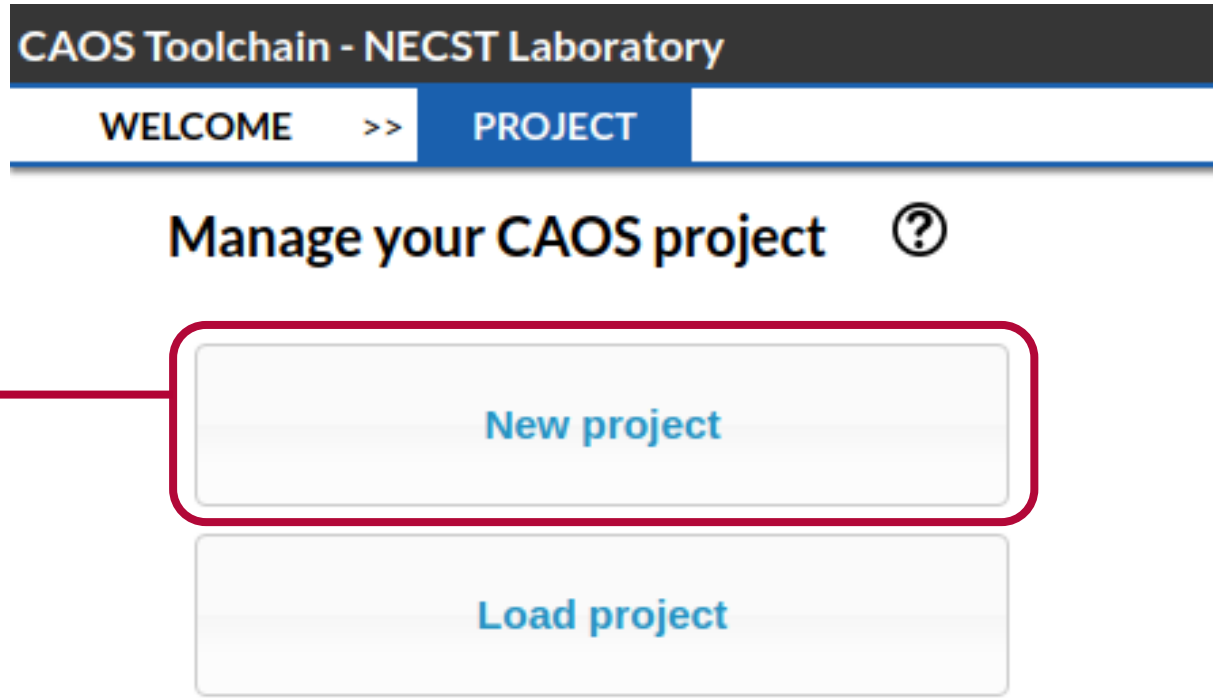
# CAOS tutorial

# Access CAOS at: http://caos-iccd.necst.it/

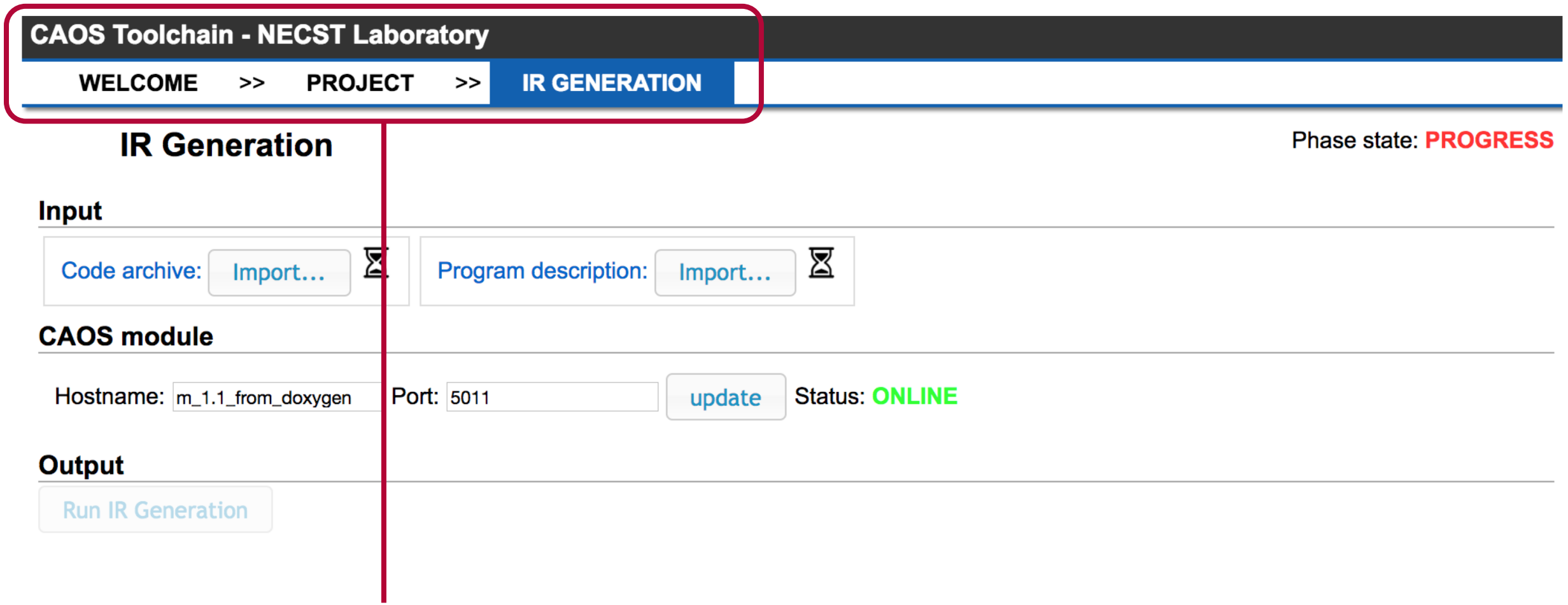
Click on "*click here to start*" to start the optimization process

# Use CAOS to optimize your application

- Start a new project or load your work right were you left it

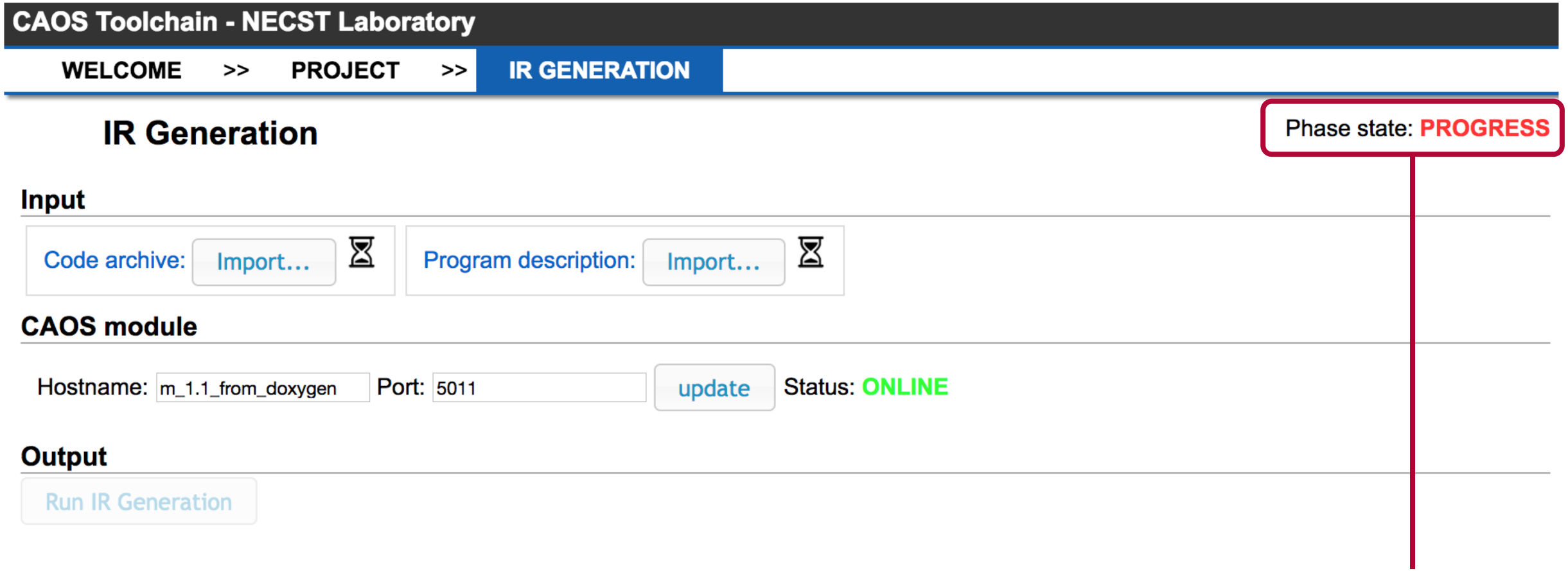


# GUI Overview



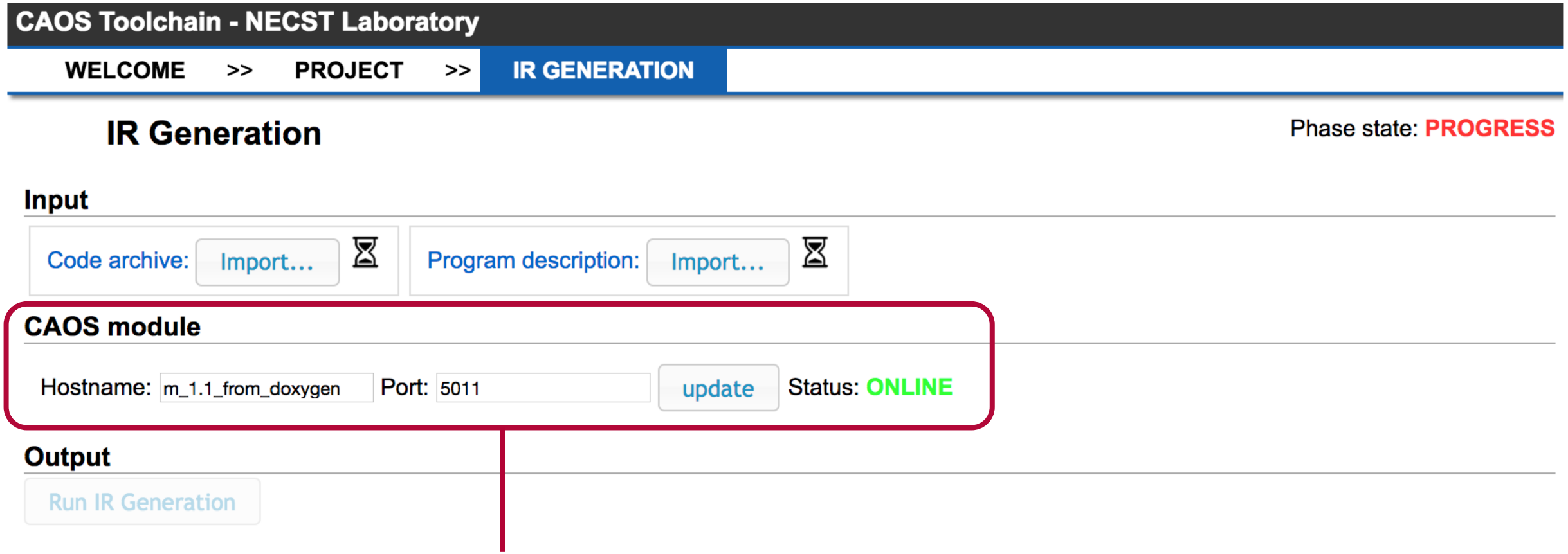

Keep track of the **phase** that is being performed using the *navigation window*

# GUI Overview



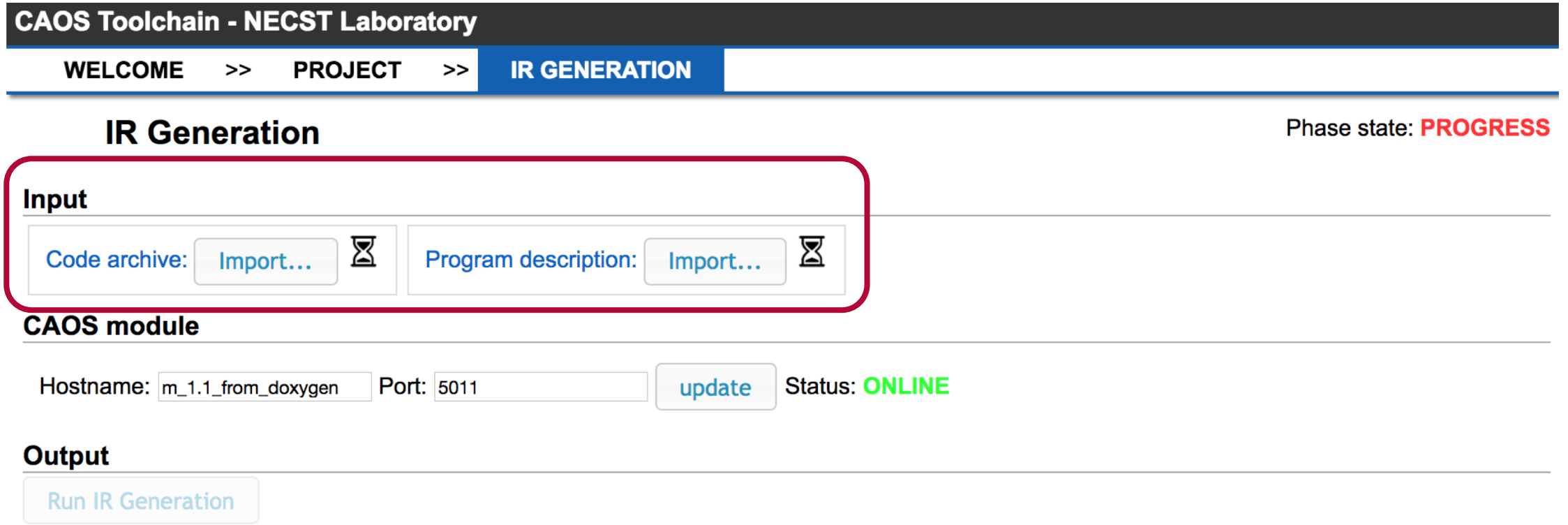

Check the **status** of the **phase** on the top-right part of the screen

# GUI Overview



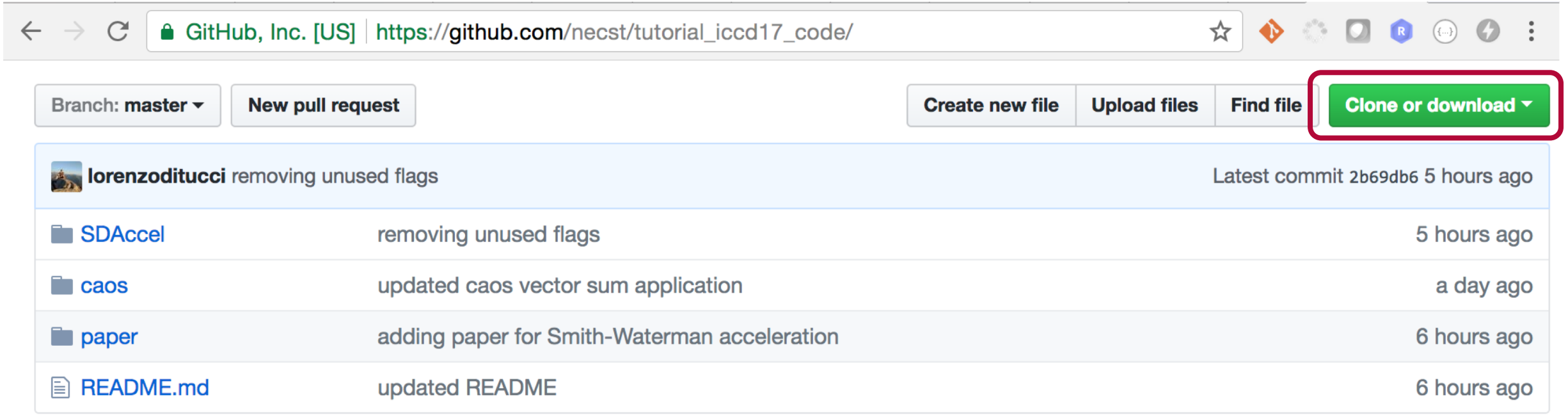

- One of the main features of CAOS is **Modularity**
  - Every phase is backed up by a CAOS module that can be accessed simply specifying its hostname and port
  - The **CAOS module panel** provides information regarding the status of the specific module

# IR GENERATION




- **IR Generation** generates the CAOS intermediate representation from the user's source code
- For this phase you need to specify:
  - A zip/tar archive containing the source code of the application to optimize
  - A program description JSON file specifying the source code language and how to compile the application

# Demo applications




- Demo applications are available at: https://github.com/necst/tutorial_iccd17_code/
- After downloading the repository, you can access the CAOS demo apps under the “caos” folder:
  - Vessel Segmentation
  - Smith-Waterman
- In the following slides we will refer to the *Vessel Segmentation* demo application

# Program Description

program-description.json

```
{
    "language": "C++",
    "supportedCompilers" : [
        {
            "compiler": "gcc",
            "arguments": "-O3"
        },
        {
            "compiler": "llvm",
            "arguments": "-O3"
        }
    ]
}
```

- **Provides information regarding the source code:**
  - Source code language
  - Supported compilers
  - Arguments needed for compiling

# IR GENERATION

CAOS Toolchain - NECST Laboratory

WELCOME >> PROJECT >> IR GENERATION

**IR Generation**

Phase state: READY complete

**Input**

Code archive: Import... Program description: Import...

**CAOS module**

Hostname: m_1.1_from_doxygen Port: 5011 update Status: ONLINE

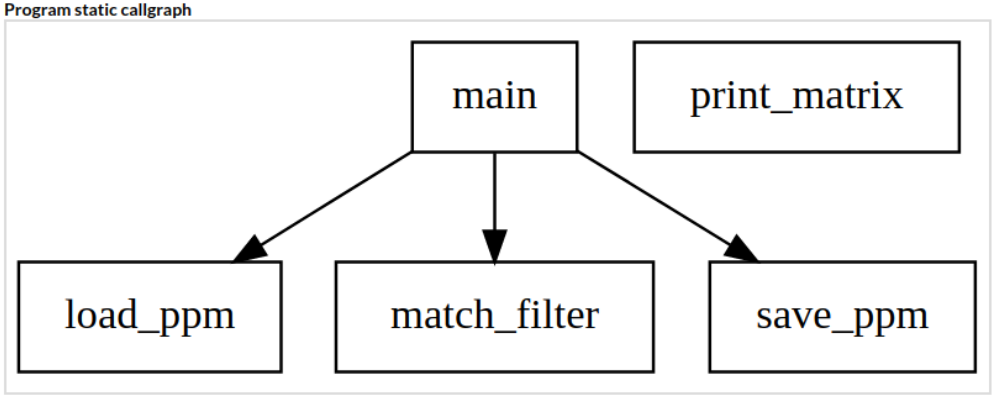
**Output**

Run IR Generation

**Identified functions**

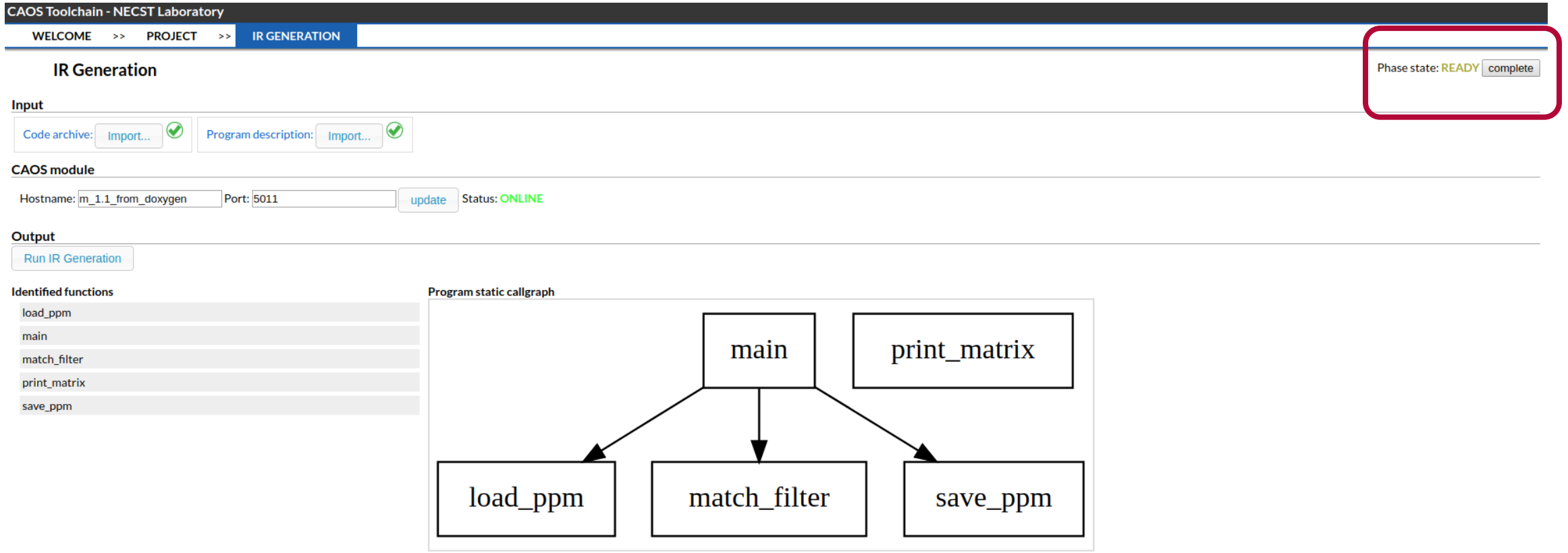
- load_ppm
- main
- match_filter
- print_matrix
- save_ppm

**Program static callgraph**



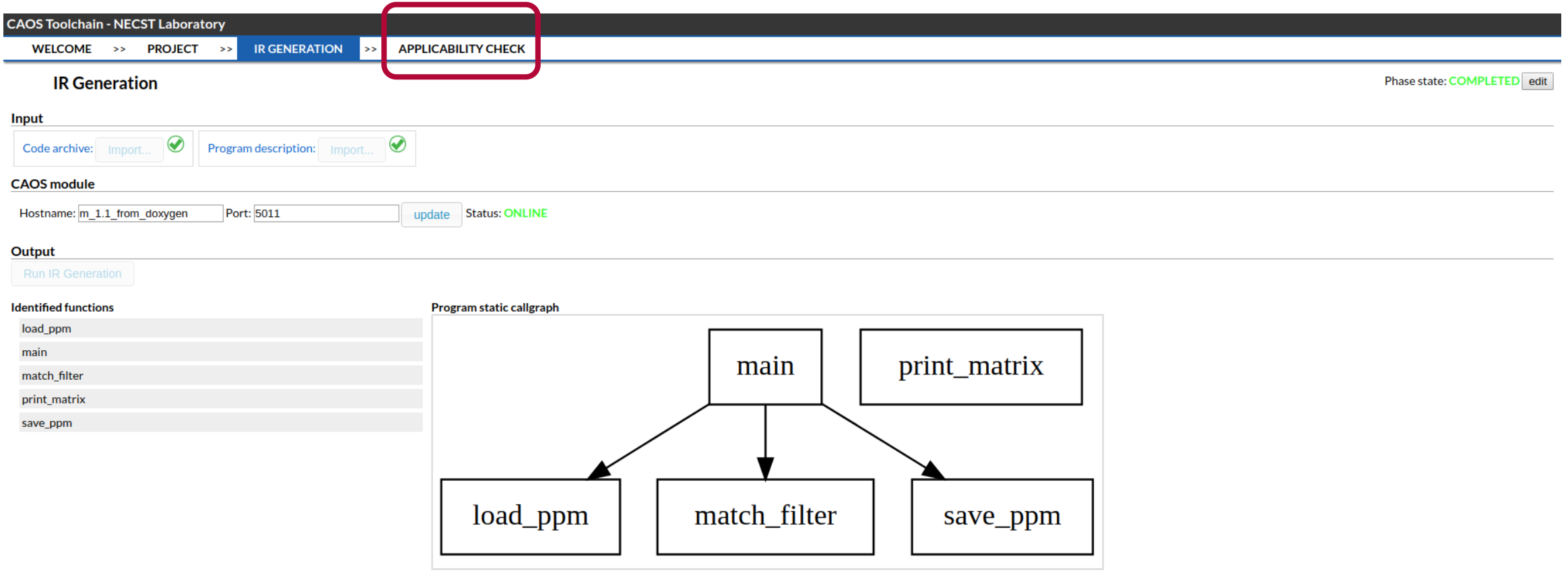

- Once the two files have been loaded, by clicking on **Run IR Generation** CAOS:
  - Identifies all the functions within the application
  - Provides a **Program Static Callgraph** highlighting the caller/calle relationships

# IR GENERATION



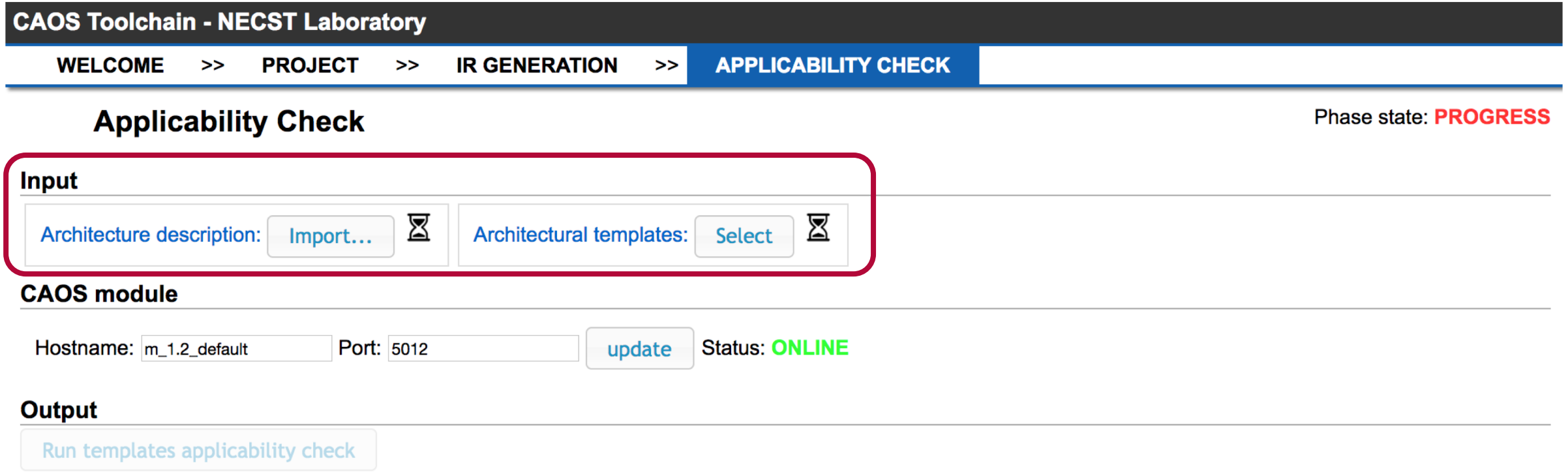

Phase state becomes *READY and* we can click on *complete* to move to the next phase

# IR GENERATION




Phase state becomes *COMPLETED* and CAOS shows the new phase in the navigation window

# APPLICABILITY CHECK




- **APPLICABILITY CHECK** verifies which CAOS architectural templates can be used for the given user code and target architecture
- This phase requires:
  - A JSON file providing the target **architecture description**
  - A selection of the **architectural templates** to check

# APPLICABILITY CHECK – Architecture Description

architecture-description.json

```
{
    "nodeDefinition" : {
        "deviceTypes" : {…
        },
        "devices" : {…
        },
        "connectionTypes" : {…
        },
        "connections" : […
        ]
    },
    "system" : {…
    }
}
```

- Describes the target architecture on which the application will be executed
- Two levels specification
  - Node description
  - System description

- Let's define the architecture description for a system consisting of a single f1.2xlarge instance…

# APPLICABILITY CHECK – Architecture Description

architecture-description.json

```
{
    "nodeDefinition" : {
        "deviceTypes" : {
            "f1-fpga" : {
                "type" : "board",
                "vendor" : "Xilinx",
                "partNumber" : "XCVU9P-FLGB2104-2-I"
            },
            "intel-vcore" : {
                "type" : "cpu",
                "vendor" : "intel",
                "partNumber" : "-"
            }
        },
        "devices" : {
            "f1-fpga-instance" : {
                "type" : "f1-fpga"
            },
            "cpu" : {
                "type" : "intel-vcore",
                "host" : true
            }
        },
        "connectionTypes" : { ... },
        "connections" : [ ... ]
    },
    "system" : { ... }
}
```

- Defines the type of devices to reference within the node specification
  - Intel CPU
  - Xilinx XCVU9P-FLGB2104-2-I board
- Instantiates the devices and specifies the device acting as “host” for the node

# APPLICABILITY CHECK – Architecture Description

architecture-description.json

```
{
    "nodeDefinition" : {
        "deviceTypes" : {...
        },
        "devices" : {...
        },
        "connectionTypes" : {
            "pcie_gen3" : {
                "standard" : "PCIe",
                "bandwidth" : "15.75 GB/s",
                "duplex" : "full",
                "version" : "3.0"
            }
        },
        "connections" : [
            {
                "source" : "f1-fpga_instance",
                "target" : "cpu",
                "type" : "pcie_gen3"
            }
        ]
    },
    "system" : {...
    }
}
```

- Specifies the available connections between the devices of the node
- Defines how the devices are interconnected

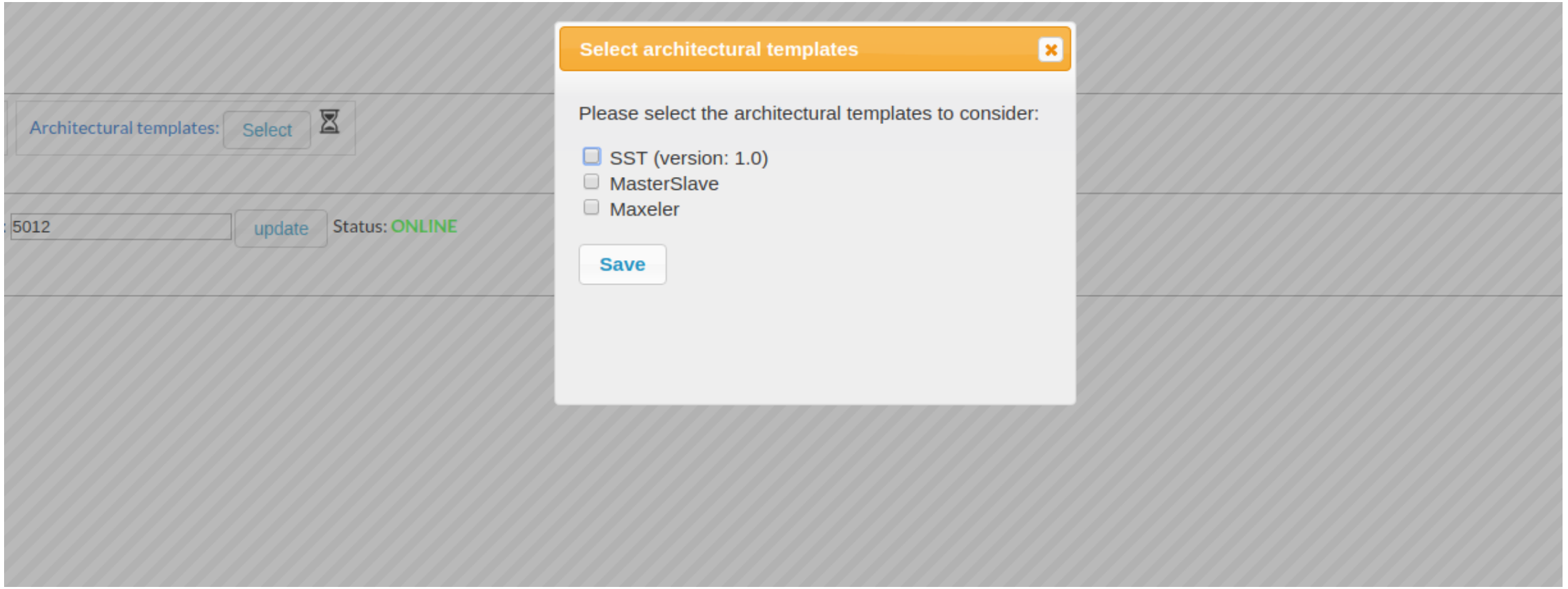
# APPLICABILITY CHECK – Architecture Description

architecture-description.json

```
{
    "nodeDefinition" : {
        "deviceTypes" : {...
        },
        "devices" : {...
        },
        "connectionTypes" : {...
        },
        "connections" : [...
        ]
    },
    "system" : {
        "nodes" : ["f1-fpga_node"],
        "connectionTypes" : {},
        "connections" : []
    }
}
```

- Defines the number of nodes available within the system and how they are interconnected
- For the f1.2xlarge instance we simply define a single-node system

# APPLICABILITY CHECK – Architectural Template




- Select which architectural template(s) you want CAOS to consider for your application
  - Click *Save*
  - Click on *Run templates applicability check*

# APPLICABILITY CHECK – Result

**Architectural Templates Validation:**

**MasterSlave**

Status: Template is supported!

**Architecture report:**

Supported devices:

- **f1-fpga_instance**

**Functions report:**

load_ppm
- Hardware acceleration: no
- Reason: the function contains calls to unsupported functions
- Additional info: function: 'load_ppm' call: [fopen]

main
- Hardware acceleration: no
- Reason: the function contains calls to unsupported functions
- Additional info: function: 'main' call: [printf]

match_filter
- Hardware acceleration: yes

print_matrix
- Hardware acceleration: no
- Reason: the function contains calls to unsupported functions
- Additional info: function: 'print_matrix' call: [printf]

save_ppm
- Hardware acceleration: no
- Reason: the function contains calls to unsupported functions
- Additional info: function: 'save_ppm' call: [fopen]

**Maxeler**

Status: Template not supported.
Reason: the system does not provide a supported device, or matches the specified connection rules
Additional info: supported devices: board_Vectis

- The *Architectural Template Validation* shows which architectural templates are being supported, and on which devices

# APPLICABILITY CHECK – Result

**Architectural Templates Validation:**

**MasterSlave**

Status: Template is supported!

**Architecture report:**

Supported devices:

- **f1-fpga_instance**

**Functions report:**

load_ppm
- Hardware acceleration: no
- Reason: the function contains calls to unsupported functions
- Additional info: function: 'load_ppm' call: [fopen]

main
- Hardware acceleration: no
- Reason: the function contains calls to unsupported functions
- Additional info: function: 'main' call: [printf]

match_filter
- Hardware acceleration: yes

print_matrix
- Hardware acceleration: no
- Reason: the function contains calls to unsupported functions
- Additional info: function: 'print_matrix' call: [printf]

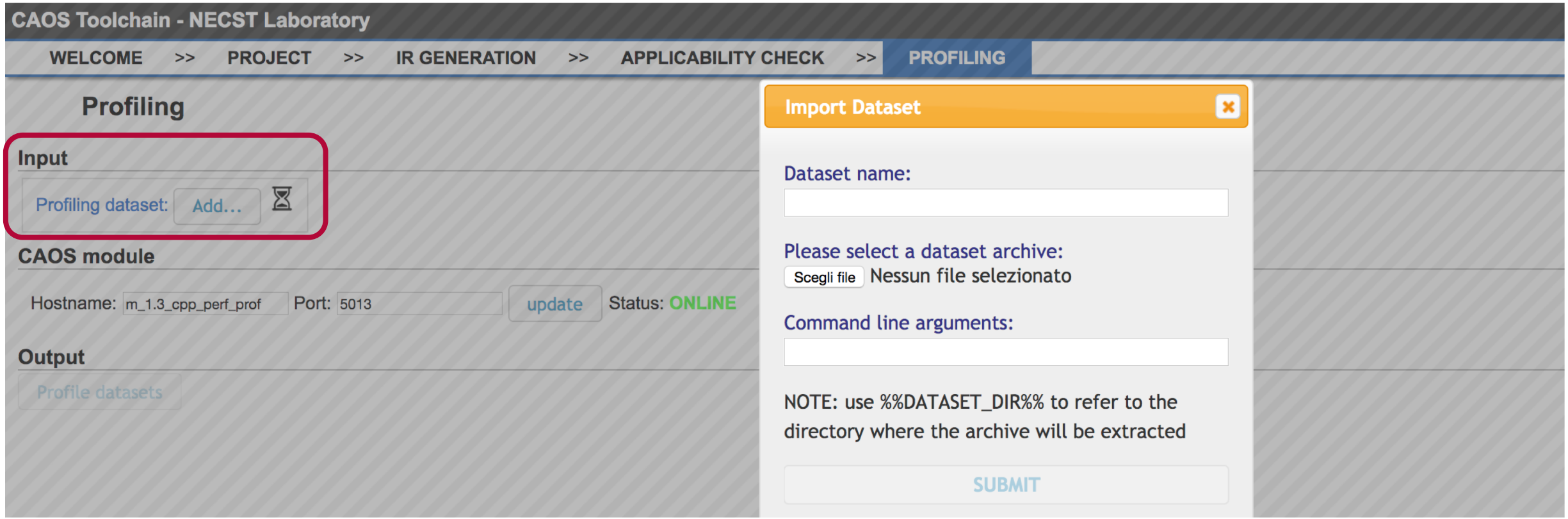
save_ppm
- Hardware acceleration: no
- Reason: the function contains calls to unsupported functions
- Additional info: function: 'save_ppm' call: [fopen]

**Maxeler**

Status: Template not supported.
Reason: the system does not provide a supported device, or matches the specified connection rules
Additional info: supported devices: board_Vectis

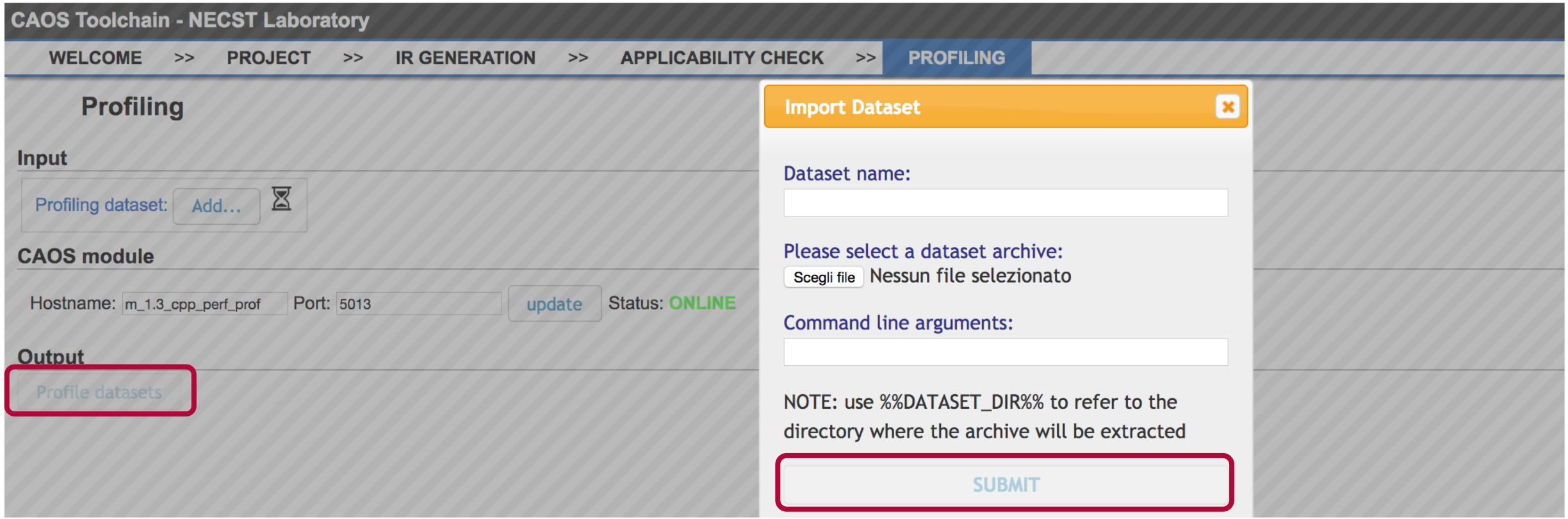
- Provides information on which functions are **suitable** for a **hardware acceleration**

# PROFILING




- Click “Add…” on *Profiling Dataset* to provide a dataset for the profiling phase:
  - Choose a *Name*
  - Upload an Archive containing the dataset (input files for the application)
  - Command line arguments necessary for the execution
- **N.B.** If the source code auto-generates the dataset, the archive and arguments might not be needed

# PROFILING




Click ***SUBMIT*** and ***Profile datasets*** to start code profiling

# PROFILING

**Output**

Profile datasets

**Dataset: NO_DATASET**

| Function | Self time % | Total time % |
|---|---|---|
| match_filter(unsigned char[1080][1440], unsigned char[1080][1440]) | 98.96% | 98.96% |
| main(int, char *[]) | 0.21% | 99.25% |
| Overall external calls | 0.84% | n.a. |

- Profiling result
  - Identifies the most **computationally intensive** functions of the application
  - Provides percentages regarding **self** and **total execution time**

# PARTITIONING

**CAOS Toolchain - NECST Laboratory**

WELCOME >> PROJECT >> IR GENERATION >> APPLICABILITY CHECK >> PROFILING >> PARTITIONING

## HW / SW partitioning

### CAOS module

Hostname: m_1.4_default Port: 5014 update Status: ONLINE

### Output

Run partitioning

| Function | Profiling Data | | | | | | HW / SW partitioning |
|---|---|---|---|---|---|---|---|
| | Self time % | | | Total time % | | | HW acceleration per architectural template |
| | min | avg | max | min | avg | max | MasterSalve |
| match_filter(unsigned char[1080][1440], unsigned char[1080][1440]) | 98.96% | 98.96% | 98.96% | 98.96% | 98.96% | 98.96% | - let partitioner decide - |
| main(int, char *[]) | 0.21% | 0.21% | 0.21% | 99.25% | 99.25% | 99.25% | Software |
| Overall external calls | 0.84% | 0.84% | 0.84% | n.a. | n.a. | n.a. | |

- For each function that is candidate for **Hardware Acceleration** it is possible to decide to *force* a hardware or software implementation, or to let CAOS decide

# PARTITIONING

## HW / SW partitioning

### CAOS module

Hostname: m_1.4_default Port: 5014 update Status: ONLINE

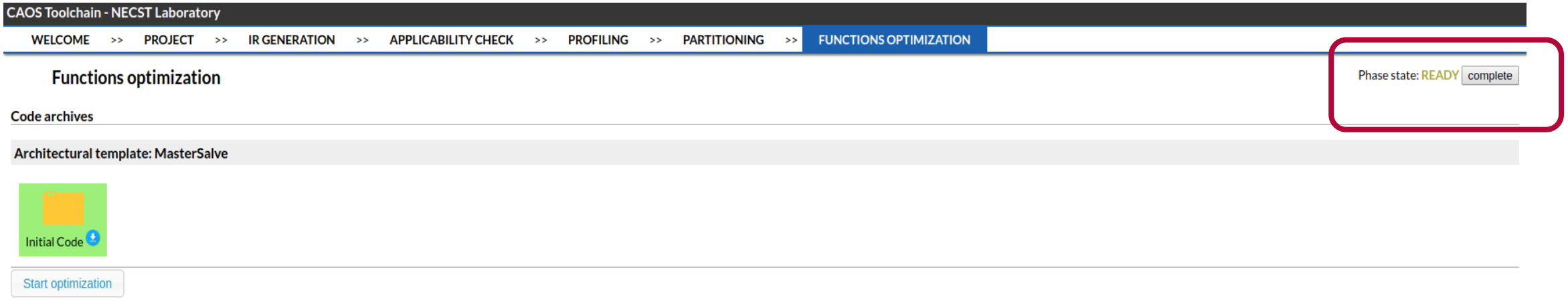
### Output

Run partitioning

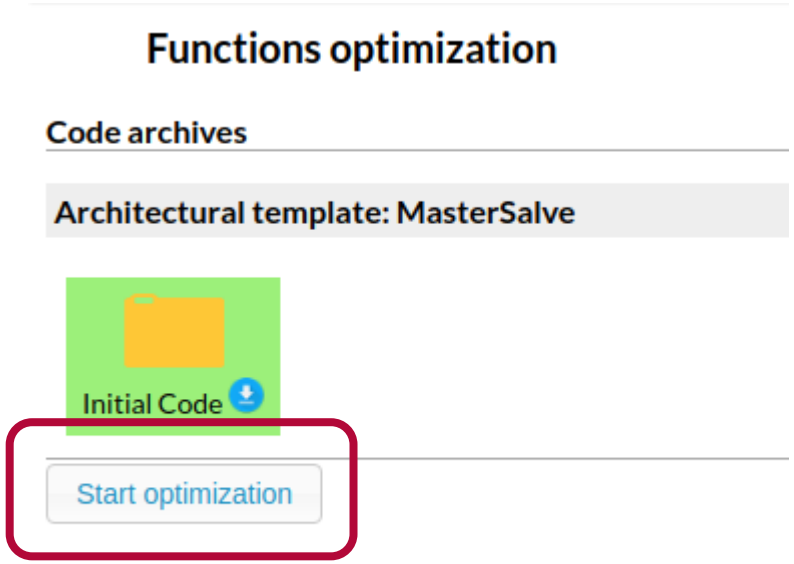
| Function | Profiling Data | | | | | | HW / SW partitioning |
|---|---|---|---|---|---|---|---|
| | Self time % | | | Total time % | | | HW acceleration per architectural template |
| | min | avg | max | min | avg | max | MasterSalve |
| match_filter(unsigned char[1080][1440], unsigned char[1080][1440]) | 98.96% | 98.96% | 98.96% | 98.96% | 98.96% | 98.96% | Hardware |
| main(int, char *[]) | 0.21% | 0.21% | 0.21% | 99.25% | 99.25% | 99.25% | Software |
| Overall external calls | 0.84% | 0.84% | 0.84% | n.a. | n.a. | n.a. | |

- After clicking *Run partitioning* CAOS automatically selects which functions to accelerate on hardware

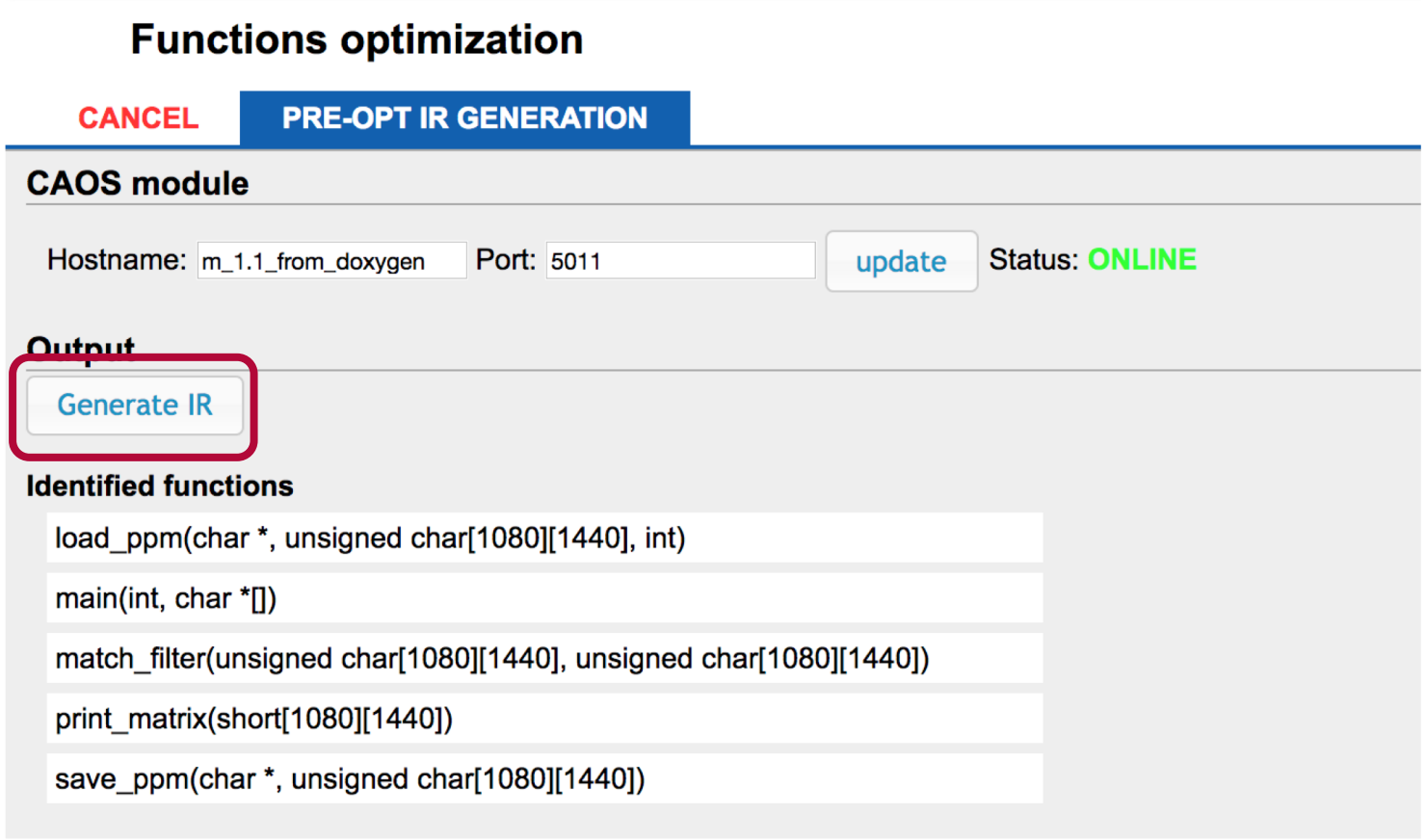
# FUNCTIONS OPTIMIZATION




- Within the **Functions Optimization** phase, we can specify the version of the code we want to work on by clicking on the corresponding folder
- Initially, only one version of the code is available
- After selecting the code version, we can either decide to:
  a) Click *complete* and move to the next phase
  b) Click on *Start Optimization* to start optimizing the code

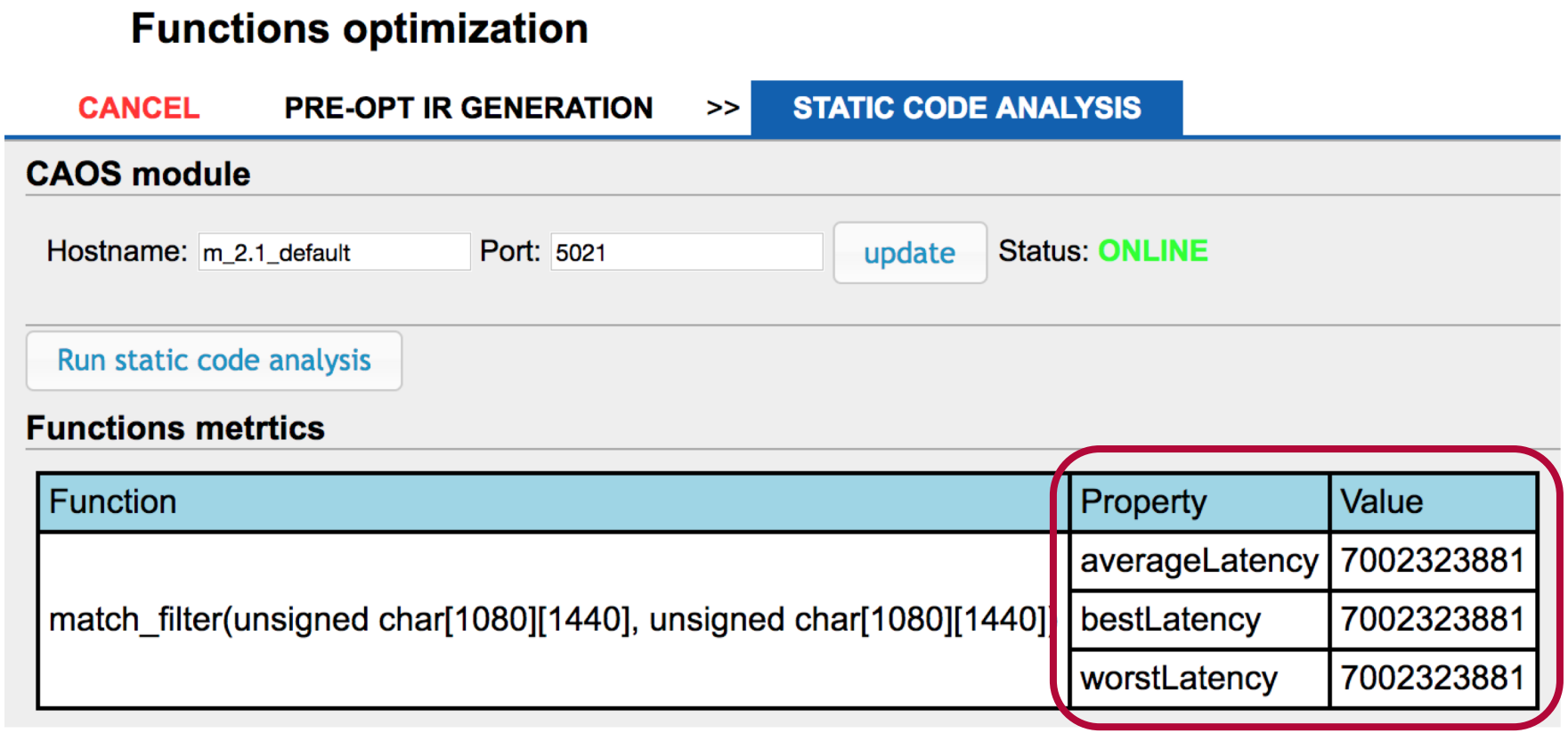


# FUNCTIONS OPTIMIZATION – Pre-Opt IR Generation




- During **Function Optimization** there are multiple sub-phases
- This first one, generates an *intermediate representation* of the code before optimizing it

# FUNCTIONS OPTIMIZATION – Static Code Analysis




- The **Static Code Analysis** phase provides a set of properties (such as expected latency) for all the functions that are candidates for hardware acceleration

# FUNCTIONS OPTIMIZATION – Hardware Estimation

**Functions optimization**

Phase state: **PROGRESS**

CANCEL | PRE-OPT IR GENERATION >> | STATIC CODE ANALYSIS >> | HARDWARE ESTIMATION

**CAOS module**

Phase state: **READY** complete

Hostname: m_2.2_default Port: 5022 update Status: **ONLINE**

Run hardware estimation

**Hardware resources estimation**

| Function | Device | Resource Type | Quantity |
|---|---|---|---|
| match_filter(unsigned char[1080][1440], unsigned char[1080][1440]) | f1-fpga | BRAM_18K | 3 (**0.07%**) |
| | | DSP48E | 7 (**0.10%**) |
| | | FF | 1024 (**0.04%**) |
| | | LUT | 1597 (**0.14%**) |
| | | URAM | 0 (**0.00%**) |

- By clicking on *Run hardware estimation* CAOS estimates the amount of **resources** needed for implementing the functions on the FPGA

# FUNCTIONS OPTIMIZATION – Performance Estimation

**Functions optimization**

Phase state: **PROGRESS**

CANCEL | PRE-OPT IR GENERATION >> | STATIC CODE ANALYSIS >> | HARDWARE ESTIMATION >> | **PERFORMANCE ESTIMATION**

**CAOS module**

Phase state: **READY** complete

Hostname: m_2.3_default Port: 5023 update Status: **ONLINE**

Run performance estimation

**Functions performance estimation and suggested optimization**

**Top function:** match_filter(unsigned char[1080][1440], unsigned char[1080][1440])

**Current function implementation**

**Performance estimation**
Execution time: 35.012 s
Clock frequency: 200 MHz
**Hardware estimation**

| | | |
|---|---|---|
| f1-fpga | BRAM_18K | 3 (**0.07%**) |
| | DSP48E | 7 (**0.10%**) |
| | FF | 1024 (**0.04%**) |
| | LUT | 1597 (**0.14%**) |
| | URAM | 0 (**0.00%**) |

**Optimization 1)**

**Optimization type:** on_chip_full_caching

**Performance estimation**
Execution time: 29.056 s
Clock frequency: 200 MHz
**Hardware estimation**

| | | |
|---|---|---|
| f1-fpga | BRAM_18K | 1523 (**35.25%**) |
| | DSP48E | 7 (**0.10%**) |
| | FF | 1211 (**0.05%**) |
| | LUT | 2211 (**0.19%**) |
| | URAM | 0 (**0.00%**) |

**Parameters:**

| | |
|---|---|
| fpga_to_host_copy: | [0,1] |
| host_to_fpga_copy: | [0,1] |

**Optimization 2)**

**Optimization type:** pipelining

**Performance estimation**
Execution time: 13.750 s
Clock frequency: 200 MHz
**Hardware estimation**

| | | |
|---|---|---|
| f1-fpga | BRAM_18K | 3 (**0.07%**) |
| | DSP48E | 7 (**0.10%**) |
| | FF | 1274 (**0.05%**) |
| | LUT | 1937 (**0.16%**) |
| | URAM | 0 (**0.00%**) |

**Parameters:**

| | |
|---|---|
| II: | 1 |
| relativeForLine: | 15 |

- After hardware estimation, CAOS estimates **performance** suggesting potential optimizations

# FUNCTIONS OPTIMIZATION – Code Optimization

**Functions optimization**

Phase state: **PROGRESS**

CANCEL | PRE-OPT IR GENERATION >> | STATIC CODE ANALYSIS >> | HARDWARE ESTIMATION >> | PERFORMANCE ESTIMATION >> | CODE OPTIMIZATION

**CAOS module**

Phase state: **READY** complete

Hostname: m_2.4_default Port: 5024 update Status: **ONLINE**

**Input**

| Function | Optimization |
|---|---|
| match_filter(unsigned char[1080][1440], unsigned char[1080][1440]) | Optimization 1 (on_chip_full_caching) |

**Output**

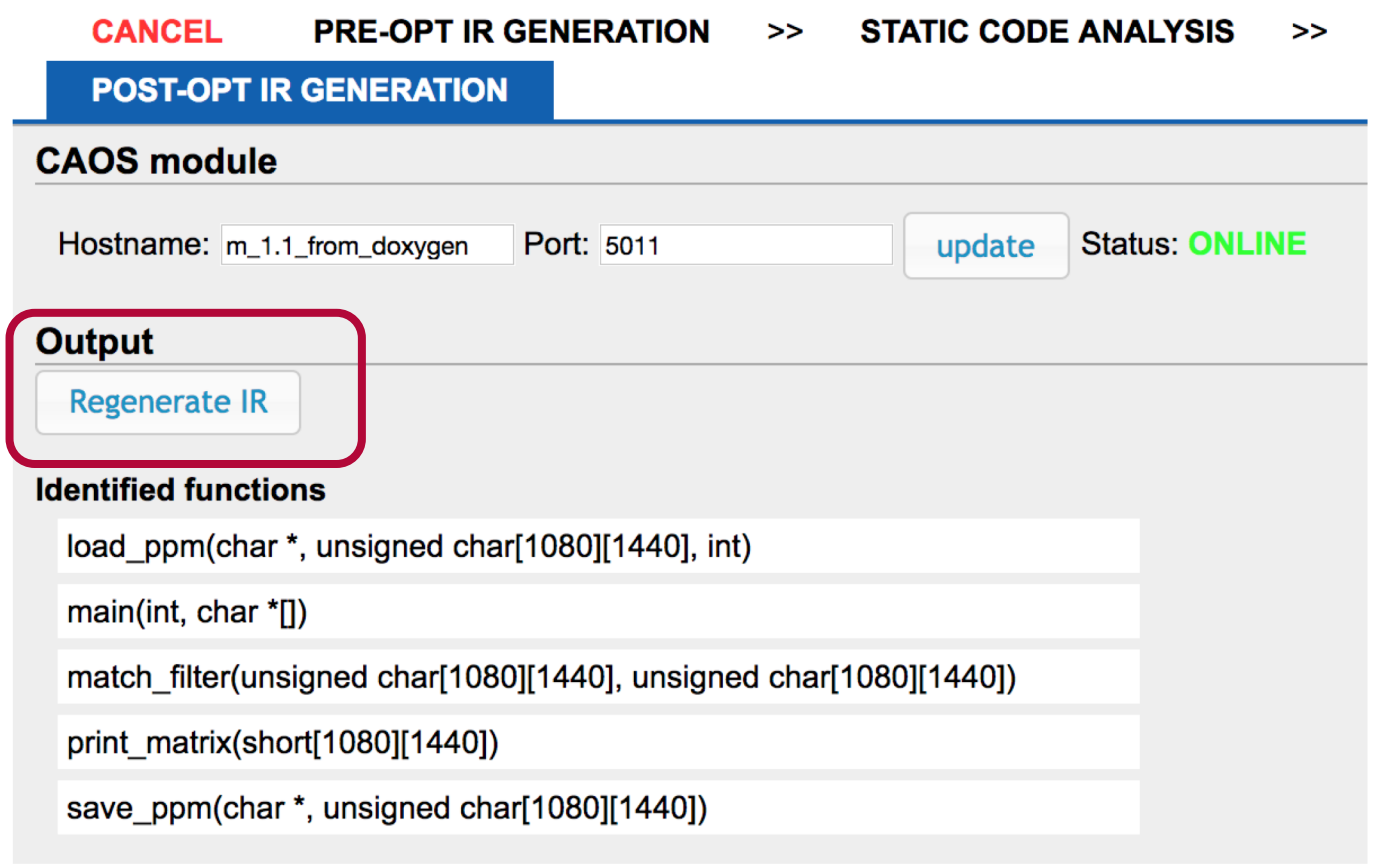
Apply code optimizations

**Optimization Report**

| function | optimization applied |
|---|---|
| load_ppm(char *, unsigned char[1080][1440], int) | NO |
| main(int, char *[]) | NO |
| match_filter(unsigned char[1080][1440], unsigned char[1080][1440]) | YES |
| print_matrix(short[1080][1440]) | NO |
| save_ppm(char *, unsigned char[1080][1440]) | NO |

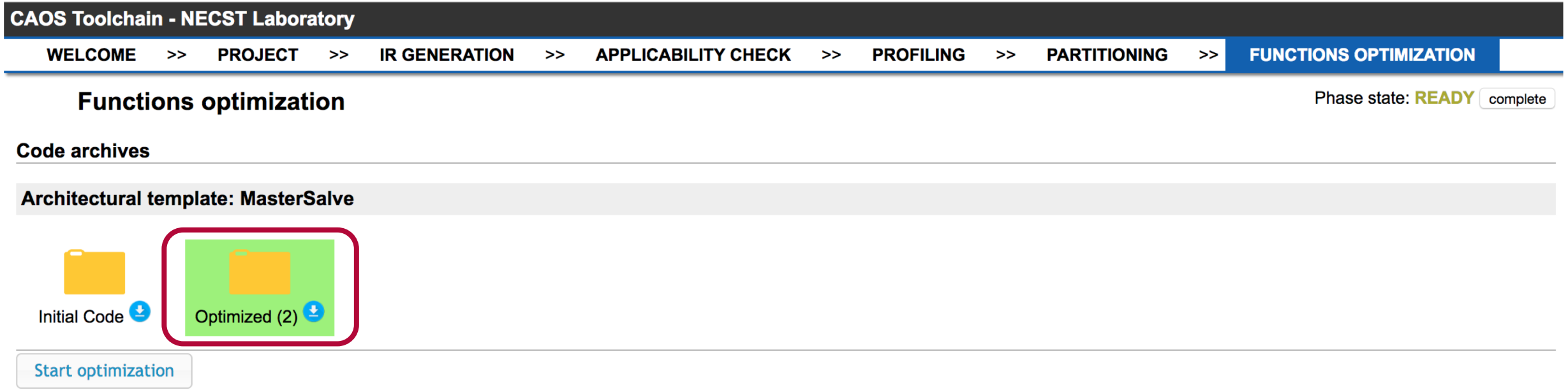
- Within **Code Optimization** sub-phase, it is possible to select among the suggested optimizations
- By clicking *Apply code optimizations* the code will be modified accordingly

# FUNCTIONS OPTIMIZATION – Post-Opt IR Generation



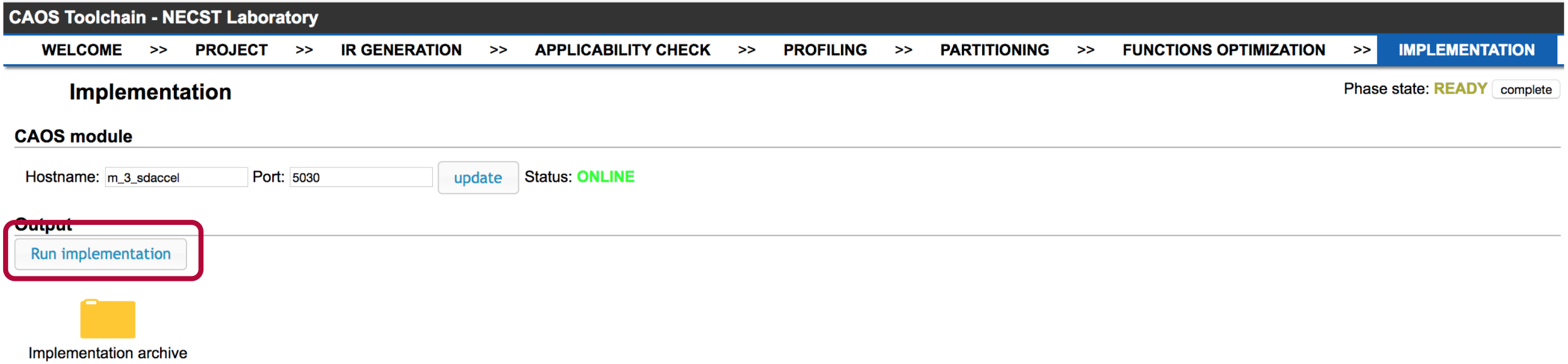

- After having applied one of the suggested code optimizations the IR might have changed
- The Post-Opt IR Generation sub-phase allows to regenerate the IR.

# FUNCTIONS OPTIMIZATION




- The optimization process is complete, we can either:
  - try to apply more optimizations by selecting the new code version and clicking on *Start Optimization*
  - select the desired code version and move to the implementation phase

# IMPLEMENTATION



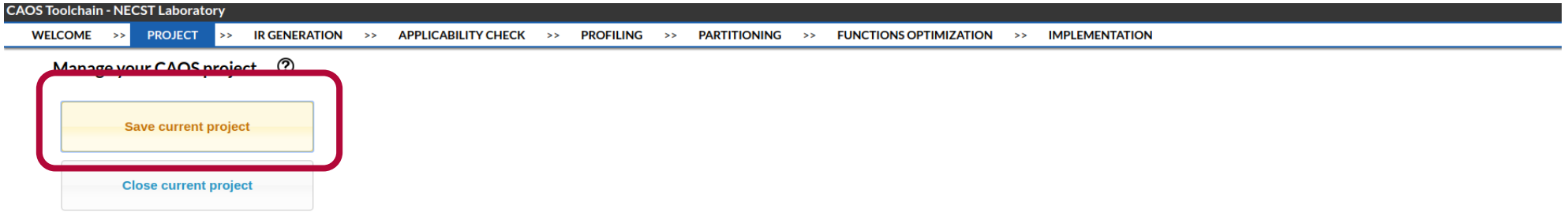

- During the **Implementation** phase CAOS will produce the host, kernel file and Makefile for SDAccel
- Once the phase is completed, click on the Implementation Archive to download the code
- After running the implementation, the output archive is also accessible on the server at:

```
~/Documents/CAOS_outputs
```

# Emulate / Build the final application

- Upload the CAOS archive to an AWS instance
  - `scp -i <pem file> <CAOS-ARCHIVE>.zip centos@<public_dns entry>:~/`
- SSH to the AWS Instance and load the SDAccel settings
  - `ssh -i <pem file> centos@<public_dns entry>`
  - `cd $AWS_FPGA_REPO_DIR`
  - `source sdaccel_setup.sh`
  - `source /opt/Xilinx/SDx/2017.1.op/settings64.sh`
- Unzip the CAOS archive
  - `cd ~/`
  - `unzip <CAOS-ARCHIVE>.zip`
  - `cd output`
- Use the Makefile to run HW / SW emulation or build the application
  - `make emulation TARGET=sw_emu`
  - `make emulation TARGET=hw_emu`
  - `make build TARGET=hw`

# Save the CAOS project




- CAOS flow is over, remember to save your project for future use, and then close it.