

Simulador HALCON

Ing. Leonardo Luis Ortiz

Instituto Balseiro
Centro Atómico Bariloche

11 de agosto de 2025

1 Simulador HALCON

2 Tools

3 Ejemplo

HALCON

DSP Simulation Engine

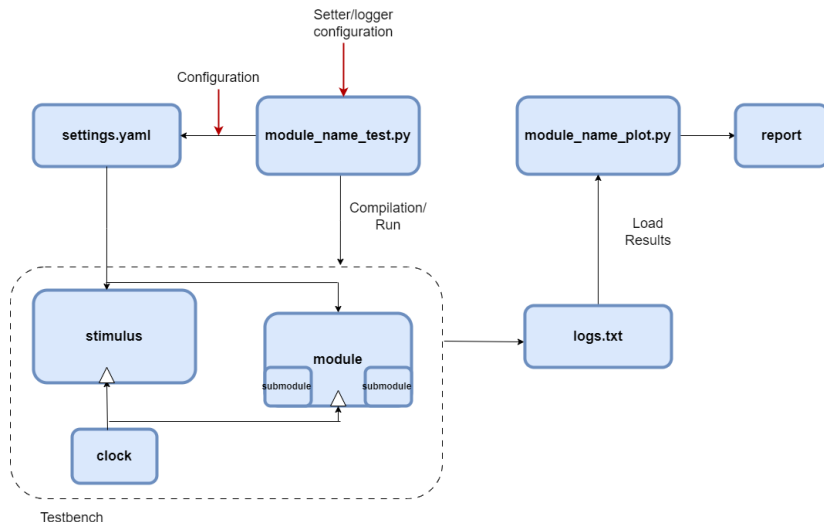


HALCON es un proyecto de código abierto desarrollado por Patricio Reus Merlo que consiste en un conjunto de herramientas desarrolladas en C++20 y Python3 que permiten a los usuarios diseñar y verificar arquitecturas de procesamiento digital de señales (DSP) desde las primeras etapas de diseño (alto nivel) hasta la implementación digital (RTL).

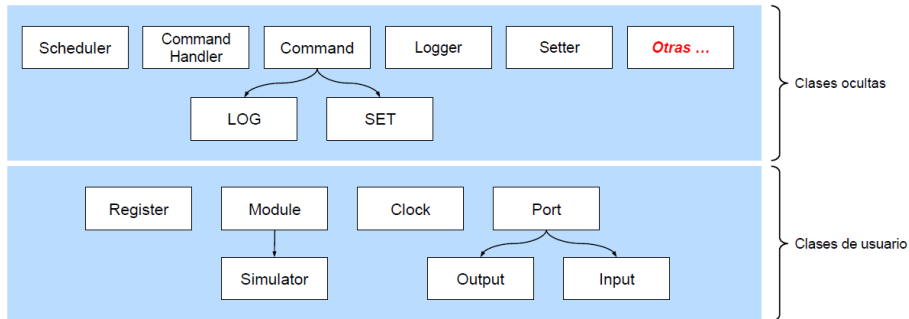
Entorno de desarrollo

- Lenguaje de programación: C++20
- Compilación: Cmake+gcc13 + Python
- Control de versiones: GitLab
- Entorno de simulación y test: Docker
- Automatización y scripting: Python
- Documentación de código: Doxygen
- Documentación funcional y tutoriales: Markdown+ LaTeX

Estructura de HALCON

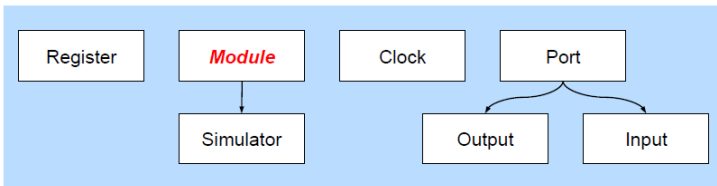


Estructura de HALCON

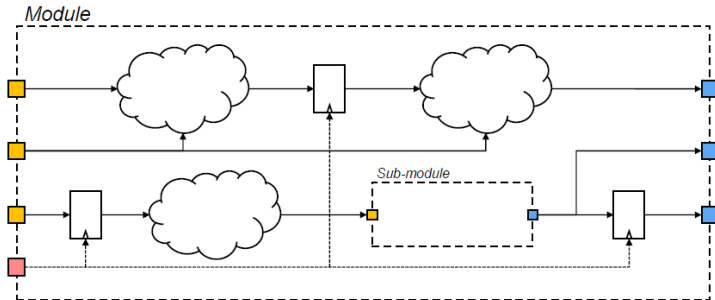


Module:

- Un módulo es un contenedor donde se describen algoritmos y/o modelos
- Puede describir estrictamente un bloque de DSP a nivel de RTL y a nivel de sistemas
- Puede contener modelos de componentes analógicos (ej. canal, ADC)
- Puede instanciar otros módulos en su interior (submódulos) y configurarlos
- Puede modelar lógica combinatorial y secuencial



Module:

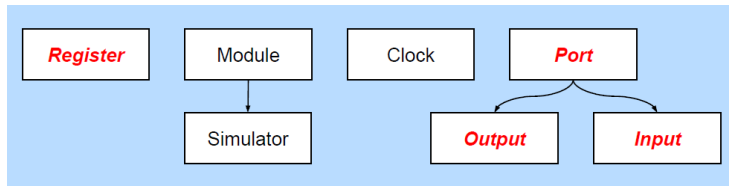


Register:

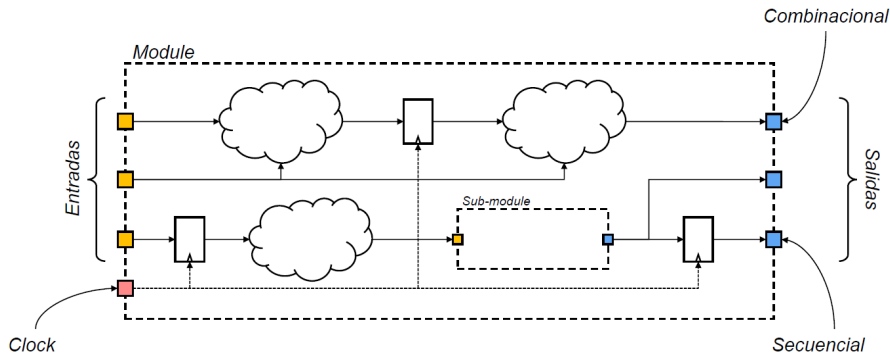
- Modela un registro de forma explícita y segura (evita registros implícitos)

Port:

- La interfase de un módulo con otros módulos se hace a través de puertos
- Según su comportamiento frente a un clock, se clasifican en secuenciales o combinacionales
- Según el flujo de la información, se clasifican como entradas o salidas

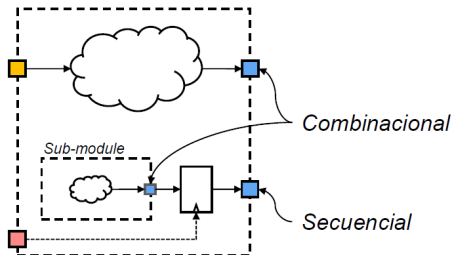


Ports y Register:



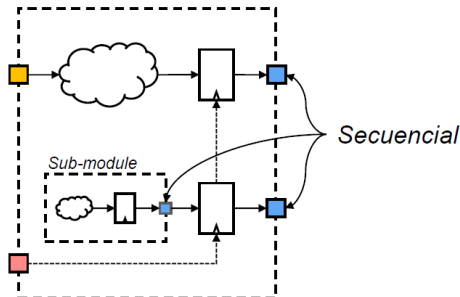
Puerto combinacioal

- Rastrear y ejecutan las funciones asociadas a ellos
- Aseguran que los valores de los datos estén actualizados



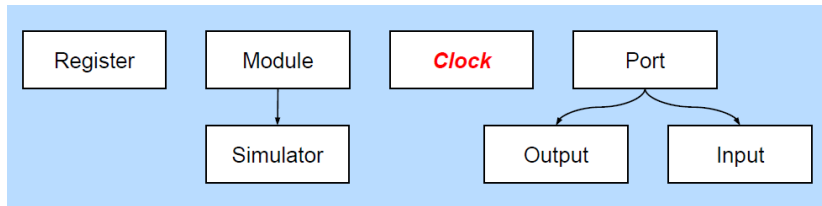
Puerto secuencial

- Lógica controlada por las secuencias de ejecución de reloj
- Los valores de los datos dependen de los registros asociados

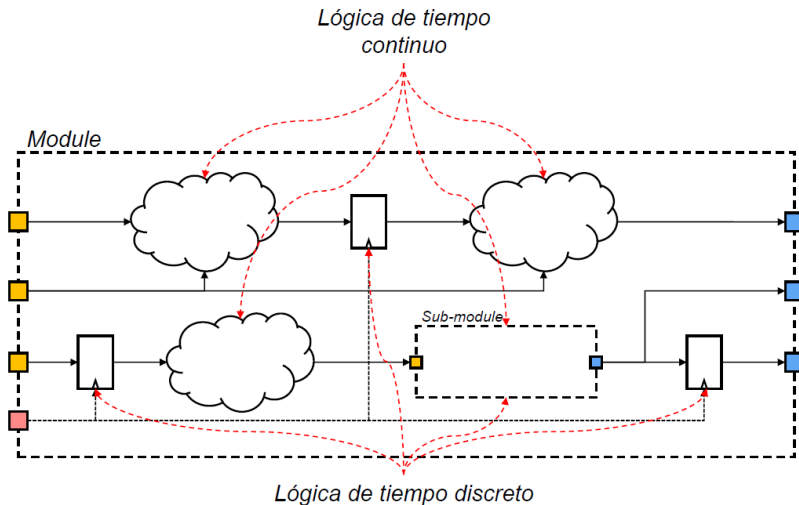


Clock:

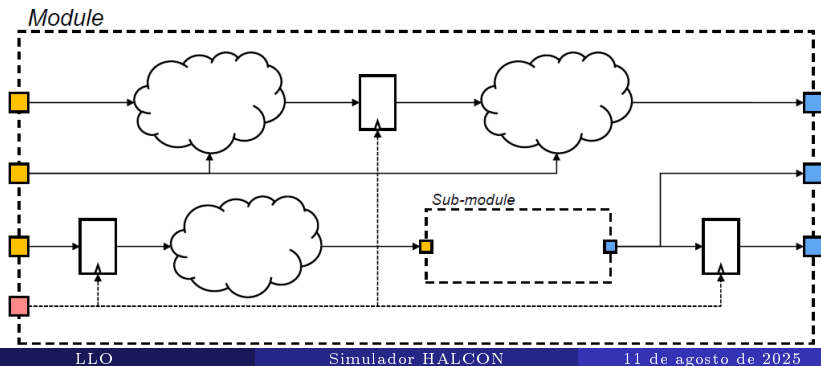
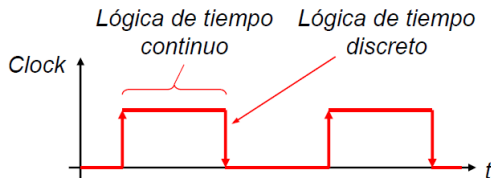
- Contiene un listado de módulos con sus respectivos registros que son sensibles a alguno de sus flancos
- Ejecuta la lógica de tiempo “continuo” y de tiempo “discreto” de los módulos asociados a él
- Lleva el conteo de tiempo de simulación



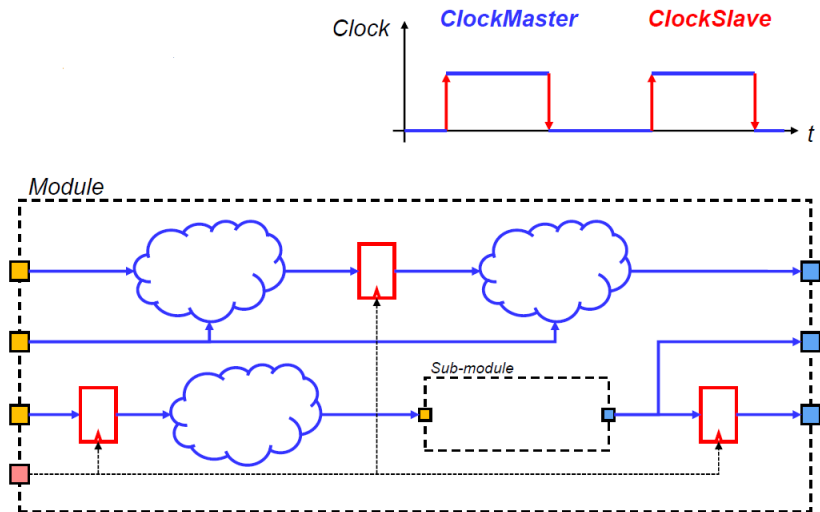
Clock:



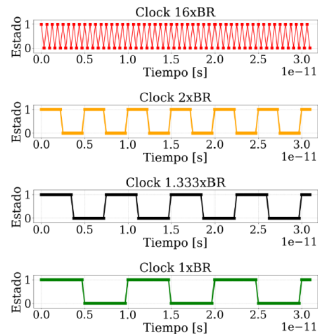
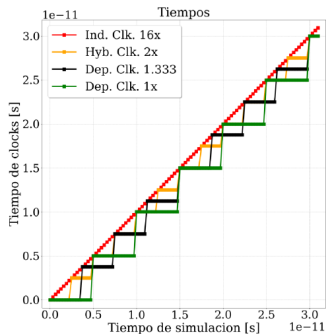
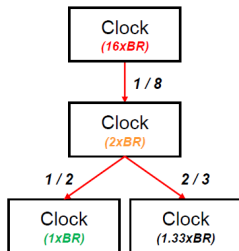
Clock:



Clock:



Clocks derivados



Clocks derivados

- Todos los relojes tienen 4 puertos:
 - `i_frequency_hz`: control de frecuencia
 - `i_phase_deg`: control de fase inicial
 - `i_division_factor_num`: num. del divisor de frecuencia
 - `i_division_factor_den`: den. del divisor de frecuencia
- El puerto `i_frequency_hz` de los clocks derivados se conecta automáticamente al puerto `i_frequency_hz` del clock del que se derivan.
- **IMPORTANTE:** luego de la derivación de un clock, no redefinir el puerto `i_frequency_hz` del clock derivado.
- Cada vez que se ejecuta un clock, éste se actualiza leyendo el valor de frecuencia de su puerto. A este puerto se puede conectar una señal variante en el tiempo.
- Todas las conexiones entre clocks y las derivaciones se deben hacer dentro de un método `Connect()` del módulo/simulador que contenga a los clocks.

```
/* Independent Clock */
clk_ch.i_frequency_hz << fs_ch;
clk_ch.i_phase_deg.SetData(0);
clk_ch.i_division_factor_num.SetData(1);
clk_ch.i_division_factor_den.SetData(1);

/* Derived Clocks */
clk_br_tx << clk_ch;
clk_br_tx.i_phase_deg.SetData(0);
clk_br_tx.i_division_factor_num << n_ovr;
clk_br_tx.i_division_factor_den.SetData(1);
```

Clock Independiente

Clock Derivado

Derivación

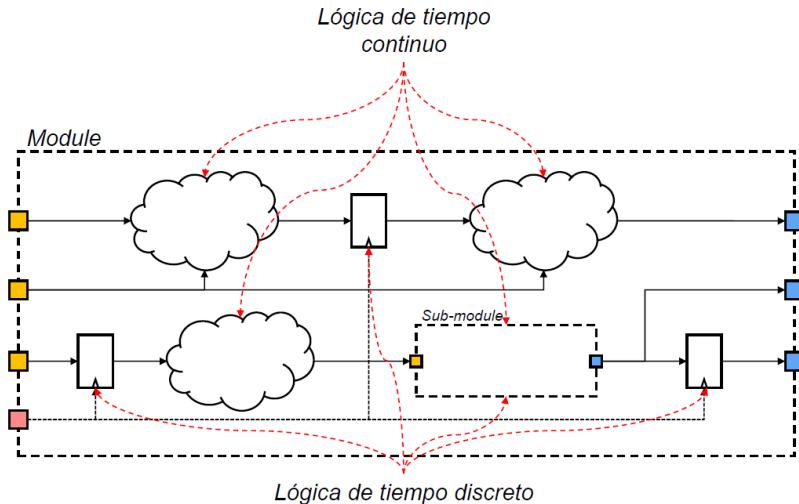
Clock Derivado

Clock Independiente

Inicialización con variable

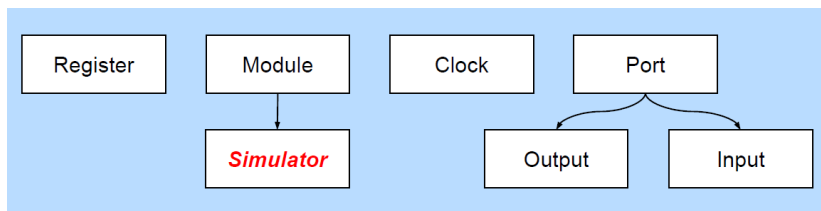
Inicialización con literal

Clock:



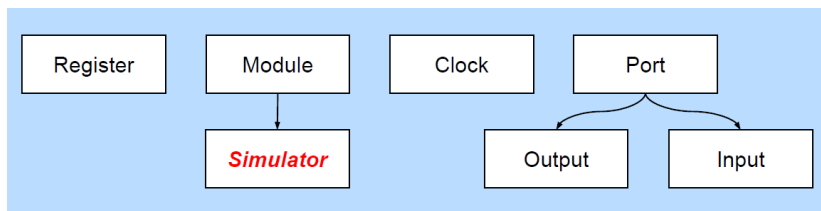
Simulator:

- Al derivar de módulo hereda todas sus propiedades. Es decir, es un contenedor de otros bloques (ej. módulos y clocks)
- Agrega bloques básicos que hacen evolucionar la simulación: Scheduler, Command Handler, Logger, Setter, etc
- Es el bloque de mayor jerarquía y por ende es el encargado de nombrar, configurar, conectar e inicializar todos los módulos y submódulos que contenga



Simulator:

- Al derivar de módulo hereda todas sus propiedades. Es decir, es un contenedor de otros bloques (ej. módulos y clocks)
- Agrega bloques básicos que hacen evolucionar la simulación: Scheduler, Command Handler, Logger, Setter, etc
- Es el bloque de mayor jerarquía y por ende es el encargado de nombrar, configurar, conectar e inicializar todos los módulos y submódulos que contenga



Simulator:

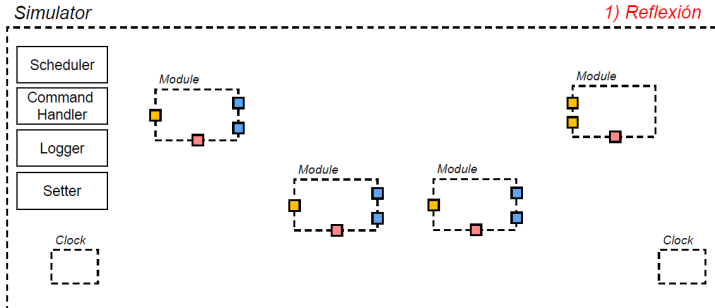
```
void Module::Initialize(YAML::Node& settings) ← Función de Startup  
{  
    NameRecursively();  
    ConnectRecursively();  
    ConfigureRecursively(settings);  
    CheckRecursively();  
    OptimizeRecursively();  
    InitRecursively();  
}
```

Simulator



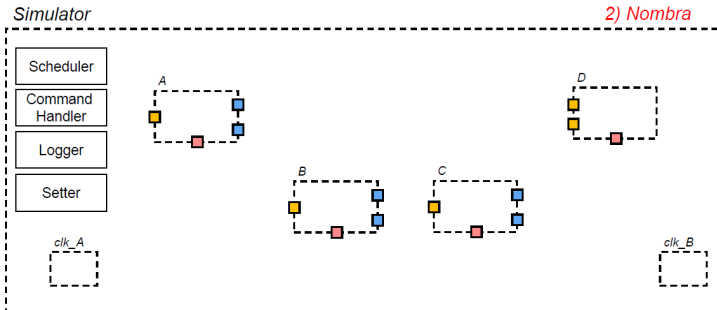
Simulator:

```
void Module::Initialize(YAML::Node& settings)
{
    NameRecursively(); ← Reflexión
    ConnectRecursively();
    ConfigureRecursively(settings);
    CheckRecursively();
    OptimizeRecursively();
    InitRecursively();
}
```



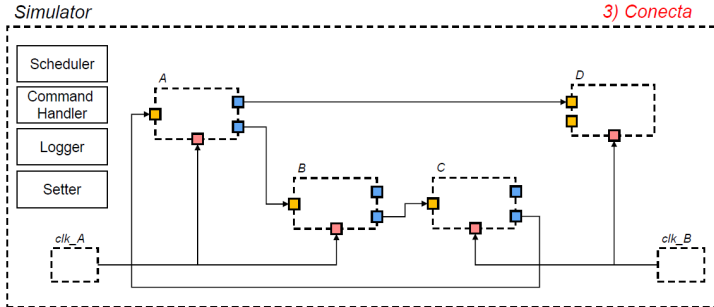
Simulator:

```
void Module::Initialize(YAML::Node& settings)
{
    NameRecursively(); ← Nombre
    ConnectRecursively();
    ConfigureRecursively(settings);
    CheckRecursively();
    OptimizeRecursively();
    InitRecursively();
}
```



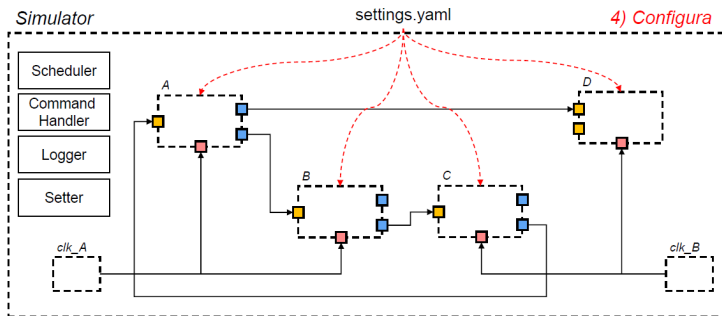
Simulator:

```
void Module::Initialize(YAML::Node& settings)
{
    NameRecursively();
    ConnectRecursively(); ← Conecta todos los
    ConfigureRecursively(settings);      módulos, submódulos y
    CheckRecursively();                  clocks
    OptimizeRecursively();
    InitRecursively();
}
```



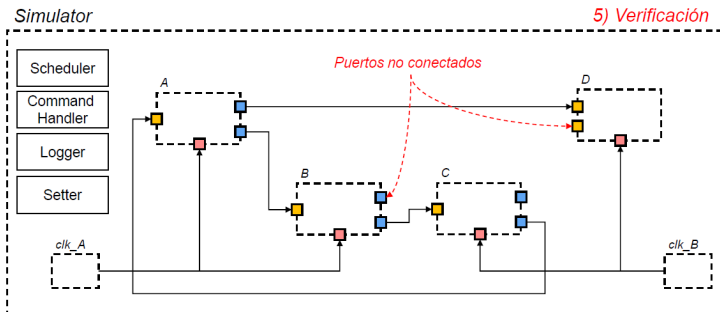
Simulator:

```
void Module::Initialize(YAML::Node& settings)
{
    NameRecursively();
    ConnectRecursively();
    ConfigureRecursively(settings); ← Configura bloques
    CheckRecursively();
    OptimizeRecursively();
    InitRecursively();
}
```



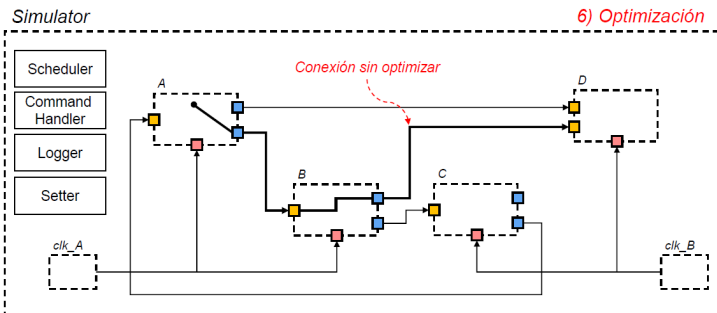
Simulator:

```
void Module::Initialize(YAML::Node& settings)
{
    NameRecursively();
    ConnectRecursively();
    ConfigureRecursively(settings);
    CheckRecursively(); ← Verifica si existen
                        puertos desconectados
    OptimizeRecursively();
    InitRecursively();
}
```



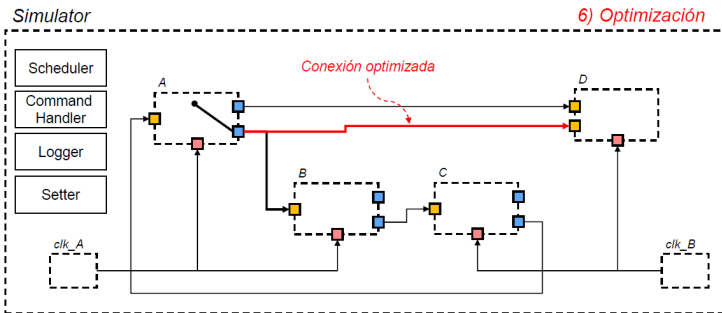
Simulator:

```
void Module::Initialize(YAML::Node& settings)
{
    NameRecursively();
    ConnectRecursively();
    ConfigureRecursively(settings);
    CheckRecursively();
    OptimizeRecursively(); ← Optimización de ruteo de puertos
    InitRecursively();
}
```



Simulator:

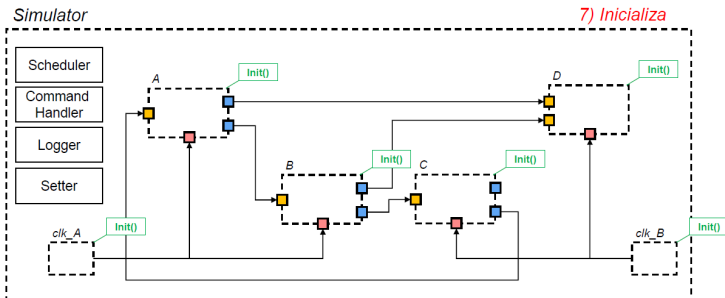
```
void Module::Initialize(YAML::Node& settings)
{
    NameRecursively();
    ConnectRecursively();
    ConfigureRecursively(settings);
    CheckRecursively();
    OptimizeRecursively(); ← Optimización de ruteo de puertos
    InitRecursively();
}
```



HALCON - Startup

Simulator:

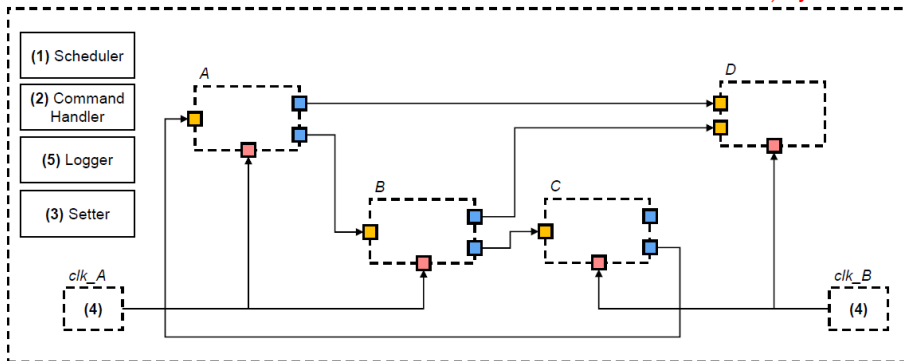
```
void Module::Initialize(YAML::Node& settings)
{
    NameRecursively();
    ConnectRecursively();
    ConfigureRecursively(settings);
    CheckRecursively();
    OptimizeRecursively();
    InitRecursively(); ← Inicialización
}
```



Simulator:

Simulator

7) Ejecución



HALCON - LOGs

- **signals**: señal de interés
- **clock**: reloj de muestreo
- **edge**: flancos de muestreo. Puede ser POSITIVE, NEGATIVE o BOTH
- **begin, step, end**: definen la ventana de muestreo y el subsample
- **file_type**: define el tipo de archive de salida. Puede ser TEXT o BINARY
- **file_name_type**: controla el nombre del archive. Puede ser SHORT o LONG
- **format**: solo afecta a los LOGs de datos de tipo ac_fixed, array ac_fixed, Port ac_fixed y Port array ac_fixed. Solo válido con archivos de texto. Las macros soportadas son DOUBLE, UINT y HEX

```
log_signals = [  
    # Clock  
    "root.clk",  
  
    # Shift Register  
    "root.u_shift_reg.i_signal",  
    "root.u_shift_reg.o_signal",  
    "root.u_shift_reg.o_shift_reg",  
  
    # Serial to Parallel  
    "root.u_ser_to_par.i_serial",  
    "root.u_ser_to_par.o_parallel",      # Port array ac_fixed  
    "root.u_ser_to_par.r_buffer.i",      # Array ac_fixed  
    "root.u_ser_to_par.r_buffer.o",      # Array ac_fixed  
  
    # Parallel to Serial  
    "root.u_par_to_ser.i_parallel",      # Port array ac_fixed  
    "root.u_par_to_ser.o_serial",  
    "root.u_par_to_ser.r_out.i"  
]  
  
log_clock = "root.clk"  
  
simh.add_log(signals = log_signals,  
             clock = log_clock,  
             edge = simh.POSITIVE,  
             begin = 0, step = 1, end = 0,  
             file_type = simh.TEXT,  
             file_name_type = simh.SHORT,  
             format = simh.UINT)
```


Nombre “SHORT”

Nombre “LONG”

```
logs
├── root.clk.txt
├── root.u_par_to_ser.i_parallel.txt
├── root.u_par_to_ser.o_serial.txt
├── root.u_par_to_ser.r_out.i.txt
├── root.u_ser_to_par.i_serial.txt
├── root.u_ser_to_par.o_parallel.txt
├── root.u_ser_to_par.r_buffer.i.txt
├── root.u_ser_to_par.r_buffer.o.txt
├── root.u_shift_reg.i_signal.txt
├── root.u_shift_reg.o_shift_reg.txt
└── root.u_shift_reg.o_signal.txt
```

```
logs
├── LOG_SIGNAL_root.clk_CLOCK_root.clk_BEGIN_0_END_last_STEP_1_TYPE_long_double_BYTES_16_SIZE_1.txt
├── LOG_SIGNAL_root.u_par_to_ser.i_parallel_CLOCK_root.clk_BEGIN_0_END_last_STEP_1_TYPE_double_BYTES_8_SIZE_10.txt
├── LOG_SIGNAL_root.u_par_to_ser.o_serial_CLOCK_root.clk_BEGIN_0_END_last_STEP_1_TYPE_double_BYTES_8_SIZE_1.txt
├── LOG_SIGNAL_root.u_par_to_ser.r_out.i_CLOCK_root.clk_BEGIN_0_END_last_STEP_1_TYPE_double_BYTES_8_SIZE_1.txt
├── LOG_SIGNAL_root.u_ser_to_par.i_serial_CLOCK_root.clk_BEGIN_0_END_last_STEP_1_TYPE_double_BYTES_8_SIZE_1.txt
├── LOG_SIGNAL_root.u_ser_to_par.o_parallel_CLOCK_root.clk_BEGIN_0_END_last_STEP_1_TYPE_double_BYTES_8_SIZE_10.txt
├── LOG_SIGNAL_root.u_ser_to_par.r_buffer.i_CLOCK_root.clk_BEGIN_0_END_last_STEP_1_TYPE_double_BYTES_8_SIZE_10.txt
├── LOG_SIGNAL_root.u_ser_to_par.r_buffer.o_CLOCK_root.clk_BEGIN_0_END_last_STEP_1_TYPE_double_BYTES_8_SIZE_10.txt
├── LOG_SIGNAL_root.u_shift_reg.i_signal_CLOCK_root.clk_BEGIN_0_END_last_STEP_1_TYPE_double_BYTES_8_SIZE_1.txt
├── LOG_SIGNAL_root.u_shift_reg.o_shift_reg_CLOCK_root.clk_BEGIN_0_END_last_STEP_1_TYPE_double_BYTES_8_SIZE_10.txt
└── LOG_SIGNAL_root.u_shift_reg.o_signal_CLOCK_root.clk_BEGIN_0_END_last_STEP_1_TYPE_double_BYTES_8_SIZE_1.txt
```

Barrido de parámetros: clases

```
#####  
# HALCON MODULES  
#####  
  
from halcon import Options  
from halcon import SimulatorHandler  
from halcon import Parameter  
from halcon import Case  
from halcon import Test  
from halcon import Processor
```

Barrido de parámetros: clases

```
#####  
# HALCON MODULES  
#####  
  
from halcon import Options  
from halcon import SimulatorHandler  
from halcon import Parameter  
from halcon import Case  
from halcon import Test  
from halcon import Processor
```

```
→ tests ./sweep.py -h
```

optional arguments:

-h, --help	show this help message and exit
-d, --delete	delete old compilation files and logs
-f, --compile_f	compile in FULL version
-p, --compile_p	compile in POSEGE version
-l, --enable_logs	enable compilation logs
-r, --run_sim	run simulator
-t, --run_tests	run simulator tests
-R, --run_cases	run all cases
-M, --run_missing	run missing cases
-A, --run_again	run all again
-K, --kill	kill running cases
-S, --summary	test summary
-F, --folders_list	case folders list
-L, --live	live_summary
-D, --delete_test	delete test folder
-P, --processing	run processing scripts

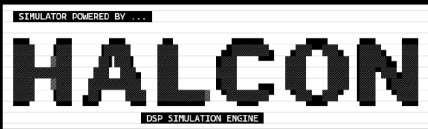
Simulador

Barrido

Barrido de parámetros: clases

```
#####  
# HALCON MODULES  
#####  
  
from halcon import Options  
from halcon import SimulatorHandler  
from halcon import Parameter  
from halcon import Case  
from halcon import Test  
from halcon import Processor
```

```
→ tests ./sweep.py -f
```



```
— Built as Release  
— Loading FULL HALCON core  
— Loading local module counter from /app/debug/tests/./src/counter  
— Loading local module parallel_to_serial from /app/debug/tests/./src/parallel_to_serial  
— Loading local module root from /app/debug/tests/./src/root  
— Loading local module serial_to_parallel from /app/debug/tests/./src/serial_to_parallel  
— Loading local module shift_register from /app/debug/tests/./src/shift_register  
— Loading local module sin_generator from /app/debug/tests/./src/sin_generator  
— Configuring done  
— Generating done  
— Build files have been written to: /app/debug/build  
Scanning dependencies of target local  
[ 11%] Building CXX object CMakeFiles/local.dir/src/parallel_to_serial/parallel_to_serial.cpp.o  
[ 22%] Building CXX object CMakeFiles/local.dir/src/root/root.cpp.o  
[ 33%] Building CXX object CMakeFiles/local.dir/src/counter/counter.cpp.o  
[ 44%] Building CXX object CMakeFiles/local.dir/src/serial_to_parallel/serial_to_parallel.cpp.o  
[ 55%] Building CXX object CMakeFiles/local.dir/src/sin_generator/sin_generator.cpp.o  
[ 66%] Building CXX object CMakeFiles/local.dir/src/shift_register/shift_register.cpp.o
```

Barrido de parámetros: clases

```
#####  
# HALCON MODULES  
#####  
  
from halcon import Options  
from halcon import SimulatorHandler  
from halcon import Parameter  
from halcon import Case  
from halcon import Test  
from halcon import Processor
```

```
p_sin_amp_v = Parameter (  
    name = 'root.u_sin_gen.amplitude_v',  
    text = 'sin_amp_{:.1f}_v',  
    alpha = 1,  
    value = np.linspace(1, 5, 3),  
    simh = simh  
)  
  
p_sin_freq_v = Parameter (  
    name = 'root.u_sin_gen.frequency_hz',  
    text = 'sin_freq_{:.1f}_khz',  
    alpha = 1 / 1e3,  
    value = np.linspace(1e3, 5e3, 20),  
    simh = simh  
)
```

Parametro a barrer

Patrón de **descripción**

Factor de escala

Valores por barrer
(definen el **hash**)

Barrido de parámetros: directorios

```

-- ebfc2de5e5c09c58 ← Caso sin ejecutar
|-- conf
|   |-- command.cmd
|   |-- settings.yaml
|-- run
|   |-- binary
|-- ec54425af1f6c25e ← Caso ejecutado
|   |-- conf
|   |-- command.cmd
|   |-- settings.yaml } ← Binario y configuración
|-- run
|   |-- binary
|   |-- logs
|       |-- SIGNAL_root.clk_CLOCK_root.clk_BEGIN_0_END_last_TYPE_long_double_BYTES_16.txt
|       |-- SIGNAL_root.u_par_to_ser.i_parallel_CLOCK_root.clk_BEGIN_0_END_last_TYPE_double_BYTES_8.txt
|       |-- SIGNAL_root.u_par_to_ser.o_serial_CLOCK_root.clk_BEGIN_0_END_last_TYPE_double_BYTES_8.txt
|       |-- SIGNAL_root.u_ser_to_par.i_serial_CLOCK_root.clk_BEGIN_0_END_last_TYPE_double_BYTES_8.txt
|       |-- SIGNAL_root.u_ser_to_par.r_buffer.i_CLOCK_root.clk_BEGIN_0_END_last_TYPE_double_BYTES_8.txt
|       |-- SIGNAL_root.u_ser_to_par.r_buffer.o_CLOCK_root.clk_BEGIN_0_END_last_TYPE_double_BYTES_8.txt
|       |-- SIGNAL_root.u_shift_reg.i_signal_CLOCK_root.clk_BEGIN_0_END_last_TYPE_double_BYTES_8.txt
|       |-- SIGNAL_root.u_shift_reg.o_shift_reg_CLOCK_root.clk_BEGIN_0_END_last_TYPE_double_BYTES_8.txt
|       |-- SIGNAL_root.u_shift_reg.o_signal_CLOCK_root.clk_BEGIN_0_END_last_TYPE_double_BYTES_8.txt
|-- pid.txt
|-- stderr.txt
|-- stdout.txt
|-- time.txt } ← Archivos de control

```

```

→ parameter_sweep ls
003434a133e8651e 392660c9207417cc 7f9fe75469b28960
05bfe8db8cfe3279 3ca567f6553d3c9c 896dc5f18f6eb5bd
09b20f2aad4b880e 43cc2dadf8664b1a 90d7ffb7f2df71a1
09f0889d40556c09 46c93f0323ea06f5 91b73cbe02a9f92d
1942130097afda6f 4f8438e6fd7ac8d9 91ca49a71c28bab8
1da0b98d925b92c8 57098e62fda6866d 96235786f2eb6e12
2421770f11e868c3 6af76e28d7c772bc 981ef78a5910efal
24b752ed9aa43dff 727049bf594ae5cd a6084f852216cffe
26e423c925bab258 75bc44b6440c8380 a92af93545b29427
2a3f1c53b36c9661 77e5dd4d092c8c08 b0486780e12bd34

```

Barrido de parámetros: clases

```
#####  
# HALCON MODULES  
#####  
  
from halcon import Options  
from halcon import SimulatorHandler  
from halcon import Parameter  
from halcon import Case  
from halcon import Test  
from halcon import Processor
```

```
#####  
# TEST  
#####  
  
test = Test (  
    name = "Parameter Sweep",  
    base_dir = "scratch/",  
    parameters = Parameter.list()  
)  
  
#####  
# CASES  
#####  
  
for p_sin_amp in p_sin_amp_v:  
    for p_sin_freq in p_sin_freq_v:  
        for p_sin_phase in p_sin_phase_v:  
  
            # Create Case  
            case = Case(sinh)  
  
            # Add Parameters  
            case.add(p_sin_amp)  
            case.add(p_sin_freq)  
            case.add(p_sin_phase)  
  
            # Add to Test  
            case = test.add(case)
```

Test

Parámetros

Caso

Barrido de parámetros: clases

```
#####
# HALCON MODULES
#####

from halcon import Options
from halcon import SimulatorHandler
from halcon import Parameter
from halcon import Case
from halcon import Test
from halcon import Processor ←
```

```
#####
# DATA PROCESSOR
#####

p = Processor(test.directory)

#####
# AXIS
#####

amp = p.axis("root.u_sin_gen.amplitude_v")
freq = p.axis("root.u_sin_gen.frequency_hz")
phase = p.axis("root.u_sin_gen.phase_deg")

#####
# PLOTS
#####

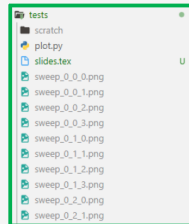
data = p.signals("root.u_par_to_ser.o_serial")

for i, a in enumerate(amp):
    for j, f in enumerate(freq):
        for k, p in enumerate(phase):

            y_v = data[i, j, k]["root.u_par_to_ser.o_serial"]['s'][100 : 200]
            n_v = data[i, j, k]["root.u_par_to_ser.o_serial"]['n'][100 : 200]

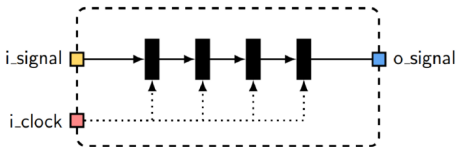
            tit = f"Amplitude {a} [V] - Frequency {f/1e3} [kHz] - Phase {p} [deg]"
            xlab = "Samples"
            ylab = "Amplitude [V]"

            ax1.plot(n_v, y_v)
            ax1.set_title(tit)
            ax1.set_xlabel(xlab)
            ax1.set_ylabel(ylab)
            plt.savefig(f'sweep_{i}_{j}_{k}.png')
            ax1.clear()
```



Ejemplo

HALCON - Registro de desplazamiento



```
#include "halcon.hpp"
```

Librería estática de C++

Los módulos del usuario heredan de module

```
template<typename T, size_t N>
class ShiftRegister : public Module
{
private:
```

```
    /* Register array */
    Register<double, N> r_array { 0 };
```

```
public:
```

```
    ShiftRegister();

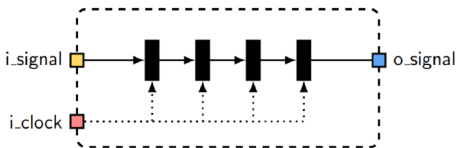
    /* Behavior */
    void Init() override;
    void Connect() override;
    void RunClockMaster() override;
```

La descripción de un módulo implica definir cuatro funciones

```
    /* Ports */
    Input<Clock> i_clock;
    Input<T> i_signal;
    Output<T> o_signal;
```

```
};
```

HALCON - Registro de desplazamiento



```
#include "halcon.hpp"

template<typename T, size_t N>
class ShiftRegister : public Module
{
private:
    /* Register array */
    Register<double, N> r_array { 0 };

public:
    ShiftRegister();

    /* Behavior */
    void Init() override;
    void Connect() override;
    void RunClockMaster() override;

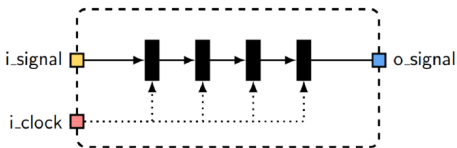
    /* Ports */
    Input<Clock> i_clock;
    Input<T> i_signal;
    Output<T> o_signal;
};
```

Librería estática de C++

Arreglo de registros

Los módulos se conectan mediante puertos de tipo Input y Output

HALCON - Registro de desplazamiento



```
template<typename T, size_t N>
ShiftRegister<T, N>::ShiftRegister()
```

```
{
    /* Registers */
    REFLECT(r_array);

    /* Ports */
    REFLECT(i_signal);
    REFLECT(o_signal);
}
```

```
template<typename T, size_t N>
void ShiftRegister<T, N>::Init()
{
    /* pass */
}
```

```
template<typename T, size_t N>
void ShiftRegister<T, N>::Connect()
{
    /* Register on clock positive edge */
    i_clock->RegisterOnPositiveEdge(this, r_array);

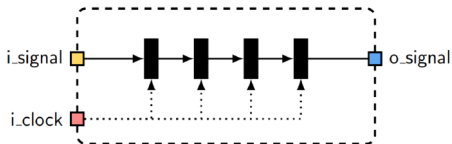
    /* Output Port */
    o_signal << r_array.o[N - 1];
}
```

*Define la sensibilidad
del arreglo de registros*

*Conecta el puerto de
salida a la salida del último
registro del arreglo*

```
template<typename T, size_t N>
void ShiftRegister<T, N>::RunClockMaster()
{
    for(size_t i { 1 }; i < N; ++i)
    {
        r_array.i[i] = r_array.o[i - 1];
    }
    r_array.i[0] = i_signal;
}
```

HALCON - Registro de desplazamiento



La propagación de señales de tiempo “discreto” se hace automáticamente

```
template<typename T, size_t N>
ShiftRegister<T, N>::ShiftRegister()
{
    /* Registers */
    REFLECT(r_array);

    /* Ports */
    REFLECT(i_signal);
    REFLECT(o_signal);
}

template<typename T, size_t N>
void ShiftRegister<T, N>::Init()
{
    /* pass */
}

template<typename T, size_t N>
void ShiftRegister<T, N>::Connect()
{
    /* Register on clock positive edge */
    i_clock->RegisterOnPositiveEdge(this, r_array);

    /* Output Port */
    o_signal << r_array.o[N - 1];
}

template<typename T, size_t N>
void ShiftRegister<T, N>::RunClockMaster()
{
    for(size_t i { 1 }; i < N; ++i)
    {
        r_array.i[i] = r_array.o[i - 1];
    }
    r_array.i[0] = i_signal;
}
```

Define la propagación de señales de tiempo “continuo”

HALCON - Registro de desplazamiento

```
class Root : public Simulator
{
public:
    Root();
private:

    /* User methods */
    void Init() override;
    void Connect() override;
    void Iteration() override;
    bool ContinueRunning() override;
    void Terminate() override;

    /* Variables */
    long double fs_hz { 100e3 };
    unsigned int n_samples { 100 };

    /* Clocks */
    Clock clk;

    /* Modules */
    SinGenerator u_sin_gen;
    ShiftRegister<double, 4> u_shift_reg;
};
```

*La descripción de un
simulador implica
definir cinco funciones*

```
void Simulator::Run()
```

```
{
```

```
    /* Global initialization */
    Initialize(settings_file);
```

*Init() y Connect() se
llaman al inicio de la
simulación*

```
    /* Main loop */
```

```
    do
```

```
    {
```

```
        scheduler.UpdateNextClocks();
        cmd_handler.Run(scheduler.next_clocks);
```

```
        if(cmd_handler.sets.size())
```

```
        {
```

```
            setter.Run(cmd_handler.sets);
```

```
        }
```

```
        scheduler.RunClocks();
```

```
        if(cmd_handler.logs.size())
```

```
        {
```

```
            logger.Run(cmd_handler.logs);
```

```
        }
```

*Iteration() se ejecuta en
todos los flancos de todos
los clocks del simulador*

```
        Iteration();
```

```
        iteration_counter++;
```

```
    } while (ContinueRunning());
```

*ContinueRunning() define el
fin de la simulación*

```
    /* End process */
```

```
    cmd_handler.Terminate();
```

```
    logger.Terminate(cmd_handler.logs, cmd_handler.flogs);
```

```
    Terminate();
```

*Terminate() se llaman al final
de la simulación*

HALCON - Registro de desplazamiento

```
class Root : public Simulator
{
public:
    Root();
private:

    /* User methods */
    void Init() override;
    void Connect() override;
    void Iteration() override;
    bool ContinueRunning() override;
    void Terminate() override;

    /* Variables */
    long double fs_hz { 100e3 };
    unsigned int n_samples { 100 };

    /* Clocks */
    Clock clk;

    /* Modules */
    SinGenerator u_sin_gen;
    ShiftRegister<double, 4> u_shift_reg;
};
```

Conexión de puertos e inicialización de variables de top level

```
void Root::Connect()
{
    /* Independent Clock */
    clk.i_frequency_hz << fs_hz;
    clk.i_phase_deg.SetData(0);
    clk.i_division_factor_num.SetData(1);
    clk.i_division_factor_den.SetData(1);

    /* Module ports */
    u_sin_gen.i_clock << clk;
    u_shift_reg.i_clock << clk;
    u_shift_reg.i_signal << u_sin_gen.o_sin;
};
```

```
void Root::Init()
{
    std::cout << "-- Simulation started\n";
}
```

HALCON - Registro de desplazamiento

```
class Root : public Simulator
{
public:
    Root();
private:

    /* User methods */
    void Init() override;
    void Connect() override;
    void Iteration() override;
    bool ContinueRunning() override;
    void Terminate() override;

    /* Variables */
    long double fs_hz { 100e3 };
    unsigned int n_samples { 100 };

    /* Clocks */
    Clock clk;

    /* Modules */
    SinGenerator u_sin_gen;
    ShiftRegister<double, 4> u_shift_reg;
};
```

```
void Root::Iteration()
{
    if(!(GetIterationCounter() % 100))
    {
        std::cout << "-- Iteration : "
                  << GetIterationCounter() << "/" << n_samples
                  << std::endl;
    }
}

bool Root::ContinueRunning()
{
    return GetIterationCounter() <= n_samples;
}

void Root::Terminate()
{
    std::cout << "-- Simulation finished\n";
}
```


Q&A