Hardware/Software Co-Simulation of SPI enabled ASICs and Software Drivers for Fault Injection and Regression Tests

Elias Kyrlies-Chrysoulidis, Andreas Plange, Thomas Gürtler, Matthias Auerswald

Continental AG, Division Powertrain, Nuremberg, Germany

{Elias.Kyrlies-Chrysoulidis|Andreas.Plange|Thomas.Guertler|Matthias.Auerswald}
@continental-corporation.com

Josef Schmid, iSyst GmbH, Nuremberg, Germany

Josef.Schmid@iSyst.de

***Abstract*—In traditional design methodology, hardware and software development takes place in isolation. Hardware-software integration only happens after the hardware is fabricated. Bugs that cannot be fixed in software lead to costly re-fabrication and can adversely affect time-to-market. In safety critical embedded systems additional safety requirements exist. Particularly in the field of automotive electronics, the increasing system complexity, along with the safety levels required by the OEMs, increase the risk of product recalls due to safety weaknesses. Hardware/software co-simulation concepts are already used in the industry. While several solutions for co-simulation exist, no solution is oriented towards automotive development, where standardized software environment is used (AUTOSAR) and safety requirements (ISO 26262) demand early verification of hardware software interface (HSI) on system design level and fault injection tests, both of which are very much facilitated by our co-simulation solution.**

Keywords*—HW/SW Co-simulation; HW/SW Co-verification; AUTOSAR; ISO 26262; FSM; Functional Safety; HSI; HW/SW-Interface; FMEA; Fault insertion & analysis; DFSS; Design for Six Sigma; virtual HW models; VHDL-AMS

I. Introduction and Motivation

The HW/SW Co-simulation concept has been widely used in the embedded systems development due to its key benefits: (a) Shorter project design time (i.e. time-to-market), thanks to the concurrent development and early integration of hardware and software. (b) Improved design quality and reduction of development costs, thanks to the increased number of test-cases, which reduce costly ASIC re-spins. In the automotive industry, increasing safety levels requested from OEMs (ISO 26262 standard up to ASIL-D, [4]) and the complexity of HW/SW automotive systems, enlarge the risk of product recalls due to safety weaknesses.

Our solution [1] offers a timed hardware software co–simulation system for automotive SPI enabled ASICs and software drivers, which achieves a sufficient tradeoff between accuracy (in terms of simulation time granularity) and performance (in terms of total execution time). The system employs an AUTOSAR compliant software simulation environment connected via TCP/IP to a hardware simulation platform (Saber [6] and Modelsim [7] Co-simulation [3]) where a VHDL-AMS model [5] of the ASIC and the module environment is evaluated. The safety concept is also incorporated. The concepts of early HSI verification (e.g. SPI) in system design level as well as fault injection tests [2] to improve the test coverage of the safety requirements during different development phases are two discussion topics of the ISO 26262 which are addressed by our method. To illustrate the approach, a case study using an existing ASIC (solenoid driver) is presented. The component has strong safety requirements and specific safety goals. Target of the simulation is to test the safety requirements defined by the FMEA (Failure Mode and Effects Analysis) by triggering faults to achieve safe states.

The employed solution is shortly described in *Section II*. The introduction of the safety concept in simulation is discussed in *Section III*. We discuss our approach on the case study in *Section V*. Finally *Section VI* includes a summary of the methodology presented in this paper along with a short reference to the adaptation of the method to future development.

II. Method Description

During development, driver code is executed natively on a workstation. When the driver code requests for interaction with the hardware (SPI or DIO), an appropriate command is transmitted to the hardware simulation via TCP/IP. Native code execution allows to achieve simulation acceleration, because the software is running at workstation speed. However the overall performance is dominated by hardware simulation speed.

Synchronization of the hardware and software domain is only performed on each SPI transmission. During the rest of the execution the clocks of the hardware and software domain run independently. This scheme offers small synchronization overhead in the overall simulation time, while being adequate for the requirements of the ASIC and driver development.

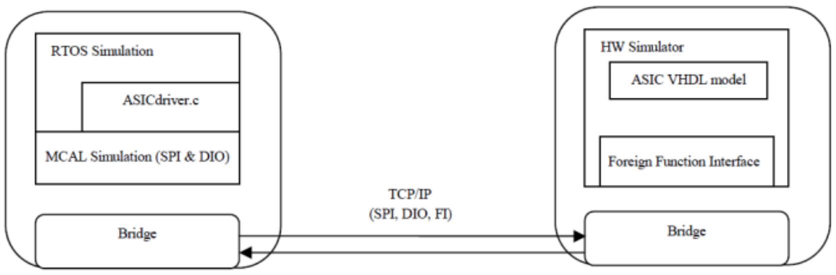


Figure 1. Overview of hardware software co-simulation system.

With reference to Figure 1, the proposed architecture is split into two parts, one for the machine is running native software and another is running the hardware simulation. These two processes are kept independent of each other by the BSD socket library (TCP/IP). The bridge software is deployed on both sides for that purpose. The HW machine behaves as a server. If a client is connected the server bridge monitors the assigned socket waiting for hardware stimuli. The SW machine behaves as a client. The client bridge forwards SPI and DIO peripheral stimuli requests from the SW driver to the server. This stimulus is read by the hardware simulation via the foreign function interface (FFI) of the simulation environment. The stimulus is then evaluated by the hardware simulation and the response is sent back to the software simulation.

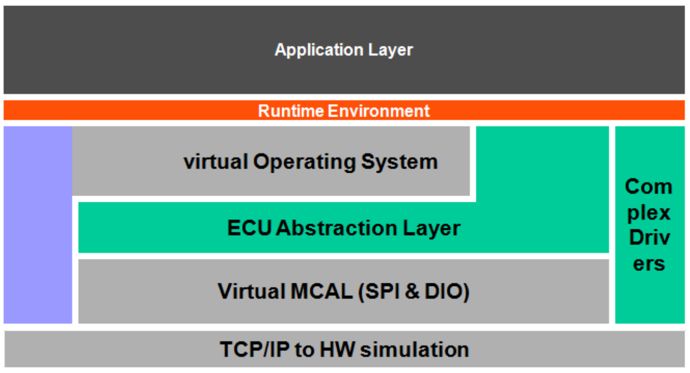


Figure 2. Simulation of AUTOSAR layers in the software domain.

The AUTOSAR software architecture is simulated on a PC workstation (Figure 2). The MCAL (Microcontroller Abstraction Layer) and the OS part of the Service layer (RTOS and MCAL) are simulated using a commercial C++ framework on the software side. The virtual MCAL layer offers the same API for the microcontroller peripherals, with modified internal functionality, so that instead of accessing the hardware directly, the functions forward stimulus requests to the hardware simulation. The virtual OS offers timely accurate task execution and interrupt handling. The ASIC driver runs as native C code on the ECU Abstraction layer. Using this architecture, rapid transition of the ECU driver module from simulation to real hardware prototyping can be achieved.

The ASIC VHDL-AMS model is evaluated with a hardware simulation platform and communicates with the software simulation via a bus functional model (BFM) of the microcontroller. This model simulates the pin behavior of the processor, acting as a communication module between the HW and the SW simulation. The foreign function interface (FFI) of the hardware simulation platform is used for receiving and transmitting software commands via a well defined protocol. Apart from peripheral commands (SPI and DIO) fault injection (FI) commands, that activate faults, can also be transmitted.

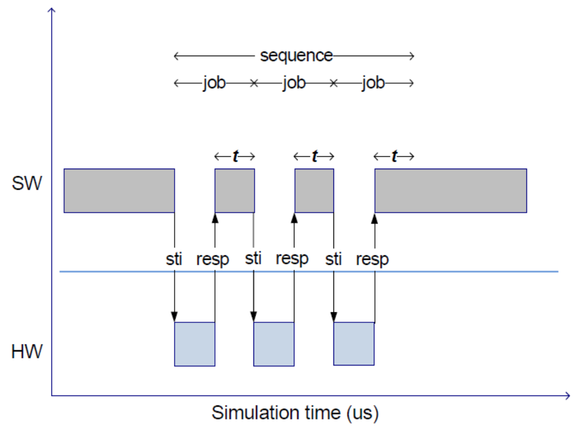


Figure 3. Synchronization between software and hardware domains.

With reference to Figure 3, synchronization is achieved by freezing the software simulation clock on each *SPI sequence* transmission request. For each *SPI job* transmission, the response data and the transmission time ***t*** are evaluated from the HW and sent back to the SW. Then the software clock advances for ***t*** time units. This is repeated for every job in an SPI sequence. When the sequence is over, the software clock is unfrozen and the software runs freely until the next transmission. In Figure 3 the data exchange between the two domains is represented by arrows across the y-axis. Gray and blue blocks represent the time when the simulation time is activated for SW and HW domains respectively.

III. Introduction of the Safety concept in the simulation

The ISO 26262 is a crucial part of the automotive development process. To comply with the process, use cases of our co-simulation system apply to different levels of the ISO 26262 development phases (Figure 4). Starting from product development at system level, where the concept can be used for design and validation of HSI specification (e.g. SPI), the validation and verification of safety goals for both, the hardware and the software level of product development.

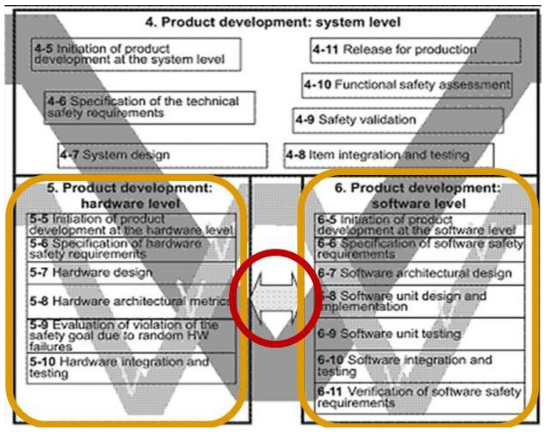

4. Product development: system level
4-5 Initiation of product development at the system level
4-11 Release for production
4-10 Functional safety assessment
4-6 Specification of the technical safety requirements
4-9 Safety validation
4-7 System design
4-8 Item integration and testing
5. Product development: hardware level
5-5 Initiation of product development at the hardware level
5-6 Specification of hardware safety requirements
5-7 Hardware design
5-8 Hardware architectural metrics
5-9 Evaluation of violation of the safety goal due to random HW failures
5-10 Hardware integration and testing
6. Product development: software level
6-5 Initiation of product development at the software level
6-6 Specification of software safety requirements
6-7 Software architectural design
6-8 Software unit design and implementation
6-9 Software unit testing
6-10 Software integration and testing
6-11 Verification of software safety requirements

Figure 4. V-Model and HW/SW interaction acc. ISO 26262.

The standard recommends fault injection on models of system level and hardware level. The goals are to check that specifications related to the behavior in the presence of faults do not contain any omissions or errors, and to ensure that the system implements appropriate mechanisms to prevent the violation of safety properties. These are effective targets linked to the verification of a system implementation, from the unit tests, through the integration phase, to the verification and validation of the complete system. At this stage, we seek to characterize the effectiveness of fault tolerance mechanisms (detection and recovery of errors) that have been implemented to increase safety as well as reliability and availability.

Fault models are inserted at defined nodes selected by Failure Mode and Effects Analysis (FMEA). The fault models are implemented in VHDL-AMS and cover analog and digital pin faults. They are identified and activated by the use of a unique fault address. Each fault model instance offers the opportunity to simulate connections to any voltage level (VDD, GND, etc.) or logic level ('1', '0', 'U', '-') in case of digital faults. For analog faults the resistance values of the connections are initially configurable via generic parameters and can be modified with a mutation factor (MF) during run time. A switch level representation is shown in Figure 5.

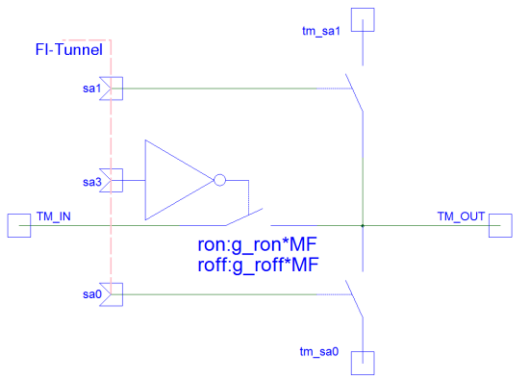


Figure 5. Switch level representation of analog fault model.

```
Type;  tx_frame;       rx_TB; FI/ADR;FI/type;wait/s; // Comment
...
SPI; 0x38000007; 0x680042E6; ;      ;       ;       // wait & ID;
SPI; 0x38000007; 0xB83A42E0; ;      ;       0;      // ID
SPI; 0x28000006;        0x0; ;      ;       0;      // read err-flags
SPI; 0x28000006;        0x0; ;      ;       0;      // read err-flags
SPI; 0x33FFFFF6;        0x0; ;      ;       0;      // clear ERR
FI;            ;           ; 0x1;   SA1;    1.0e-6; // fi pulse
FI;            ;           ; 0x5;   SA0;    0;      // fi enable
FI;            ;           ; 0x1;   SA1;    1.0e-6; // fi pulse
SPI; 0x21030401;        0x0; ;      ;       0;      // COM HS, PWM HS
SPI; 0x38004202; 0xE1030401; ;      ;       0;      // SPI/PWM1
SPI; 0x08004005; 0x38004202; ;      ;       0;      // SPI/PWM4
SPI; 0x20000170; 0x48004005; ;      ;       0;      // COM/CHA124 set
SPI; 0x33FFFFF6; 0xA0000170; ;      ;       0;      // clear SR
SPI; 0x28000006;        0x0; ;      ;       2.5e-3; // read err-flags
...
```

Figure 6. Online protocol with SPI communication and fault insertion commands (FI).

The instantiated models support various modes for activation of the faults. Reading constants defined in VHDL packages (PKG) or from a file (FileIO) is called offline mode. The online mode is used to set the models via global signals, which specify fault addresses and fault types. It is controlled by the SW driver environment, using a TRX-communication channel. An example protocol is shown in Figure 6.

IV. Case Study & Results

A. *Virutal HW environment including fault insertion*

The basic architecture of the virtual HW environment of a prototype project is shown in Figure 7. By means of a communication module (TRX) the SW developer (basic SW and driver development) has access to a virtual ASIC and module environment consisting of analog-mixed-signal components (e.g. charge pump, current and voltage monitors, SPI-interface, FETs, motor and load).

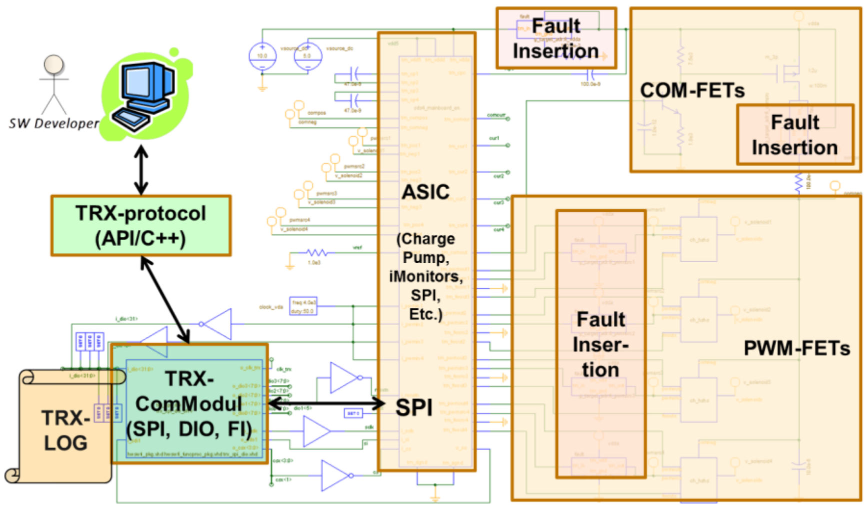


Figure 7. Architecture of the virtual HW environment.

An example of the offline fault insertion is shown in Figure 8. In offline mode the fault table is defined in a VHDL package (array of records) or a text file via FileIO (integer/real values). In this case several faults can be inserted in parallel for different signals and terminals. They can also be manipulated in a sequential, timing dependent behavior for the fault activation time and fault duration.

```
constant c_FI_ARRAY : t_fi_array := (
--ADR          type     MF/R     @time    delta
(fi_pwmsrc1,SA1,    1.0e-3, 0.5 ms, 0.5 ms   ),
(2,            NONE,    0.5,     1 ms,    2 ms     ),
(3,            SA1,     1.0e-3, 1 ms,    0 ms     ),
(10,           SA0,     1.0e-3, 4.8 ms, 0.1 ms   ),
(5,            SA0,     1.0,     2 ms,    0.5 ms   ),
(1,            SA0,     1.0e-3, 1 ms,    0.5 ms   ),
(1,            SA1,     1.0e-3, 1 ms,    0.5 ms   ),
(1,            SA3,     1.0e6,  1 ms,    0 ms     ),
(31,           SA0,     1.0,     1 ms,    0 ms     ),
(31,           SA1,     1.0,     1 ms,    1.0 ms   )
);
```

Figure 8. Fault table (array of records).

In Figure 9 the SPI communication, which configures the system and the resulting fault behavior is shown for related signals. Multiple and sequential faults in parallel, according to the fault table are active. Resulting error flags of the related status registers can be checked by SW/SPI access to analyze the error flags.

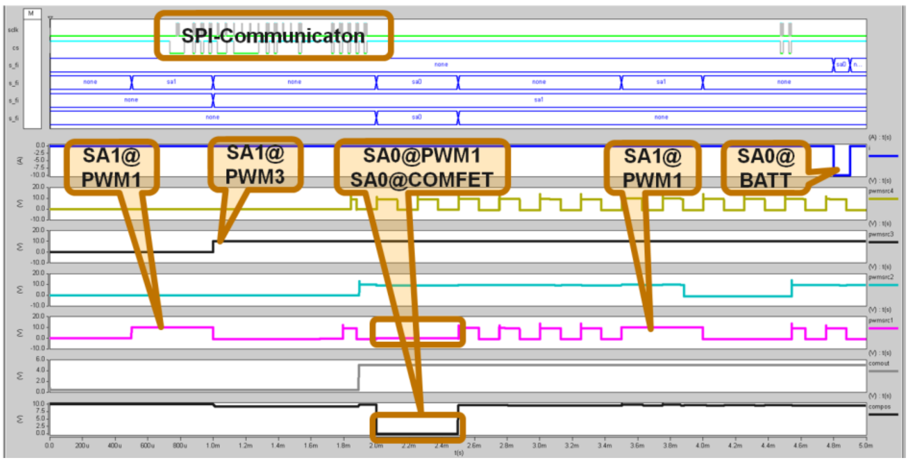


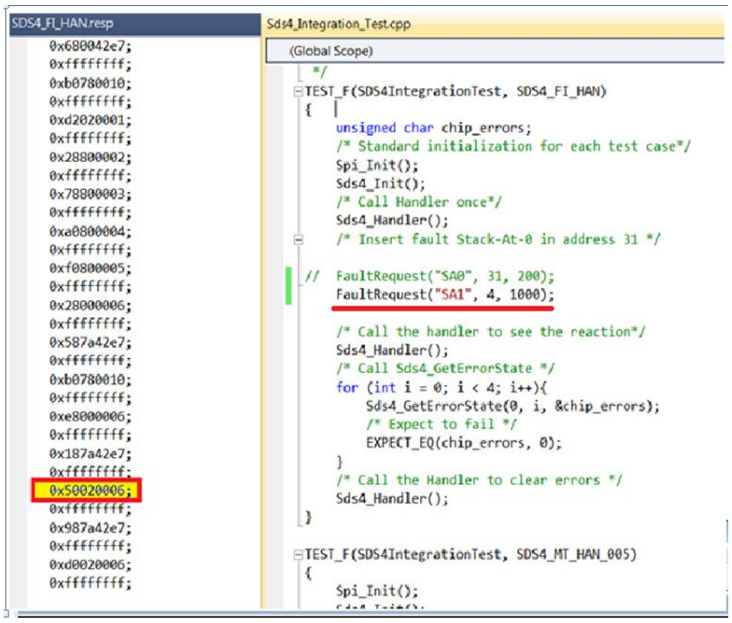
Figure 9. SPI communication and faulty analog nodes.

B. *SW Testcases*

Online fault injection as described above is practically possible from the software side. Request for fault insertion happens during test execution. The goal of such test cases would be to satisfy the safety goals, which practically mean arrival at the safe state of system operation.

The desired operation after injecting the fault can be tested, in terms of correct error reporting as well as correct reacting by the system to this error. This scheme enables the developers to validate the selection of the safety mechanism as well as their implementation of the software.

As an example, the test case in Figure 10 demonstrates the usage of dynamic (i.e. online) fault activation. In this case a SA1 (Stack-At-1) fault is requested. The function *FaultRequest* accepts three arguments: type, address and duration (ms) of fault. The request can be called from test case code. On the left side of Figure 10, the SPI response is illustrated. The SPI frame responsible for reporting the error is highlighted in yellow. The expected error bit is raised. Software can question the response for errors (Sds4_GetErrorState) and take the required actions (in the Sds4_Handler invocation), to arrive at the safe state.



```
0x680042e7;
0xffffffff;
0xb0780010;
0xffffffff;
0xd2020001;
0xffffffff;
0x28800002;
0xffffffff;
0x78800003;
0xffffffff;
0xa0800004;
0xffffffff;
0xf0800005;
0xffffffff;
0x28000006;
0xffffffff;
0x587a42e7;
0xffffffff;
0xb0780010;
0xffffffff;
0xe8000006;
0xffffffff;
0x187a42e7;
0xffffffff;
0x50020006;
0xffffffff;
0x987a42e7;
0xffffffff;
0xd0020006;
0xffffffff;
```

```
 */
TEST_F(SDS4IntegrationTest, SDS4_FI_HAN)
{
    unsigned char chip_errors;
    /* Standard initialization for each test case*/
    Spi_Init();
    Sds4_Init();
    /* Call Handler once*/
    Sds4_Handler();
    /* Insert fault Stack-At-0 in address 31 */

//  FaultRequest("SA0", 31, 200);
    FaultRequest("SA1", 4, 1000);

    /* Call the handler to see the reaction*/
    Sds4_Handler();
    /* Call Sds4_GetErrorState */
    for (int i = 0; i < 4; i++){
        Sds4_GetErrorState(0, i, &chip_errors);
        /* Expect to fail */
        EXPECT_EQ(chip_errors, 0);
    }
    /* Call the Handler to clear errors */
    Sds4_Handler();
}

TEST_F(SDS4IntegrationTest, SDS4_MT_HAN_005)
{
    Spi_Init();
```

Figure 10. Test case for validating the safety mechanisms and their implementation from software side.

V. SUMMARY AND FUTURE WORK

We presented a new hardware/software co-simulation environment for SPI enabled ASICs and software drivers. Based on VHDL-AMS models and tools early functional verification is possible. The ISO 26262 topics HW/SW interface, fault injection and regression tests are covered. Furthermore BSW driver development can be started with the virtual environment long before real prototypes are available.

A pilot project with a new BLDC motor driver ASIC was started where the HW/SW system- and co-simulation features and the presented fault analysis topics are applied in a real production environment to support the FSM requirements according to the ISO 26262 standard and the Design for Six Sigma (DFSS) activities.

ACKNOWLEDGMENT

The work was done within a internal innovation project of Continental AG and supported by José A. Avila, Executive Board Member, and responsible for the Automotive Division Powertrain. Colleagues of the Competence Centers Electronics and Software of the Business Unit Transmission were involved. Special thanks for discussions, contribution, comments and review to Michal Pluska, Marta Giribet, Sida Xu, Michael Eckl, Eric Taistra, and Marc Arabackyj.

REFERENCES

[1] A. Plange, E. Kyrlies-Chrysoulidis, J. Schmid, T. Gürtler, HW/SW-Cosimulation of ASICs and SW Drivers for Fault Analysis and Regression Tests, CSC-2013 (Continental SW Conference), Timisoara, Romania

[2] On/Offline Fault Insertion & Analysis in a Virtual ASIC System Simulation Environment; J. Schmid, M. Eckl, iSyst GmbH; M. Arabackyj, A. Plange, Continental AG (Division Powertrain); 26. ITG/GI/GMM-Workshop TuZ-2014, Kloster Banz, Germany

[3] J. Schmid, B. von Edlinger, A. Plange, F. Lehmann, Automotive System Verification using Saber/Modelsim Cosimulation in Conjunction with ISO 26262, SNUG-2012, Munich

[4] ISO 262672, Road vehicles - Functional safety, http://www.iso.org/iso/search.htm?qt=26262&published=on

[5] VHDL-AMS, analog-mixed-signal extention of VHDL, http://en.wikipedia.org/wiki/VHDL-AMS

[6] Saber/SaberRD, Synopsys, www.synopsys.com/Systems/Saber

[7] ModelSim, Mentor Graphics, http://model.com/