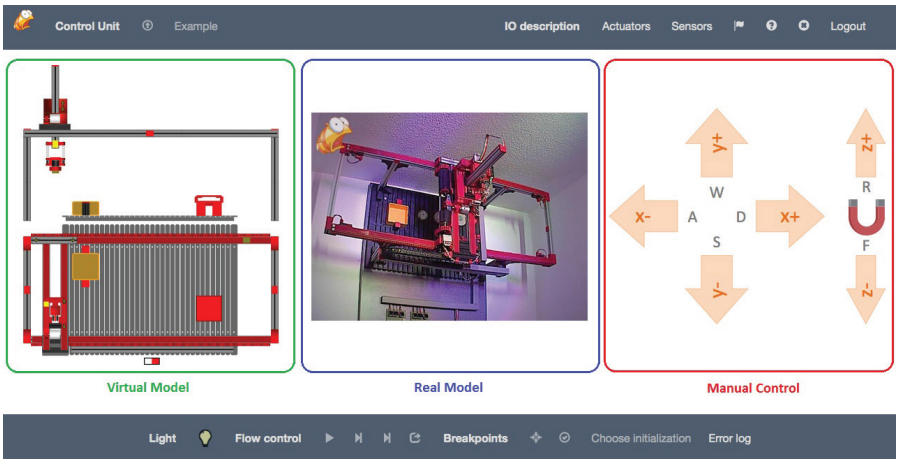


Figure 8: Manual Control of the 3-Axis-Portal

Instead of clicking on a control button, you can also use your keyboard. The movement in x- and y-direction corresponds to the keys "W-A-S-D", the z-direction corresponds to the keys "R-F" and the magnet can be switched on and off by the key "E" (see figure 2 below).

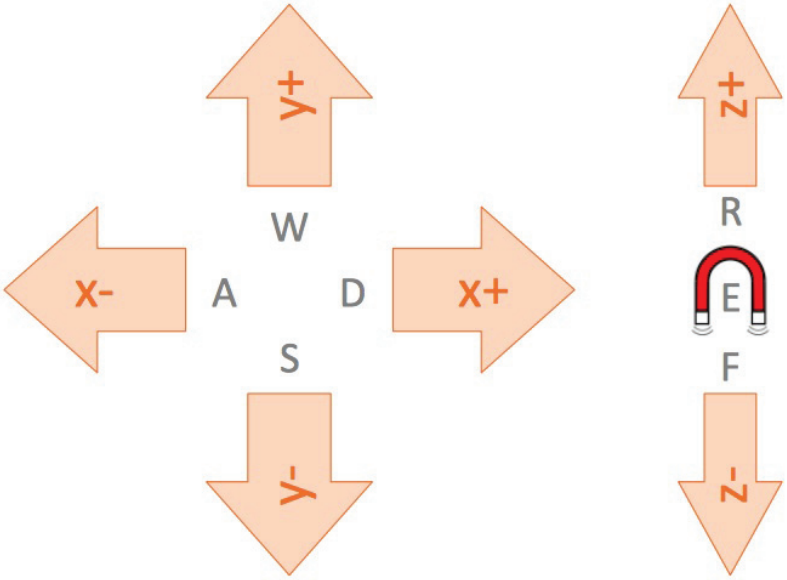


Figure 9: Control Buttons and corresponding keys

3 Digital Demo Board

The **Digital Demo Board** is a special rapid prototyping board, which was developed for educational purposes. Our rapid prototyping board is based on the MAX V - 5M1270Z CPLD from ALTERA. Furthermore it consists of input buttons, LED outputs and several other compoments. You can use the rapid prototyping board as a standalone or a web-based version.

3.1 Description of the Digital Demo Board

The Digital Demo Board consists of the following components:

- MAX V - 5M1270Z (CPLD from Altera)
- Input buttons: 8 slide switches, 8 push buttons, 2 rotary hex encoder
- LED outputs: 4 7-segment displays, 1 LED bar display
- Other components: 10 MHz crystal oscillator, Frequenzy synthesizer, Piezoelectric oscillator, Incremental encoder, UART (via USB)

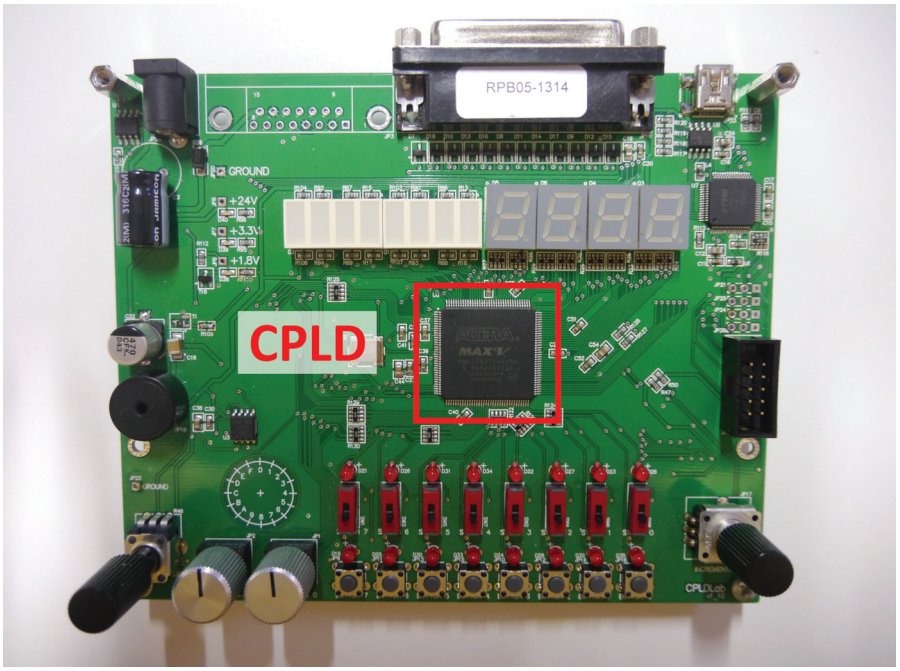


Figure 10: CPLD

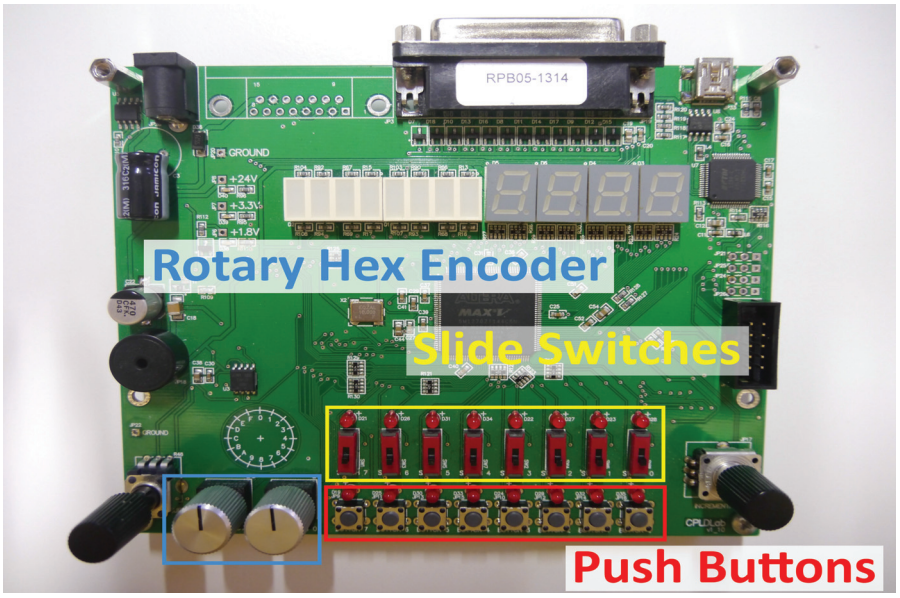


Figure 11: Input buttons

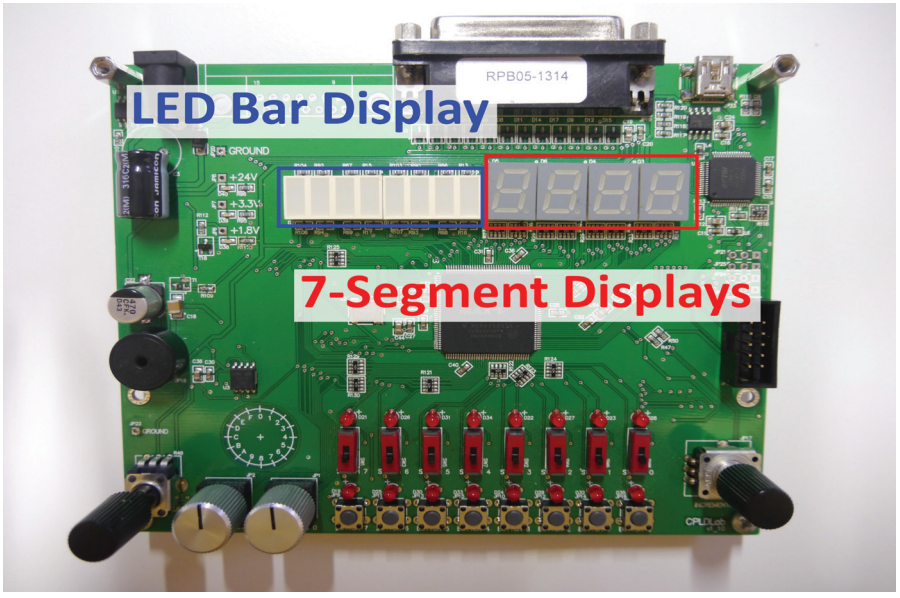


Figure 12: LED outputs

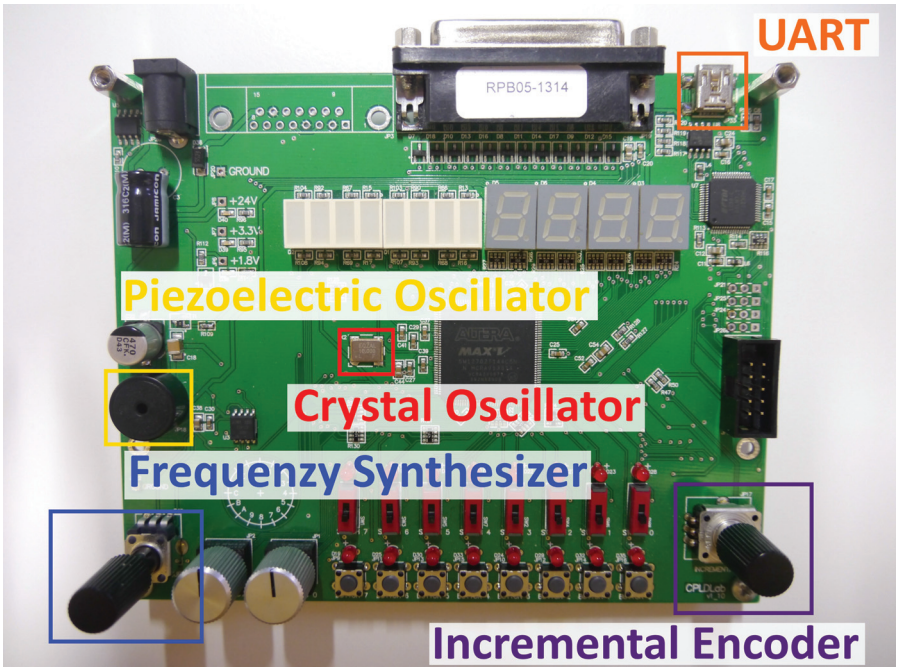


Figure 13: Other components

3.2 Pin assignments of the different Digital Demo Boards

- Pinout-file for Version v1_00: Pinout DDB v100
- Pinout-file for Version v1_10: Pinout DDB v110
- Pinout-file for Version v1_11: Pinout DDB v111
- Pinout-file for Version v1_20: Pinout DDB v120
- Pinout-file for Version v1_21: Pinout DDB v121

Here you can download a summery of all Pin assignments: PDF-file

3.3 Quartus II

To create the files for programming the CPLD on the digital demo board you need the software Quartus II Web-Edition from ALTERA. After creating your own user account at ALTERA's website, it is possible to download the free licence version of Quartus II here:

http://dl.altera.com/?edition=web

Download the latest version of Quartus II (version 15.0) or at least version 12.0 or later. You can decide whether you want to download the Combined Files or the Individual Files. With the Individual Files you only have to download in Device the MAX II, MAX V device support, store all in the same directory. After the download is finished, all will be installed by starting the Quartus Setup.

If you want to get a detailed introduction how to install Quartus II and how to handle the software, you can follow these links:

Quartus II - An Introduction
Quartus II handbook
Altera Software Installation and Licensing

3.4 Standalone version of the Digital Demo Board

The simplest way to use the designed hardware platform is as a stand-alone solution for the rapid prototyping of digital systems (without Internet connectivity). To program the MAX V CPLD on this standalone version of the rapid prototyping board, an additional FTDI chip was placed on the board. You have to connect the prototyping board via USB to your PC or laptop to upload the synthesized CPLD design to the FTDI chip at the prototyping board. This chip will program the CPLD automatically via JTAG interface.

3.5 Web-based version of the Digital Demo Board

With the web-based version of the digital demo board it is possible to upload the synthesized CPLD code of the design to program the CPLD on the rapid prototyping board automatically in the remote lab. Therefore select the Digital Demo Board as Control Unit and as Physical System to start an experiment, so that you can see the web-interface of the board in the ECP. This web-interface allows you to manipulate all the inputs of the rapid prototyping board virtually (slide switches, hex coding switches, pushbuttons and incremental encoder) in the visual model of the prototyping board (on the upper left side). All the inputs are realized as HTML5 control elements and can be activated via a mouse interactively. Changes are immediately sent to the rapid prototyping board in the remote lab and the corresponding results are displayed again inside the visual model. Furthermore a webcam will be used to observe the rapid prototyping board (on the upper right side) to watch the results of the user's actions directly as reaction in the remote lab.

For a web-based usage of the rapid prototyping board, an additional "interconnection" FPGA is placed on the bottom side of the PCB to realize the communication with the remote lab infrastructure. All inputs of the board (buttons, synthesizer, incremental encoder and oscillator) have to be removed from the PCB and are replaced by a direct connection to the outputs of the

"interconnection" FPGA, which will set all input signals according to the user's input via the user interface at the client PC. The generated outputs of the prototyping board can be directly read by the "interconnection" FPGA without removing any LED. The FPGA is interconnected to the BPU within the remote lab infrastructure.

4 Programmable Logic Device

A **programmable logic device (PLD)** is an electronic component for integrated circuits. A PLD has an undefined function at the time of manufacture, so before the PLD can be used in a circuit it must be programmed.

In this case we use one of ALTERA's MAX V CPLDs (MAX V 5M570ZT100C5). The architecture of these CPLDs integrates external functions such as flash, RAM, oscillators and phase-locked loops. Furthermore the CPLD we use delivers 570 logic elements and 100 pins for a particular package.

4.1 Quartus II

To adapt the files for programming the PLD in an experiment you need the software Quartus II Web-Edition from ALTERA. After creating your own user account at ALTERA's website, it is possible to download the free licence version of Quartus II here:

http://dl.altera.com/?edition=web

Download the latest version of Quartus II (version 15.0) or at least version 12.0 or later. You can decide whether you want to download the Combined Files or the Individual Files. With the Individual Files you only have to download in Device the MAX II, MAX V device support, store all in the same directory. After the download is finished, all will be installed by starting the Quartus Setup.

If you want to get a detailed introduction how to install Quartus II and how to handle the software, you can follow these links:

Quartus II - An Introduction
Quartus II handbook
Altera Software Installation and Licensing

4.2 How to get files to start an experiment

To start an experiment with a PLD on GOLDi you have to prepare the following Quartus-projects for your selected physical system. You have the choice either to start with a textual design or with a graphical design in Quartus. Here you can download the example-projects for the physical system "3AxisPortal":

Textual Design: PortalCrane_Mealy
Graphical Design: PortalCrane_Mealy

These projects include all files you need to conduct an experiment. You only have to modify the UserDesign-files and the TopLevelDesign-files to adapt the downloaded project to your experiment. For more information read the following introduction to a quartus design project on the basis of the "PortalCrane-Mealy" example.

4.3 Quartus project example "PortalCrane-Mealy"

In this example we use the graphical design project to demonstrate how to use Quartus II for GOLDi.

Step 1: Creating a project in Quartus II

- Download the rar-archive Graphical Design: PortalCrane_Mealy and extract the files. The created folder "Example_TextualDesign" is the project folder.
- Start Quartus II and click on **Open Project**. Select the qpf-file

  "BPU_ProgrammableLogicDevice.qpf"

  in your project folder and click "Open".

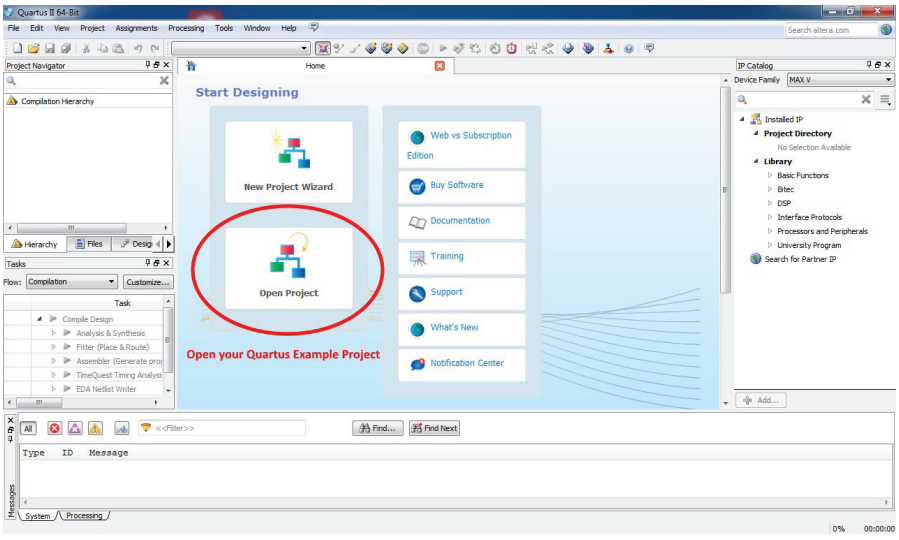


Figure 14: Quartus II user interface

- In the **Project Navigator** you can see the files of the Quartus project. It has mainly the following structure:

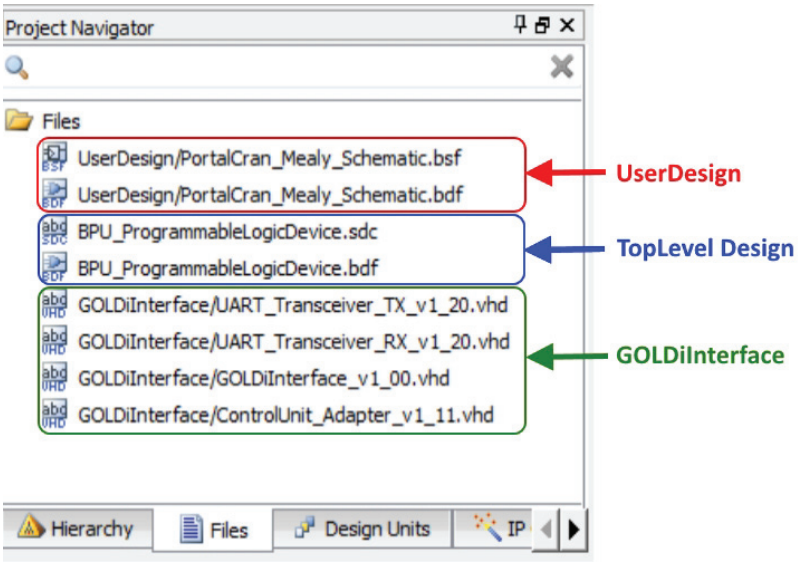


Figure 15: Quartus II project navigator

- The folder **GOLDiInterface** contains all the files which are necessary for the communication within the GOLDi infrastrucuter. These files are ready to use - no modifications are necessary. They will be included automatically.
- The folder **UserDesign** contains the actual control algorithm. It can be used a plain VHDL description (e.g. PortalCrane_Mealy_FSM.vhd), a design as automaton graph by using the integrated FSM graph editor (e.g. PortalCrane_Mealy_FSM.smf) or a schematic design (e.g. PortalCrane_Mealy_Schematic.bdf). In this case we use the schematic design.
- The **TopLevel Design** is the connection between the GOLDiInterface and the UserDesign. It can be implemented as a graphical block design (e.g. BPU_ProgrammableLogicDevice.bdf) or as a plain VHDL description (e.g. BPU_ProgrammableLogicDevice.vhd). In this case we use the graphical block design.

Step 2: Adapt the files to your project
The following files must be adapted according to the actual control algorithm and the selected physical system:

- The **UserDesign** file according the used specification (*.vhd, *.smf or *.bdf)
- The **TopLevelDesign** file BPU_ProgrammableLogicDevice (*.vhd or *.bdf)

Graphical Design

1. **UserDesign file** ./UserDesign/PortalCrane_Mealy_Schematic.bdf
   In this file you have to adapt the schematic design to your control algorithm. The bdf-file generates the corresponding symbol in the file ./UserDesign/PortalCrane_Mealy_Schematic.bsf.

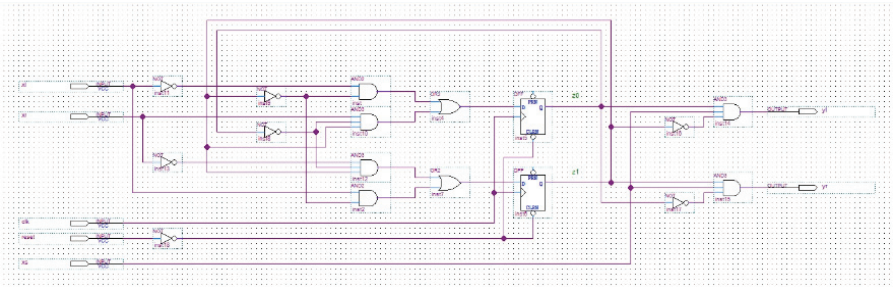


Figure 16: Schematic of the UserDesign (*.bdf)

2. **TopLevelDesign file** ./BPU_ProgrammableLogicDevice.bdf
   Within this file you must insert the symbol, which was generated from the UserDesign in the file ./UserDesign/PortalCrane_Mealy_Schematic.bsf . Furthermore the inputs and outputs of the symbol must be interconnected with the corresponding signal bits of the GOLDiInterface_v1_00 sensor and actuator bits according the pin description of the physical system.

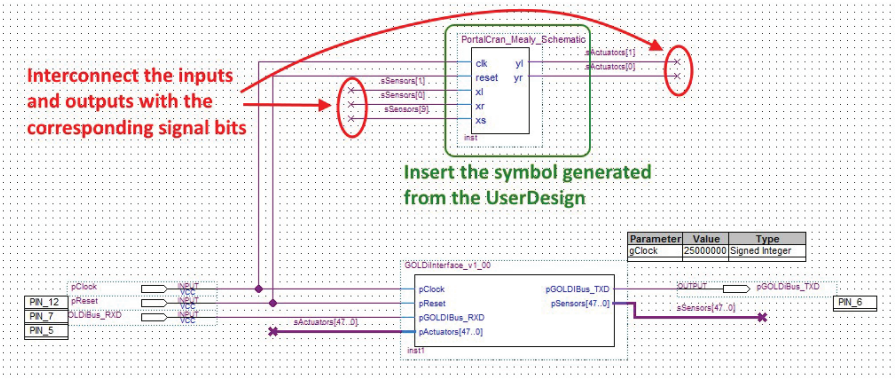


Figure 17: TopLevel file for a schematic design (*.bdf)

Textual Design

1. **UserDesign file** ./UserDesign/PortalCrane_Mealy_FSM.vhd
   The following VHDL description was automatically generated from the user design automaton graph ./UserDesign/PortalCrane_Mealy_FSM.smf , which you can create by using the integrated State Machine Wizard.

```
-- Copyright (C) 1991-2014 Altera Corporation. All rights reserved.
-- Generated by Quartus II Version 14.1.0 Build 186 12/03/2014 SJ Web Edition
-- Created on Fri Aug 07 13:12:32 2015

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY PortalCrane_Mealy_FSM IS
    PORT (
        clock   : IN STD_LOGIC;
        reset   : IN STD_LOGIC := '0';
        xl      : IN STD_LOGIC := '0';
        xr      : IN STD_LOGIC := '0';
        xs      : IN STD_LOGIC := '0';
        yl      : OUT STD_LOGIC;
        yr      : OUT STD_LOGIC
    );
END PortalCrane_Mealy_FSM;

ARCHITECTURE BEHAVIOR OF PortalCrane_Mealy_FSM IS
    TYPE type_fstate IS (Z0,Z1,Z2);
    SIGNAL fstate : type_fstate;
    SIGNAL reg_fstate : type_fstate;
BEGIN
    PROCESS (clock,reg_fstate)
    BEGIN
        IF (clock='1' AND clock'event) THEN
    END PROCESS;
```

Figure 18: Textual description of the UserDesign in VHDL

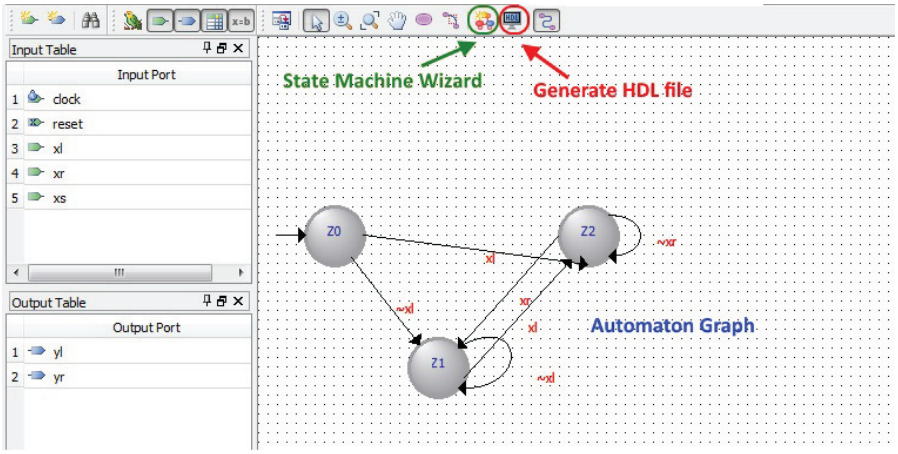
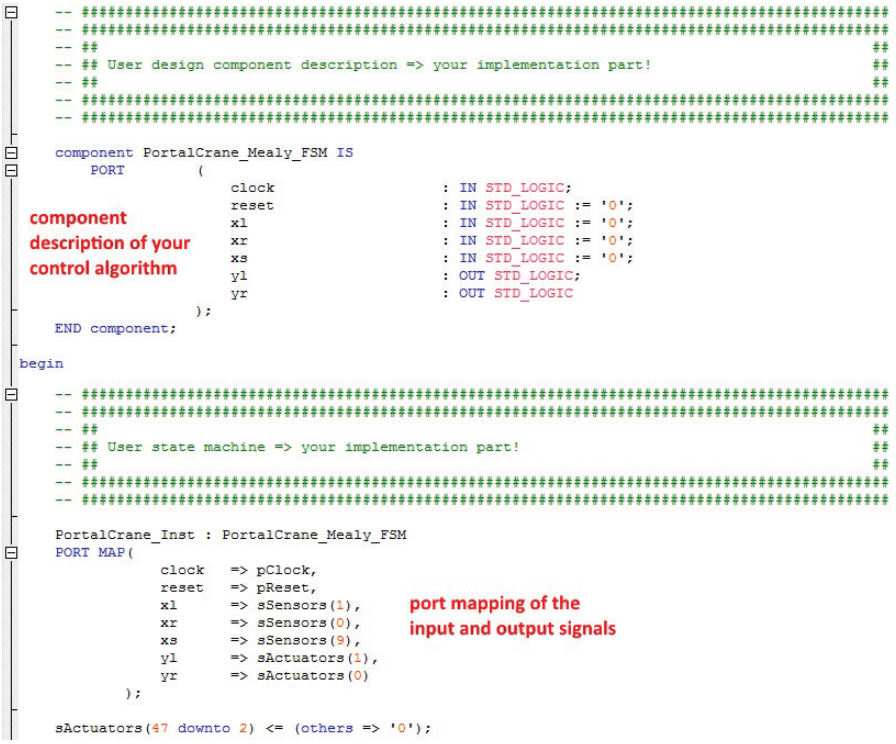


Figure 19: UserDesign automaton graph

2. **TopLevelDesign file** ./BPU_ProgrammableLogicDevice.vhd
   Within the TopLevelDesign file you have to modify the component description of your implemented control algorithm and the port mapping of the input and output signals to the corresponding sensor and actuator bits according the interface description of the physical system.




```
    -- ##############################################################################################
    -- ##############################################################################################
    -- ##                                                                                          ##
    -- ## User design component description => your implementation part!                           ##
    -- ##                                                                                          ##
    -- ##############################################################################################
    -- ##############################################################################################

    component PortalCrane_Mealy_FSM IS
        PORT        (
                        clock                   : IN STD_LOGIC;
                        reset                   : IN STD_LOGIC := '0';
                        xl                      : IN STD_LOGIC := '0';
                        xr                      : IN STD_LOGIC := '0';
                        xs                      : IN STD_LOGIC := '0';
                        yl                      : OUT STD_LOGIC;
                        yr                      : OUT STD_LOGIC
                    );
    END component;

begin

    -- ##############################################################################################
    -- ##############################################################################################
    -- ##                                                                                          ##
    -- ## User state machine => your implementation part!                                         ##
    -- ##                                                                                          ##
    -- ##############################################################################################
    -- ##############################################################################################

    PortalCrane_Inst : PortalCrane_Mealy_FSM
    PORT MAP(
                clock   => pClock,
                reset   => pReset,
                xl      => sSensors(1),
                xr      => sSensors(0),
                xs      => sSensors(9),
                yl      => sActuators(1),
                yr      => sActuators(0)
            );

    sActuators(47 downto 2) <= (others => '0');
```

Figure 20: TopLevel file for a textual design in VHDL

Step 3: Complete the project

- After adapting the files, you have to compile your project in Quartus II. If errors occur, you have to correct them according to the error message.
- If your data is correct, you can upload the Outputfile (*.pof) from the new folder "output_files" to the experiment.

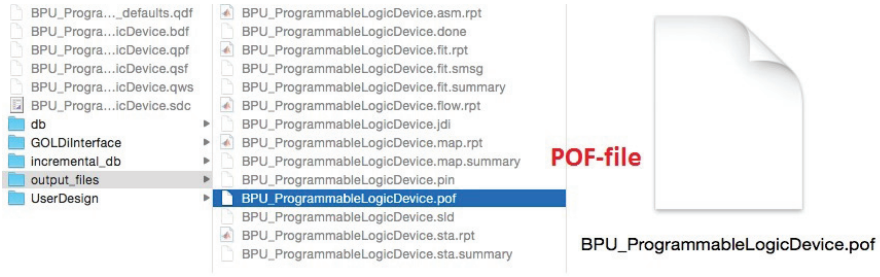


Figure 21: pof-file within the folder "output_files"

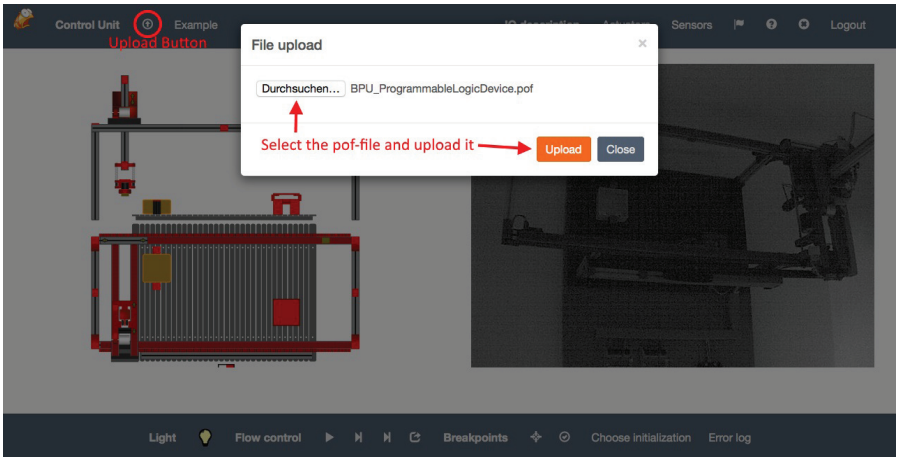


Figure 22: Upload the outputfile to the experiment

- For further information how to test your project within your chosen physical system you can read the description to "How to use ECP".

5 Finite State Machine

A **finite state machine (FSM)** is a model of a behaviour composed of a finite set of states and transitions. Within this context the FSM describes the sequential behaviour of a digital system. Depending on the internal state the system can react to the same input with different outputs.

In this example a single FSM is not enough to describe the digital system, so we use a parallel machine with two machine parts to implement the requirements.

5.1 Graphical Interactive FSM Tool

To design a finite state machine we use the **graphical interactive FSM tool (GIFT)**. This tool can create a FSM in form of a state machine, a transition table, a machine table or z-equations. For more information about GIFT and the various forms of a FSM, you can read the user guide to GIFT.

Click here to open the Graphical Interactive FSM Tool (GIFT).

5.2 Example of a parallel machine for the physical system "3-Axis-Portal"

In this example we create a parallel machine, which describes the following behaviour of the "3-Axis-Portal". The crane starts in the X-,Y-Position. First of all the crane drives to the right until it reaches the right position on the X-axis. Then the crane drives down and up again in Z-direction. Once the crane has finished its movement upwards it drives to the left until it reaches the left position on the X-axis.

We need to create two seperate machines in GIFT, which we adapt to each other afterwards. The first machine controls the movement on the X-axis (left-right). With the help of the State Diagram and the Transition Table we create the machine and generate the corresponding z-/y-equations.

The input-variable "x3" is only a placeholder for the state Z0 ("up") from machine A1, which will be defined later. The other variables correspond to the sensors and actuators from the physical system "3AxisPortal".

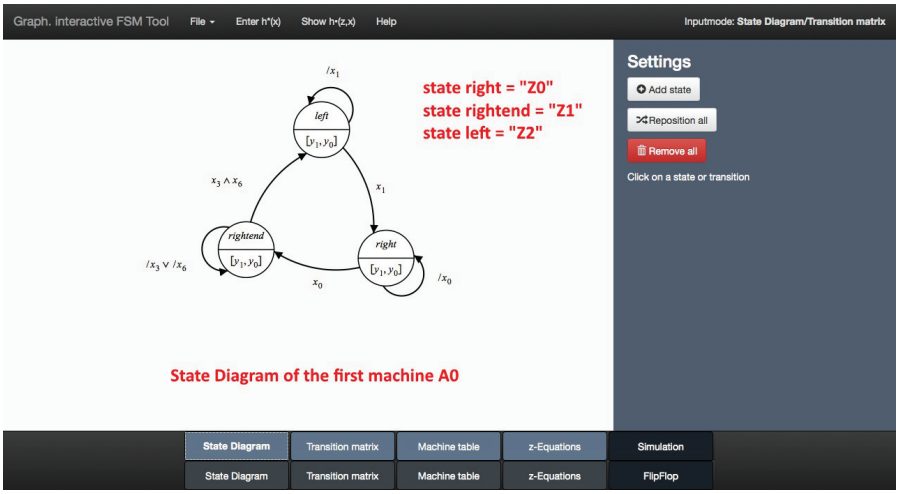


Figure 23: State Diagram of machine A0

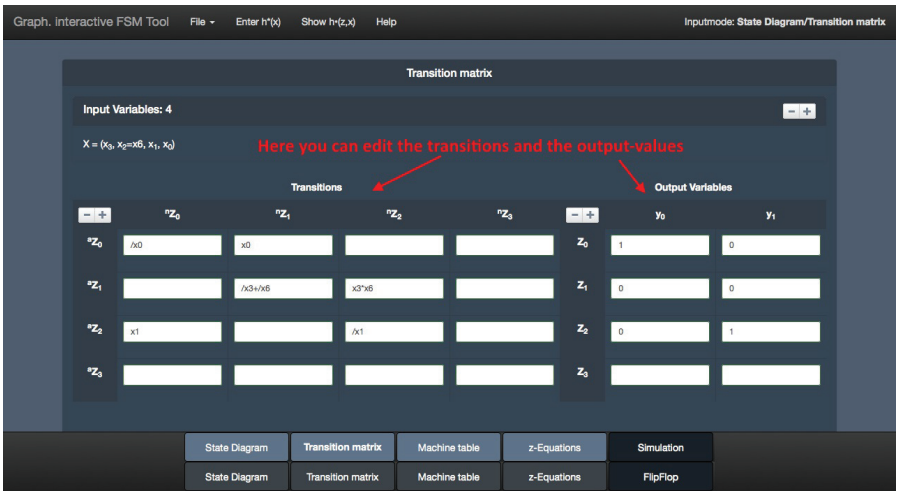


Figure 24: Transition Table of machine A0

The first machine A0 can be imported into the experiment with the ECP Export from GIFT. Therefore click on ECP Export in the File-menu in GIFT and click on the Import from GIFT -Button in the File-menu in the experiment. Then you have to name your machine and adapt the variables from GIFT to the physical system in a pop-up window.

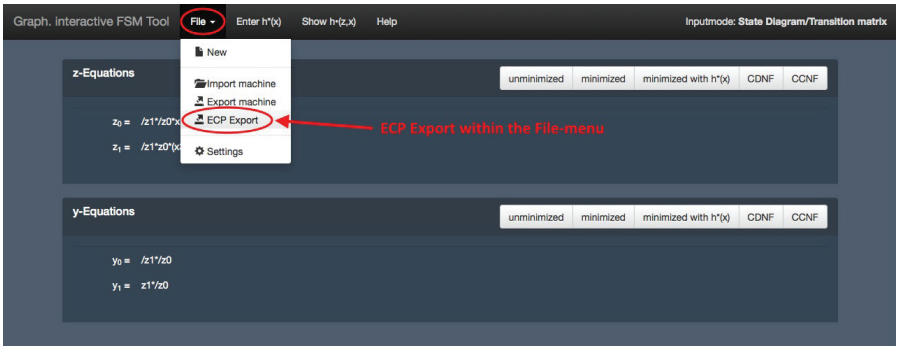


Figure 25: ECP-Export in GIFT

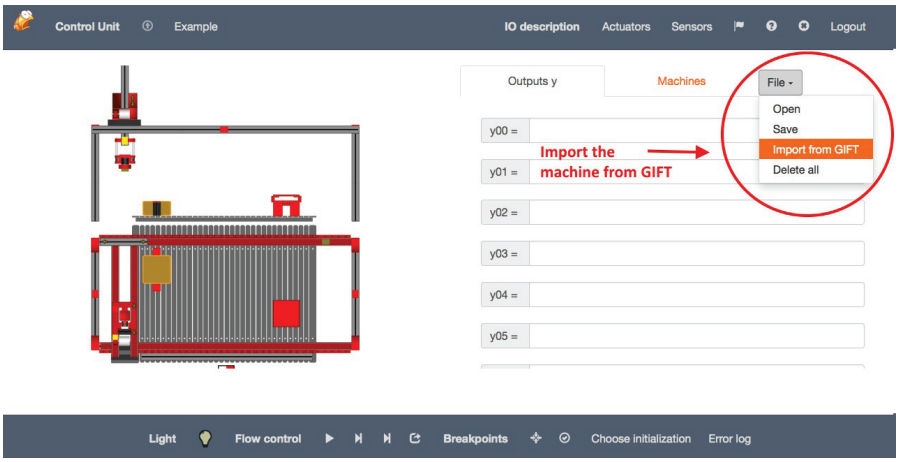


Figure 26: Import from GIFT in experiment

Name the first machine "A0" and assign the variables to the sensors and actuators from the system. In this example link the variables x6 to X6, x1 to X1, x0 to X0, y1 to Y1 and y0 to Y0. The variable x3 is still a placeholder, so you can assign it to any sensor of the system that you do not need (here: x3 to X9).

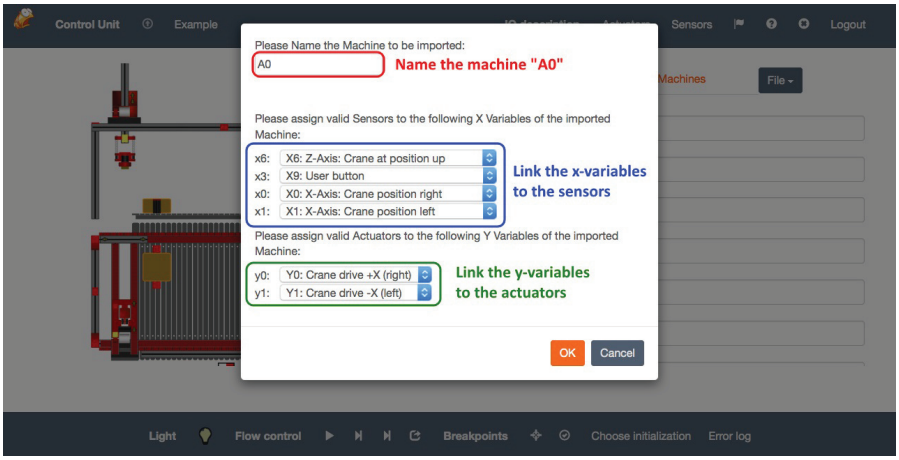


Figure 27: Settings for imported machine in the experiment

After importing the first machine A0 into the experiment, add a new machine A1 and change back again to GIFT to create the second machine A1 similar to the machine A0, which controls the movement on the Z-axis (down-up). This time you can not import the machine via ECP into the experiment, so you have to copy the z-/y-equations from GIFT and insert them into the correct input field in the experiment.

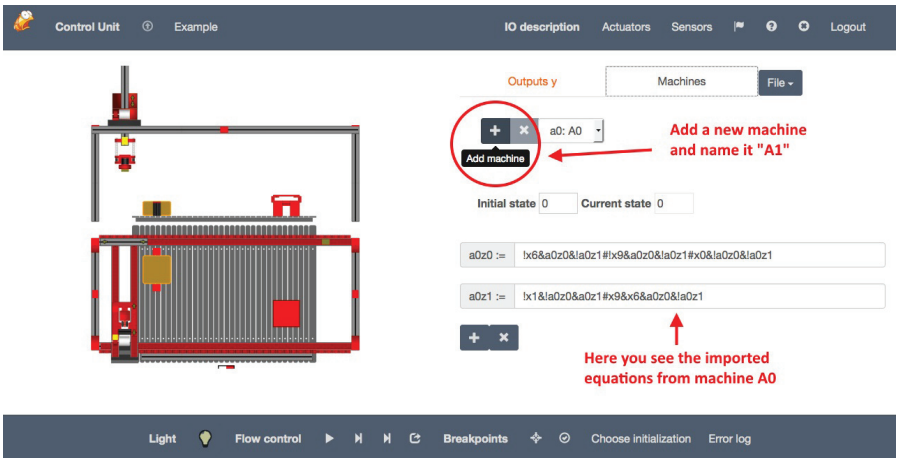


Figure 28: Add a new machine (A1)

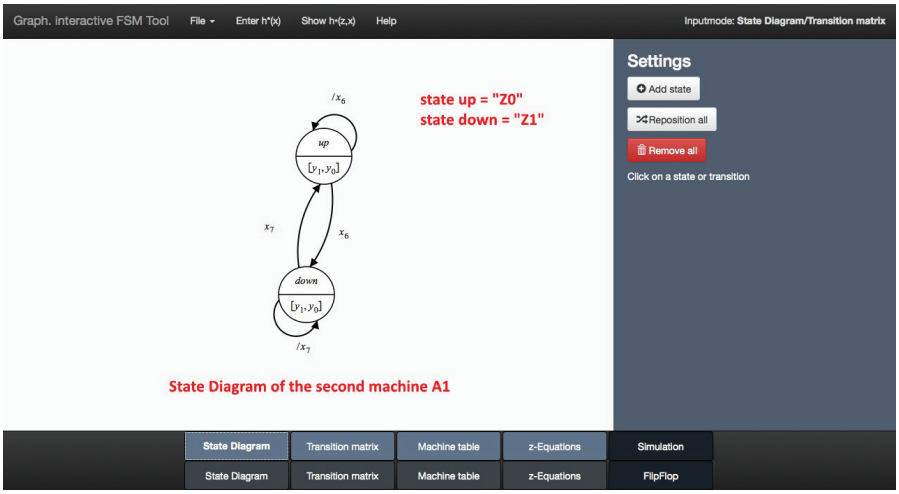


Figure 29: State Diagram of machine A1

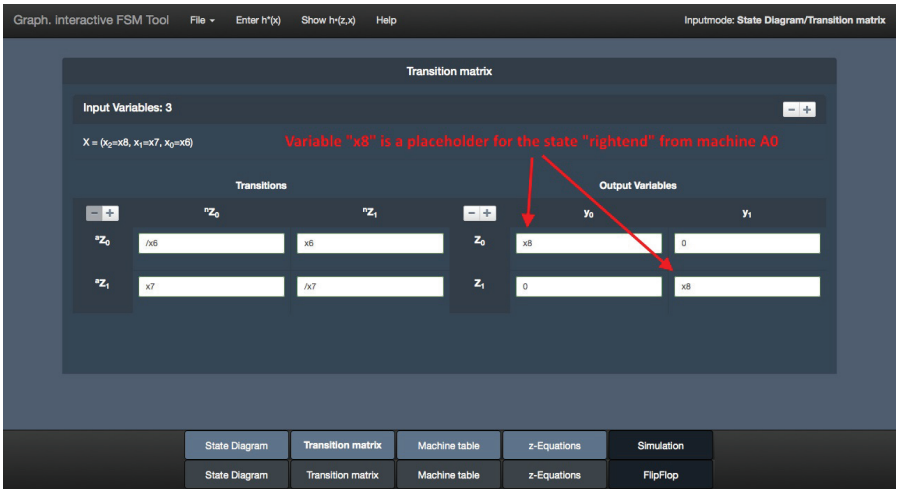


Figure 30: Transition Table of machine A1

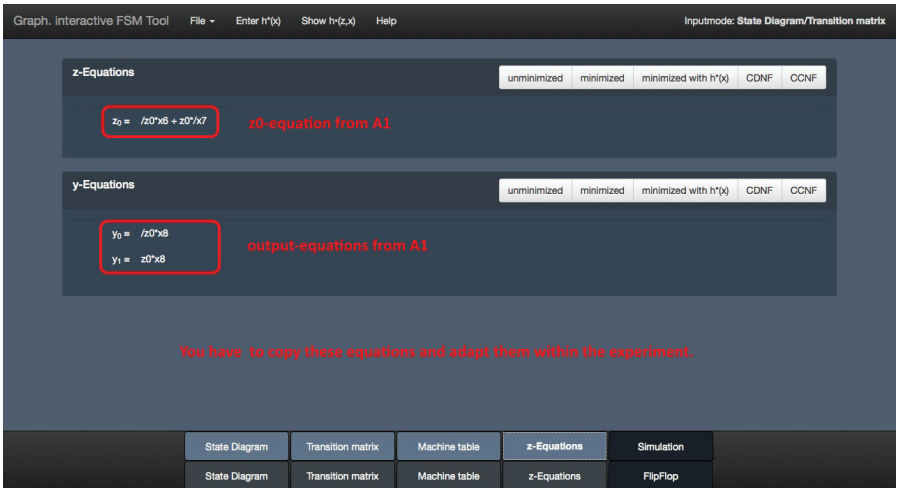


Figure 31: z-/y-equations of machine A1

Now you have to copy the z0-equation into the "a1z0"-field from machine A1, the y0-equation into the "y04"-field and the y1-equation into the "y05"-field from the "Outputs y" and adapt the equations. Change "z0" to "a1z0", "/" to "!", "*" to "&", "+" to "#" and "x8" to "!a0z1&a0z0". Furthermore you have to change the placeholder-variable "x9" from machine A0 to "!a1z0".

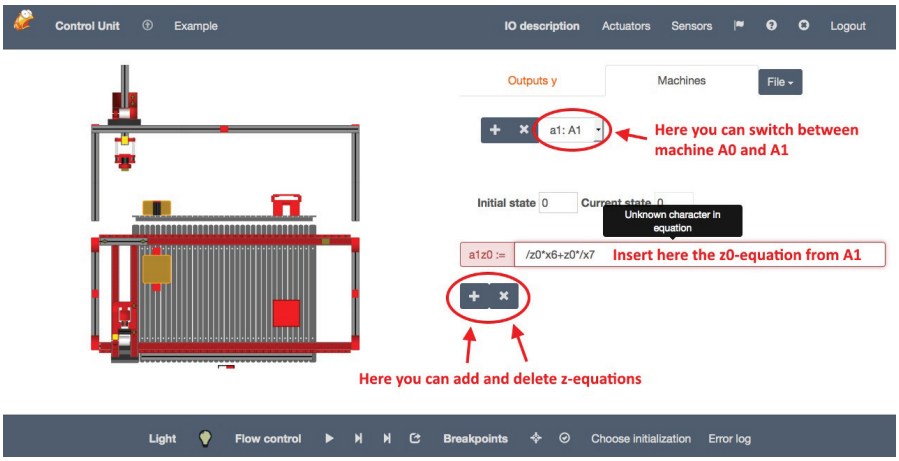


Figure 32: Copy the z0-equation from A1

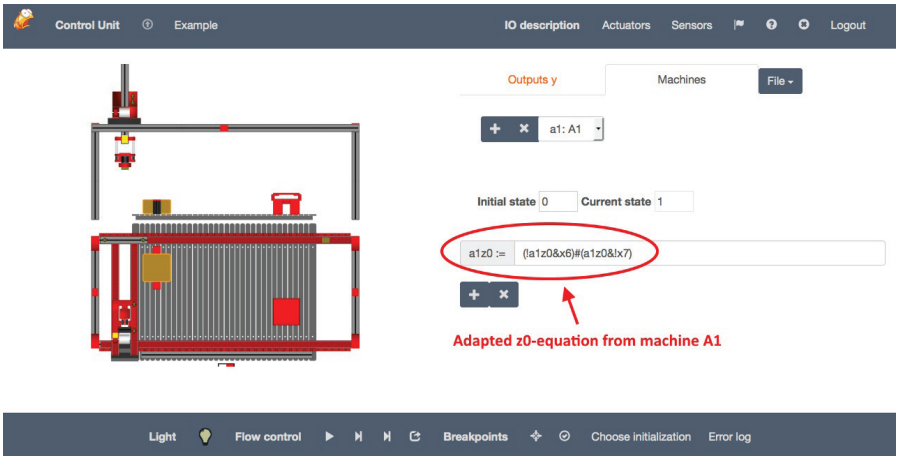


Figure 33: Adapt the z0-equation from A1

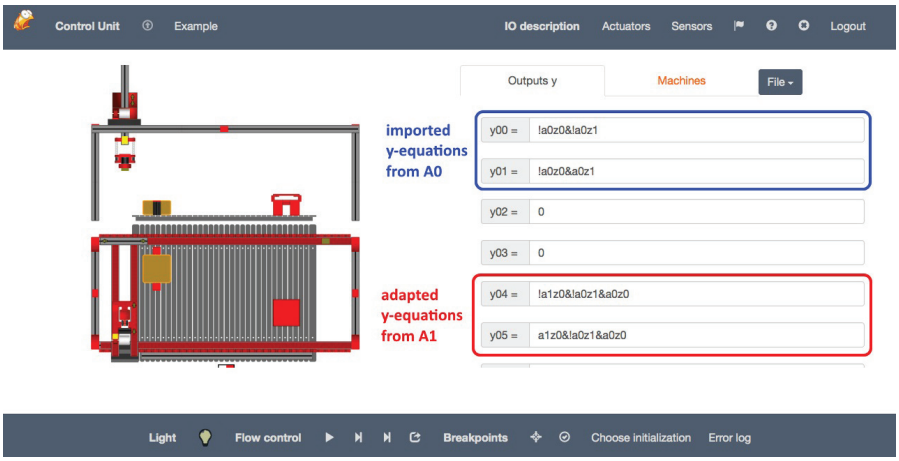


Figure 34: Copy and adapt the output-equations

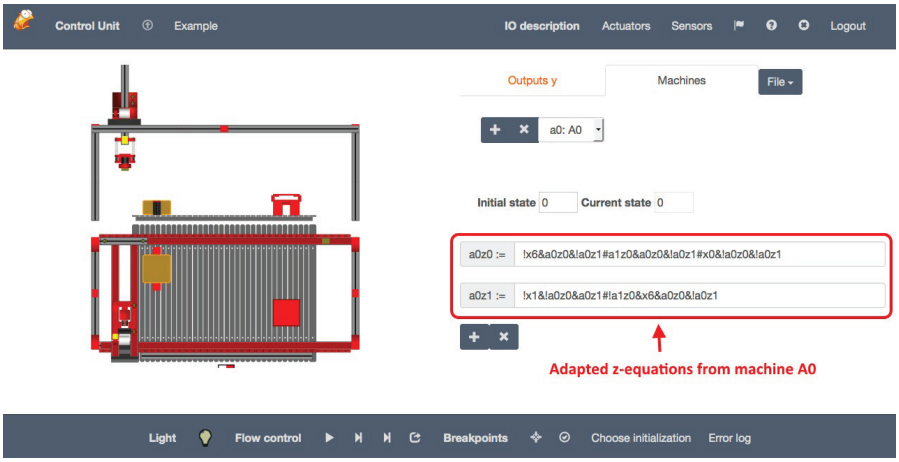


Figure 35: Adapted z-equations of machine A0

When you have created and adapted both machines A0 and A1 in the experiment, you can test the functionality of the parallel machine by clicking on the "Start"-button. If everything is correct, you can see the described movement of the "3-Axis-Portal" in the animation on the left.

6 Microcontroller

A **Microcontroller** is a small computer on a single integrated circuit containing a processor core, memory and programmable input/output peripherals.Depending on the device, the program memory may be permanent, read-only memory that can only be programmed at the factory, or program memory that may be field-alterable flash or erasable read-only memory.

In this case we use a MicroController from ATMEL (Atmega644a). These MicroControllers combines 64KB ISP flash memory with read-while-write capabilities, 2KB EEPROM, 4KB SRAM, 32 general purpose I/O lines, 32 general purpose working registers, a real time counter, three flexible timer/counters with compare modes and PWM, two USARTs, a byte oriented 2-wire serial interface, an 8-channel 10-bit A/D converter with optional differential input stage with programmable gain, programmable watchdog timer with internal oscillator, SPI serial port, a JTAG (IEEE 1149.1 compliant) test interface for on-chip debugging and programming, and six software selectable power saving modes.

6.1 WIDE

WebIDE (WIDE) is the integrated development environment created specifically for GOLDi. It can be accessed in its standalone version under Tools as well as inside of the Experiment Control Panel for experiments using Microcontrollers as their Control Unit. There it is accessible under WIDE in the top toolbar.

6.2 Project example "3AxisPortal_Mealy"

Step 1: Create a project

- Open WIDE
- Create your first project

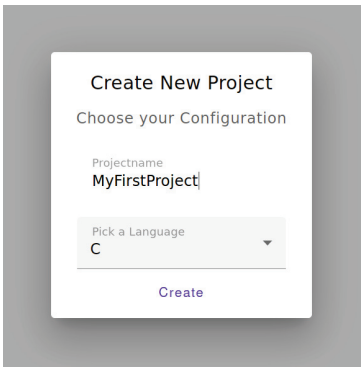


Figure 36: Project creation prompt

- Load the example "3AxisPortal_Mealy"

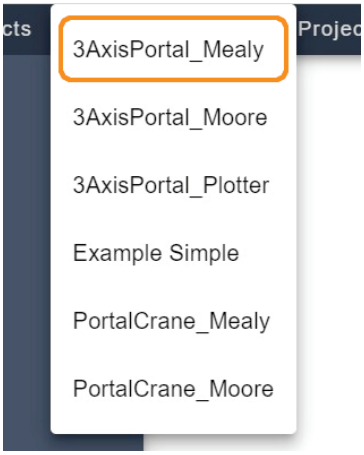


Figure 37: Examples dropdown menu with 3AxisPortal_Mealy highlighted

- On the left you can see a filetree of the currently selected project. The loaded example has the following structure:

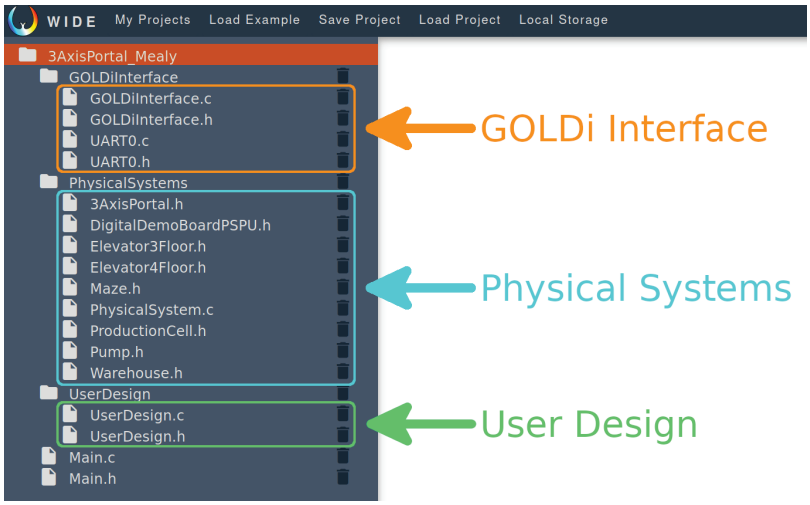


Figure 38: Structure of the example 3AxisPortal_Mealy

- The folder **GOLDiInterface** contains the files which are necessary for the communication within the GOLDi infrastructure. These files are ready to use - no modifications are necessary. They will be included automatically.

- The folder **PhysicalSystems** contains the interface definitions for all available physical systems (electromechanical hardware models). These files are ready to use - no modifications are necessary. They will be included automatically.
- The folder **UserDesign** contains the actual control algorithm.
- The file **Main.c** is ready to use - no modifications are necessary.
- The file **Main.h** contains the library definitions for the project.

Step 2: Adapt the files for your project
The following files must be adapted according to the actual control and the selected physical system:

- **Main.h**
- **UserDesign.h** in the "UserDesign" folder
- **UserDesign.c** in the "UserDesign" folder

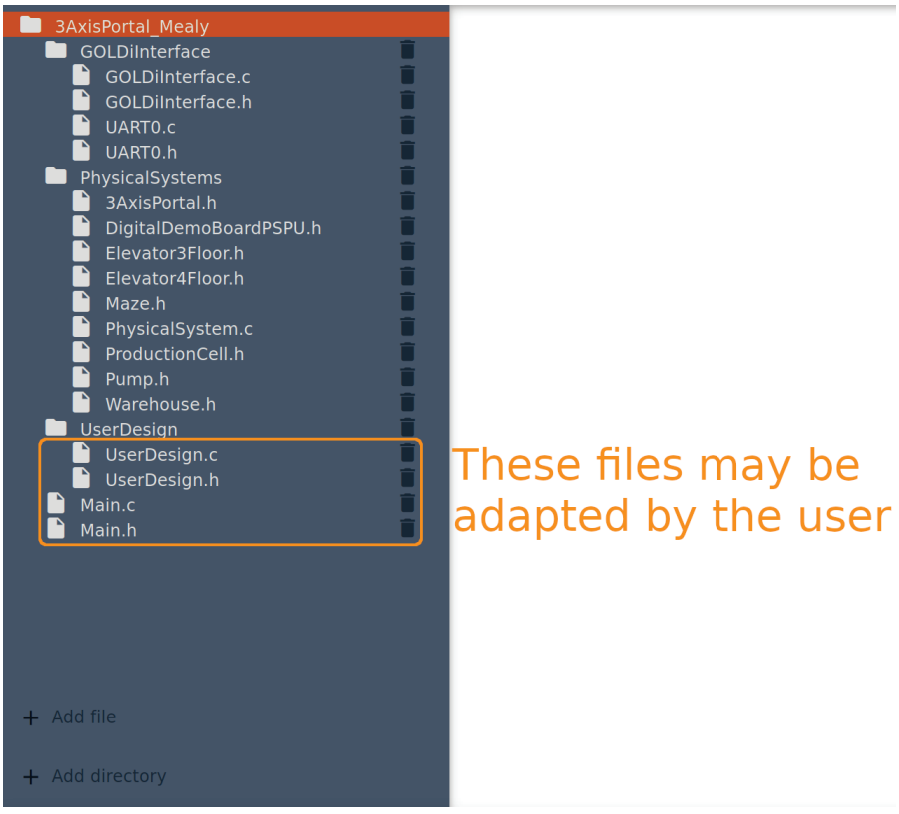


Figure 39: User-adaptable files

Don't modify the directory structure or rename the files!

1. **Main.h** ./Main.h
   Within this file you must select your connected physical system.

```
// ###########################################################################
// #                                                                         #
// #  This module selects the controlled physical system                     #
// #                                                                         #
// ###########################################################################

#ifndef MAIN_H_
    #define MAIN_H_

// ###########################################################################
// #  Uncomment the controlled physical system                               #
// ###########################################################################

    #define PSPU_3AxisPortal
    //#define PSPU_ProductionCell
    //#define PSPU_Elevator3Floor
    //#define PSPU_Elevator4Floor
    //#define PSPU_Elevator4FloorClassic
    //#define PSPU_Pump
    //#define PSPU_Warehouse
    //#define PSPU_Maze
    //#define PSPU_DigitalDemoBoardPSPU


// ###########################################################################
// #  Physical system specific libraries -- DO NOT CHANGE ANYTHING BELOW! -- #
// ###########################################################################

    #ifdef PSPU_3AxisPortal
        #include "PhysicalSystems/3AxisPortal.h"
    #endif

    #ifdef PSPU_ProductionCell
        #include "PhysicalSystems/ProductionCell.h"
    #endif

    #ifdef PSPU_Elevator3Floor
        #include "PhysicalSystems/Elevator3Floor.h"
    #endif
```

Here you need to select your connected Physical System
For this example 3AxisPortal was selected

Figure 40: Defines for selected Physical System in Main.h

   - In this case "3AxisPortal" has been selected
   - You can comment out the redundant physical systems by writing "//" in front of the corresponding line

2. **UserDesign.h** ./UserDesign/UserDesign.h
   Within this file you make some definitions for the file UserDesign.c.

```
// ###########################################################################
// #                                                                         #
// #  This module implements the users design                                #
// #                                                                         #
// ###########################################################################

#ifndef _USERDESIGN_H
    #define _USERDESIGN_H

    #include "../Main.h"
    extern void StateMachineInit(void);                    // This function inits the state machine
    extern void StateMachineUpdate(void);                  // This function updated the state machine

// ###########################################################################
// #  Add a new state for state machine here                                 #
// ###########################################################################
    typedef enum
    {
        Z0_WaitForFallingEdge,                             // Z0 - wait for 1/0 edge
        Z1_DriveXMinusYMinus,                              // Z1 - drive to left/down
        Z2_DriveYPlusXPlus                                 // Z2 - drive to up/right
    } AutomatStates_t;

#endif
```

Here you can define the states of the state machine

Figure 41: The states of our state machine in UserDesign.h

- Here we define for example an automaton with three states (Z0, Z1, Z2).
- For a FSM based design, the FSM states must be defined.
- **These state names must be used within the program "UserDesign.c"**

3. **UserDesign.c** ./UserDesign/UserDesign.c
   Within this file you define the control algorithm.
   - At first define the initial state:

```
// ###############################################################################
// #                                                                             #
// #  This module implements the users design                                    #
// #                                                                             #
// ###############################################################################

#include <util/delay.h>
#include "UserDesign.h"

AutomatStates_t State;


// ###############################################################################
// #  This function initializes the finite state machine with start state        #
// ###############################################################################
void StateMachineInit(void)
{
    State = Z0_WaitForFallingEdge;
}

// ###############################################################################
// #  This function updates the current state of the finite state machine        #
// ###############################################################################
void StateMachineUpdate(void)
{
    switch (State)
    {
        case Z0_WaitForFallingEdge:
        {
            Actuators.XAxisDriveToXPlus     = 0;
            Actuators.XAxisDriveToXMinus    = 0;
            Actuators.YAxisDriveToYPlus     = 0;
            Actuators.YAxisDriveToYMinus    = 0;
            Actuators.ZAxisDriveToZPlus     = 0;
            Actuators.ZAxisDriveToZMinus    = 0;
            Actuators.Magnet                = 0;

            if (!Sensors.UserSwitch)
                State = Z1_DriveXMinusYMinus;
            //else
```

Here you define your initial state
In our example we choose Z0 as the initial state

Figure 42: Initial state of our state machine in UserDesign.c

   - for the second step you have to define the output (here: actuator values for each state).

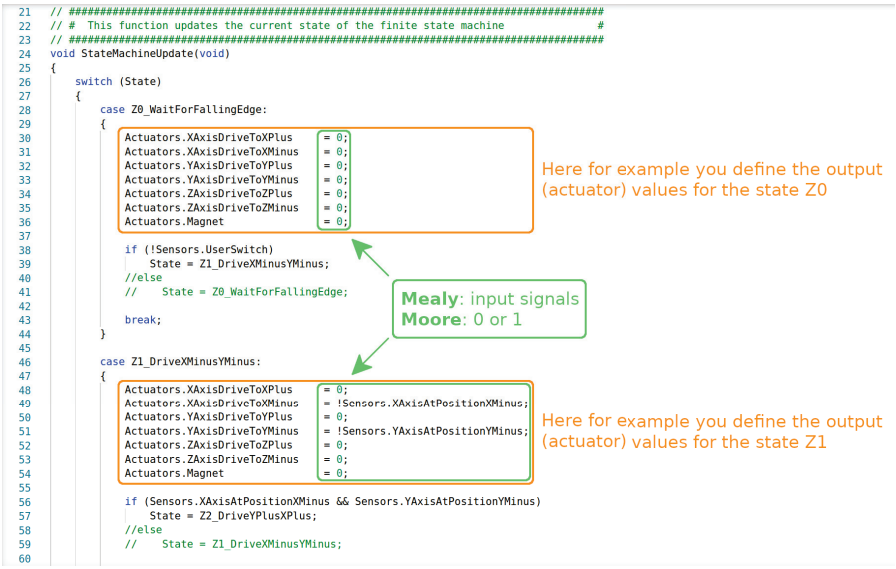


Figure 43: The output values of our states in UserDesign.c

- Here we have a mealy automaton, so we must define input signals.
- You find the definition of all inputs (sensors) and outputs (actuators) from your physical system in the corresponding file in the 'PhysicalSystems'-folder. Here for example for the 3AxisPortal you will find them in ./PhysicalSystems/3Axis.h
- The definition of all inputs (sensor) and outputs (actuators) according the pin description
- **The specific signal names for each physical system must be used within the ./UserDesign program files.**

– At the third step you have to define the next state function for each state.

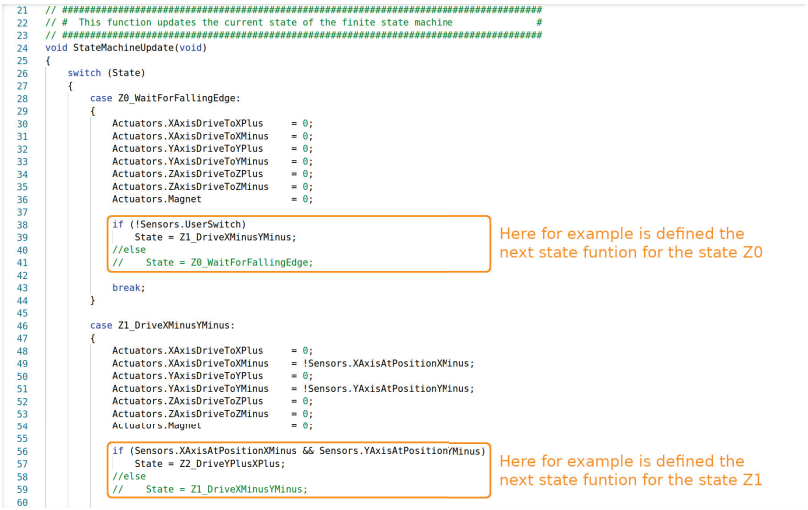


Figure 44: The next state functions of our state machine in UserDesign.c

Step 3: Complete the project

- After adapting the files, you have to compile your project. This can be achieved by clicking on Compile in the upper right corner. If any errors occur they need to be fixed according to the contents of the error message before the next step.
- If your program doesn't contain anymore syntactical errors you can proceed to upload your program on the Control Unit by clicking the Upload button next to the Compile button (this is only available in the ECP-version of WIDE and not in the standalone version).

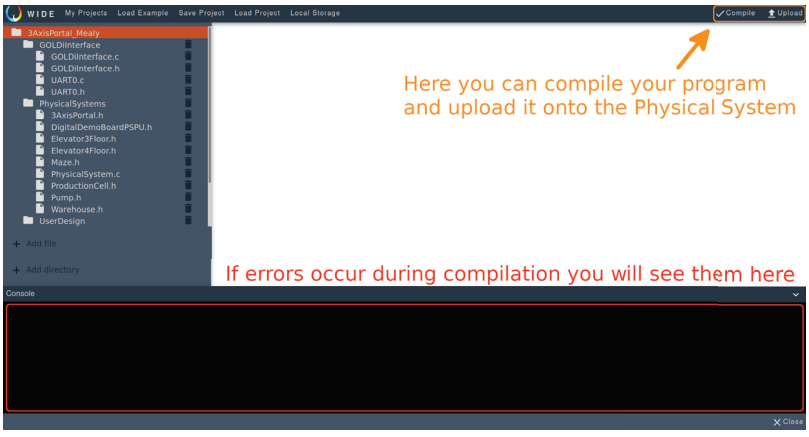


Figure 45: Compile, upload and console