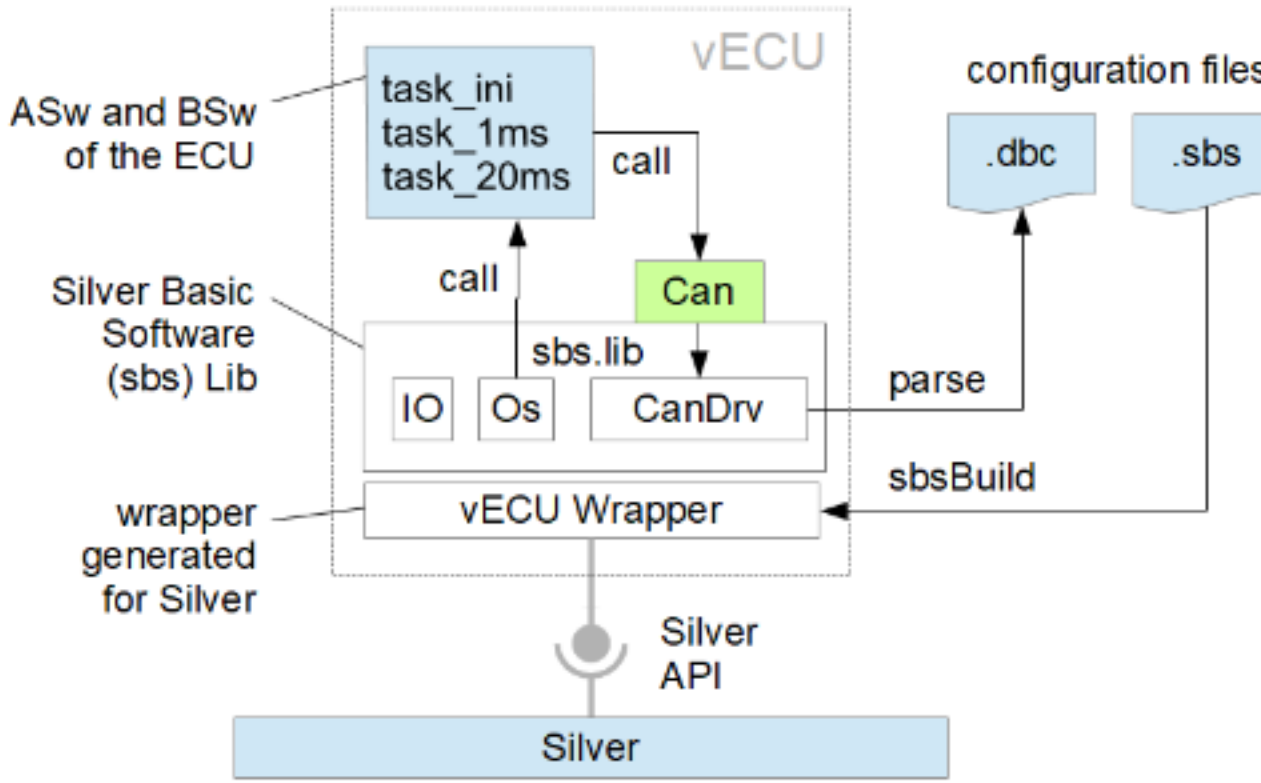
What's new in Silver 3.5

December 2017

AUTOSAR drivers for Silver

Silver 3.5 ships now sample AUTOSAR 4.2 drivers for Silver, as well as a demo that shows how to use these. Available drivers: **Can, Lin, Fr, Dio, Adc, Pwm, Spi, NvM, Gpt**. The drivers are implemented atop the **SBS** library (Silver Basic Software) and can be used to quickly implement **AUTOSAR** and **non-AUTOSAR** drivers for Silver. In **examples\AUTOSAR\BSw** see: **virtualECU\vecu.sil** and **doc\silver_mcal.pdf.**

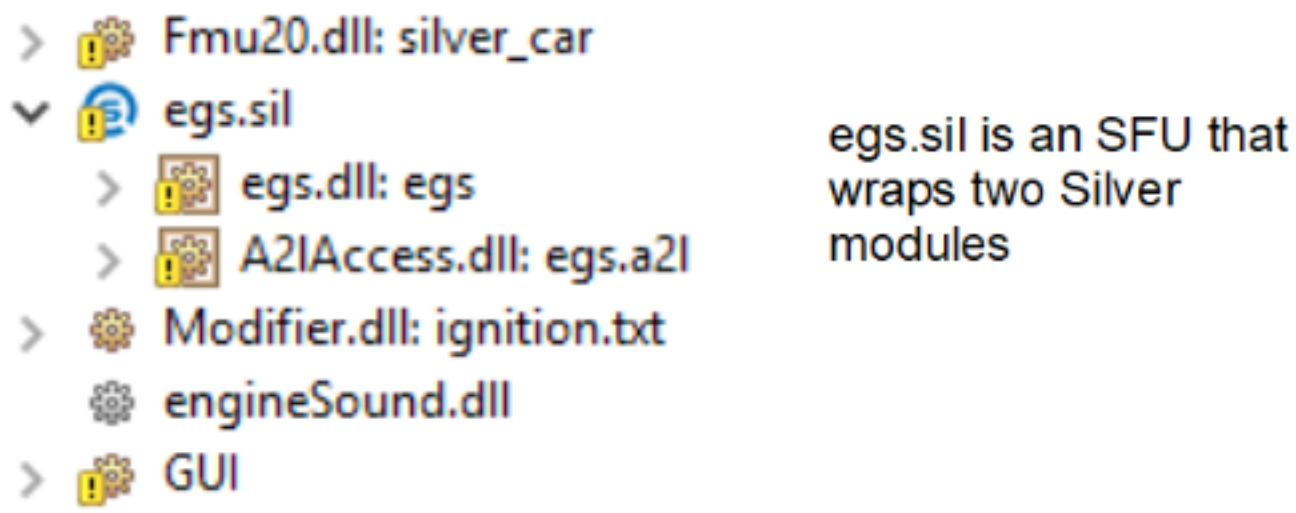


The figure above shows a **Can driver** (green box) implemented using the `sbs.lib` and called by the tasks of a vECU.

Silver Functional Unit (SFU)

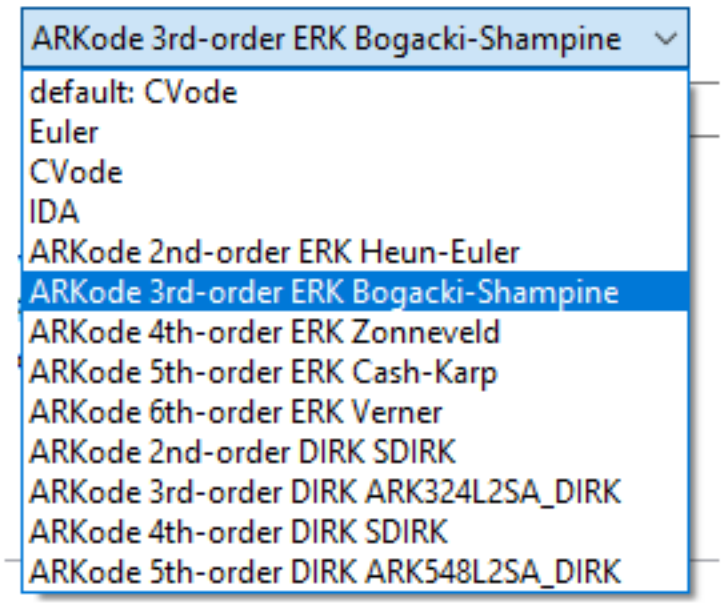
A Silver configuration (sil file) can now be composed from smaller sil files, called **Silver Functional Units** (SFUs). This enables the composition, exchange, reuse and replacement of standardized, independent SiL components and eases collaboration between departments delivering parts into a larger system simulation.
Furthermore, SFUs help to manage multiple simulation variants built from SFUs tailored to a specific engineering context, like calibration, test or stress testing.

A demo for this feature is avaiable in **examples\general\car\car_demo_sfu.sil**.
Current limitations: Adding SFUs to a setup is not yet supported from the GUI. Use the Silver command-line option `--sfu` instead. SFUs are not yet documented in the Silver user guide.



Runge-Kutta solvers for FMU and Sfunction

Silver 3.5 adds 9 implicit and explicit **Runge Kutta solvers**. The solvers can be applied to solve FMUs for model exchange (`.fmu`) and MATLAB/Simulink sfunctions (`.mexw32` and `.mexw64`).

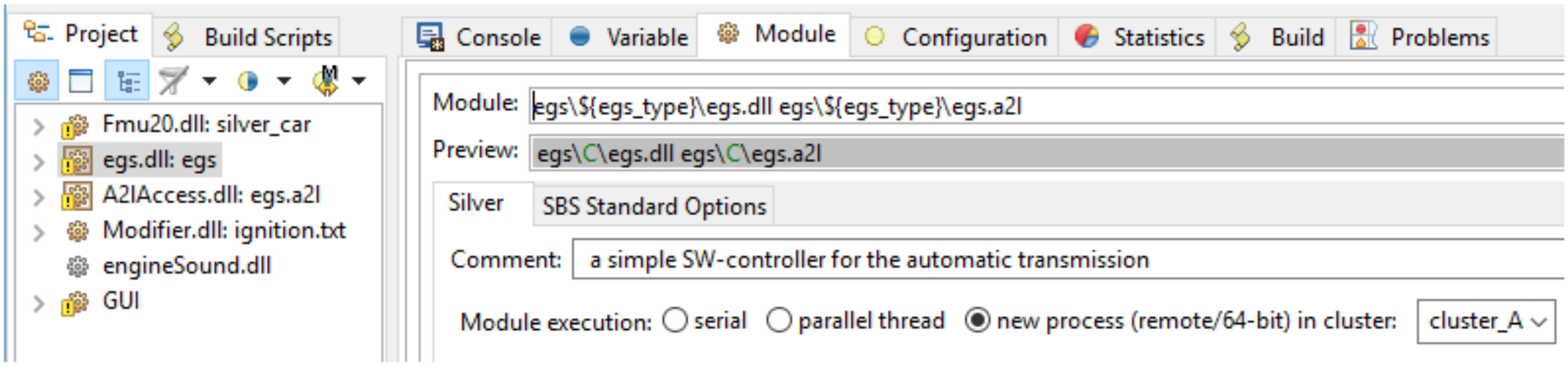


The solver **3rd-order explicit RK Bogacki-Shampine** corresponds to the popular **ode23** variable-step size solver available in MATLAB/Simulink.

Silver 3.5.0 - December 2017

Clusters of remote modules

Silver's **remote module** support allows to co-simulate modules that run in their own processes outside Silver. In previous versions each remote module created its own process. Starting with Silver 3.5.0, you can ask Silver to run several remote modules in the same 32-bit or 64-bit external process. This feature allows remote modules to **share and access the same memory**. This is required by modules like `A2lAccess` and `XCP`, and lately, by FMUs that require shared memory access for exchanging large binary data structures.



The above image illustrates two remote modules `egs.dll` and `A2lAccess.dll` that will be executed in the same process, called `cluster_A` here.

Silver 3.5.0 - December 2017

Silver generates C-Code FMUs

Building binary vECUs requires selecting the target execution platform when the vECU/FMU is generated. C-Code FMUs, however, can be imported on any platform because the compilation step is deferred to the import time.

Virtual ECUs created with the new `sbsBuild` option `-t fmuCCode` include its C sources and can therfore be executed on platforms such as Linux, Debian or real-time platforms such as dSPACE Scalexio. A virtual ECU built this way can still use `a2l` measurements and characteristics as inputs and outputs. However, virtual busses (Can, Lin, FlexRay, Eth) and communication via XCP is not yet fully supported by the C-code vECUs.

Silver's modules `fmu20` and `fmu20cs` can now also execute C-code FMUs: If an FMU does not contain a Windows binary, but it contains C source code, Silver attempts to build the binary on the fly, following the conventions defined by the FMI specification.

Silver 3.5.0 - December 2017

Support for Automotive Ethernet

Silver supports since many years **CAN** and **Lin** networks for vECUs. Recently we have added **Flexray** and **CAN-FD**. Starting with Silver 3.5.0, we support also **Automotive Ethernet**.

- In the `.sbs` file passed to `sbsBuild`, you can now use `add_ethernet(...)` to add an Ethernet bus to a virtual ECU based on specified `.arxml` files.
- The **SBS** library now contains functions to implement an AUTOSAR **Eth** driver. See the header file `SILVER_HOME/include/sbs.h` for more details.

Multi-core support, infinite tasks, timeouts

When creating a virtual ECU with `sbsBuild`, you can now optionally define on which cores the tasks will be executed. For example:

```
task_periodic(task10ms, 0.01, 10, 2)
```

will run the `task10ms` on core number 2. Tasks of the same time slice will be executed in their own thread.

The new `sbsBuild` command `task_infinite` allows you to start a task/function that the scheduler will not block for. Such tasks can be used to service low priority background jobs, such as emulating an on-chip system clock. The task should call `SBS_GetTime()` to synchonize with the Silver time.

Last but not least, you can use the new `sbsBuild` command `timeout` to specify a timeout for each core - to prevent tasks from blocking the simulation forever. For instance: you could use a virtual core to run a watchdog task. If that fails to return within a reasonable time, the timeout will stop that watchdog and, depending on your choice, you can continue or exit the simulation.

Silver 3.5.0 - December 2017

Input Bypass

Easily connect tasks with alternative inputs provided by bypass functions using new `sbs.lib` and `sbsBuild` features.

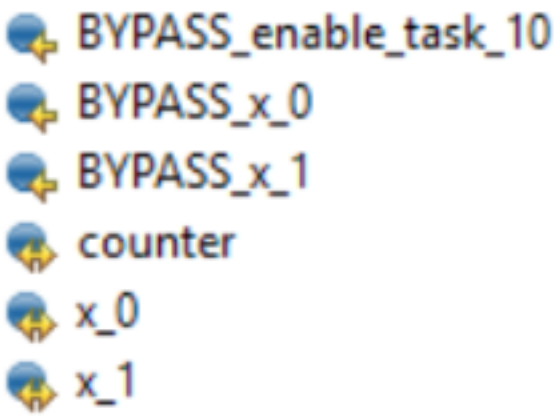
You can now specify for a task alternative inputs, called **bypass inputs**. These can be used, depending on a control signal, instead of the task's inputs declared by the ECU description. This greatly simplifies testing with several function variants. Which function and which variants can be selected at execution time - without recompiling the vECU.

Example

The a2l file declares a function task_10 with inputs x_0 and x_1.
The sbs file used to generate the vECU contains commands

```
function_periodic(task_10, 0.01, 10)
input_bypass(task_10)
```

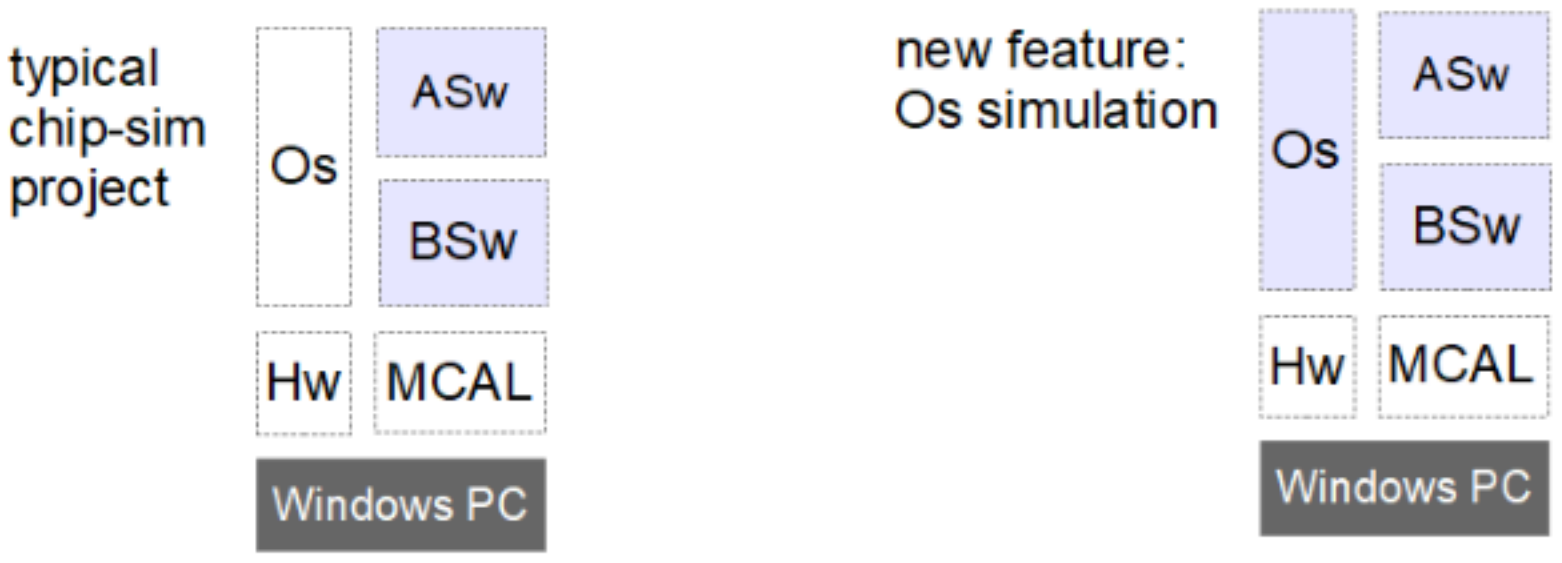
The resulting vECU contains inputs and outputs shown right.
If BYPASS_enable_task_10 = 1 at runtime, task_10 uses BYPASS_x_0, BYPASS_x_1 as inputs, otherwise x_0 and x_1



Executing the original Os with Silver's chip simulation

The `scs` configuration language, used to configure the chip simulation, includes now commands to run the original **Operating System** (Os) from a given hex file.

Before 3.5, Silver's built-in Os emulation had to be used to run the tasks from a hex file. The new feature allows to study the original Os, for instance, the effects of pre-emptive multitasking. The price paid for the detailed OS simulation is a significantly slower execution time.



New commands to configure Os simulation in scs file

```
// specify the code to run
task_loop(Core0_main, 18000) // run 18000 instructions per Silver macro step
plugin(chipHw.dll) // replacement functions for Silver and callback function
callback(updateChipHardware) // increment system clock, start cores and pending ISRs
```

for a complete list of changes please consult the Release Notes for Silver 3.5.0
http://www.qtronic.de