6

From Control Design to FPGA Implementation

Marcus Müller, Hans-Christian Schwannecke and Wolfgang Fengler
Ilmenau University of Technology
Germany

1. Introduction

As a modelling and simulation tool, MATLAB/Simulink plays a significant role in industrial control design. To allow the deployment of the designed control solutions a number of code generation facilities have been developed, that aim on implementing PC-based control units or embedded controllers based on micro-controllers or digital signal processors (DSP). This has enabled MATLAB/Simulink to cover the model-based development process from conceptual design over simulative validation to rapid-prototyping (Krasner (2004)). Although not a new technology per se, since the first commercial field-programmable gate array (FPGA) has been released by Xilinx in 1985, recent development in the field of reconfigurable hardware has created FPGAs, the capabilities of which apply to the requirements of industrial control in a growing number of application domains (Monmasson & Cirstea (2007)). Therefore, also code generation facilities targeting FPGAs have emerged, that link the modelling level to the level of hardware description languages (HDL), from which the lower-level descriptions (RTL) and chip-specific implementations are derived using hardware synthesis tool chains.

MATLAB/Simulink manages to incorporate the processes of high level control design, system modelling on various levels of abstraction and, the availability of respective tool boxes provided, the target-specific code generation. Yet, there is a semantic gap between the control system model, and the model describing an implementation of the controller. The former is a platform-independent simulation model, an executable specification for a controller design, that fulfills the control task within the given simulation scenario. From such a model various implementations can be derived, but not without a step via a platform-specific model, that, to a certain extend, has to incorporate the characteristics of the target platform. An important issue to consider at this point is, that the modelling semantics changes from a functional view to the execution view. The question is no longer “What is being computed?”, but “How is it computed?” - a question, that has even more impact on models aimed on generating hardware designs, than on those targeting software generation.

Especially models targeting HDL code generation and hardware synthesis have to undergo an explicit change of the abstraction level, i.e. *model refinement*, towards a hardware-specific implementation model. The Simulink model has to very closely resemble the hardware structure as well as the behaviour of the data flowing through the chip.

This chapter will illustrate the characteristics of this semantic gap and demonstrate a number of techniques on how to make a consistent transition. An overview of control system modelling using Simulink as well as the HDL Coder code generation flow will be given. As

an example the model of a 3D trajectory tracking controller for deployment in high-precision positioning and measuring stages will be introduced. The main parts of this chapter regard the structural and behavioural changes, that create a HDL-compatible implementation model from a given simulation model in a top-down manner.

In conclusion a modelling flow is proposed, that - in the course of a model-based top-down development process - precedes the utilization of MATLAB-associated HDL code generation facilities and describes the transition from a platform-independent control system model to a hardware-specific implementation model of the controller. This flow incorporates all discussed modelling activities in a consistent manner, aims on minimizing redesign iterations and allows a simulative validation during all design steps.

2. Control design using Simulink

The extent of MATLAB/Simulink allows the design of control systems on various levels of abstraction. A *system* model is comprised of:

- a model of the controlled process/plant,
- environment and instrumentation, and
- the controller, the *device under development*.

While the plant model can be of a very abstract form, only describing the transfer function or functional behaviour of the controlled process, the controller model is the part, that undergoes the model-based development from control design over refinement to implementation.

The *control design* incorporates the functional design of the controller with regard to the requirements of the plant. As a result, a controller structure and parametrization is achieved, that fulfills the required control task and quality, which can be validated by simulation. This model is called a *platform-independent model* (PIM) of the controller.

When aiming on deploying this solution to the real world, the subsequent step would be the generation of an executable implementation for a processing platform. MATLAB/Simulink provides a variety of possibilities to generate software for PC-based or embedded targets using the MATLAB tool boxes Real-Time Workshop and Embedded Coder plus several vendor-specific extensions, that require no or little additional modelling effort to build a compatible model - a *platform-specific model* (PSM). All code generation facilities provide an abstraction of the targeted processing platform in form of a platform-specific block library and the according tools to transform the blocks in adequate code fragments. For single-processor targets this is quite straight-forward. More difficult is the targeting of multi-processor hardware, since the inter-processor communication has to be incorporated on modelling level (Müller et al. (2009)), and embedded hardware, that often includes the addressing of peripherals.

Figure 1 depicts a Simulink-based development flow tailored for an FPGA implementation of the controller. Regarding the *Target platform abstraction libraries and tools* for HDL, two methods of tool integration can be distinguished, that greatly ease the transition from model to HDL code and the implementation on hardware. First, there are the tool boxes provided by hardware vendors, like the Altera Corporation (2011) DSP Builder and the Xilinx, Inc. (2011) System Generator for DSP, that provide proprietary block sets as additional Simulink libraries and access the respective synthesis tools. Second, there is the HDL generation process via The

Mathworks (2011b) Simulink HDL Coder, which has undergone an extensive development during recent MATLAB revisions, and conducts the HDL code generation from a growing subset of Simulink library blocks (Auger (2008)), allowing a more hardware-independent design approach. Similar within all approaches is the final utilization of the vendor-specific implementation tool chains, that conduct the synthesis and FPGA implementation of the generated HDL code and provide the option to conduct a further validation step on code level via HDL simulation.

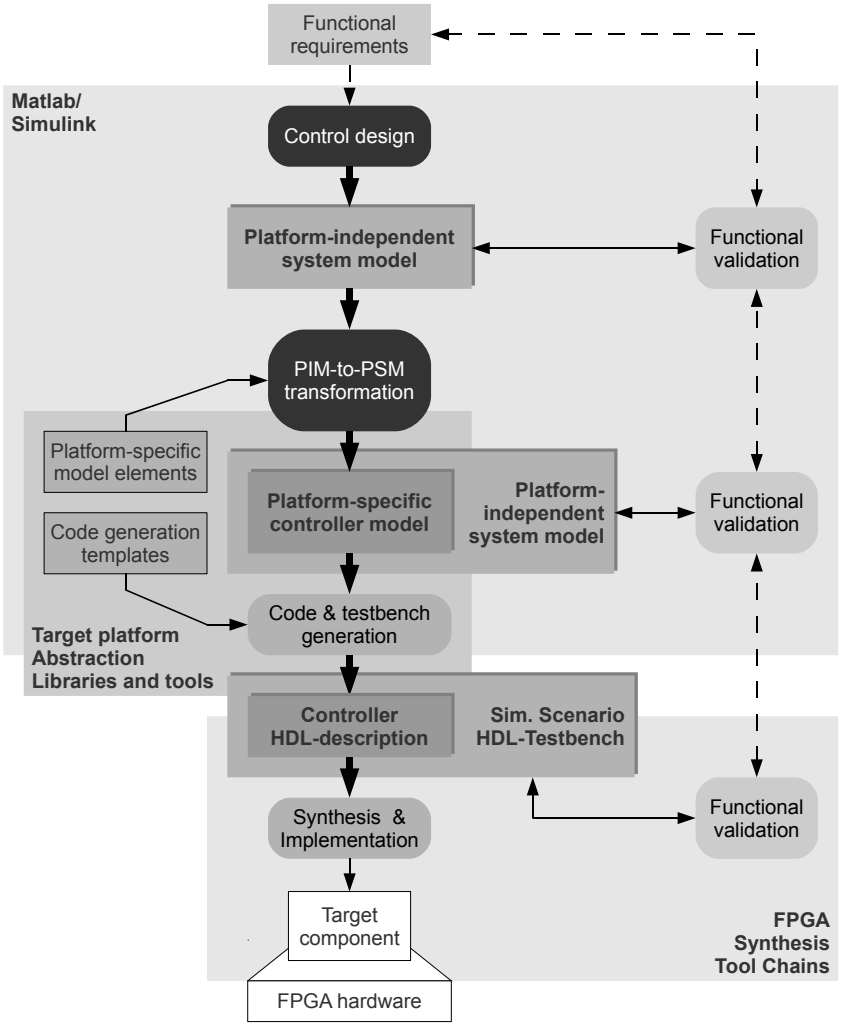


Fig. 1. Overview of the Simulink-based controller development flow targeting FPGA hardware.

Result of Simulink control system modelling is a controller design that fulfills the specified control task and quality, which has been validated by simulation within the bounds of the

given scenario. The critical activity is the *transformation of the PIM into the PSM* - the creation of a model, that conforms to the characteristics of hardware execution and fulfills all the requirements to seamlessly run the respective HDL code generators. Therefore, the given controller design and behaviour serves as specification, against which the derived and refined HDL-compatible implementation model has to be validated.

Before the Simulink model characteristics, that may not conform with HDL code generation and/or hardware implementation, are discussed, the example system will be introduced, which will provide the *device under development* for this contribution.

2.1 Example control system

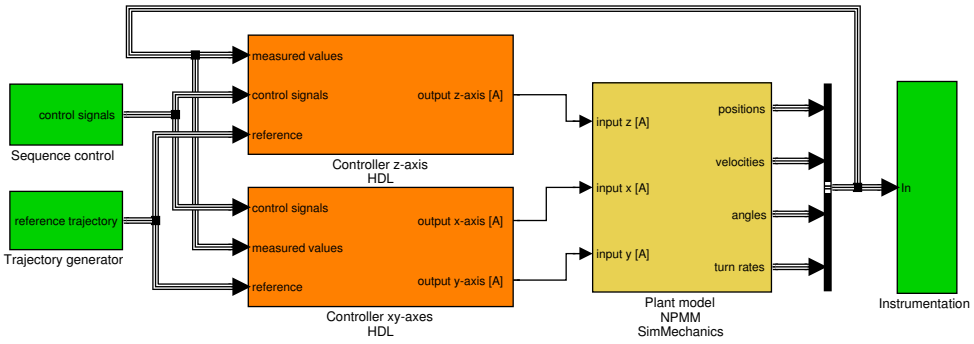


Fig. 2. Top level view of multi-axis trajectory control system model.

As an example the model of a 3D trajectory tracking controller for deployment in high-precision positioning and measuring stages, as presented by Zschäck et al. (2010), will be used. As depicted in Figure 2 the system model is comprised of three partitions. The yellow part contains the *Plant Model*, a SimMechanics model of the positioning stage of a nano-metre positioning and measuring machine, short NPMM. The stage receives input currents to power the electro-magnetic actuators for all three spacial axes, while the lateral positions and velocities as well as the angles and turn velocities of the stage leave the plant model as measured values. The green partition, the *Environment and Instrumentation* partition, consists of a sequence control to manage the system's general modes of operation and a trajectory generator to provide the set points for the controller partition. The control sample rate of the process is 10kHz, so the model is simulated with a base sample time of $100\mu s$.

The *Controller* partition itself, coloured in orange, consists of PID controllers coupled with Kalman filter disturbance observers for each axis, supplemented by additional controllers for angle correction between the axes. The controller is divided into separate modules for z-axis and x-y-plane control. In the course of this contribution a closer look will be taken on the x-axis controller, that is depicted in Figure 3.

In the x-axis controller module first the tracking errors of both position and velocity are determined, before both are subjected to the P and D gains, respectively. From the position error a nonlinear I gain is obtained from a look-up table, after which the values are submitted to the actual discrete integrator. A limiting block after the summation of the P, I and D components and an integrator anti-windup function complete the PID controller. A *Kalman filter* block serves as a disturbance observer for the non-measurable effects - in this application

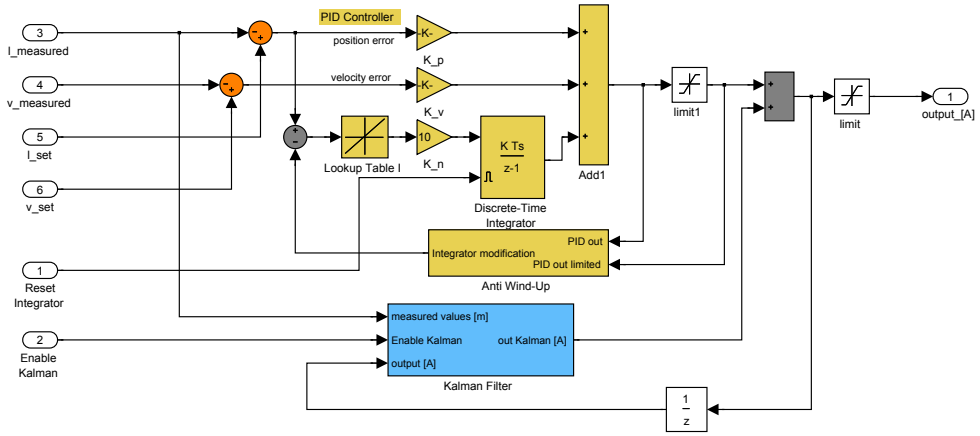


Fig. 3. Internal view of a single axis controller module.

domain, these are mainly friction forces, as described by Amthor et al. (2008). The filter works on the current measured position and velocity values, as well as on the last control step's accumulated output value, to correct its internal state and predict the current step's friction compensation value. The x-axis control output value is accumulated from the PID controller's and Kalman filter's outputs and represents a current to drive a linear electromagnetic actuator.

2.2 Controller partition and the embedded hardware realization

Since the purpose of this model is to design a controller for the specified plant and control task, the properties of the hardware interface and the future implementation hardware are omitted. In this model, digital measurement and set point values are submitted to the controller at discrete points in time without any delay - a behaviour, that in a physically realized control unit requires the passing of peripheral hardware, including A/D and D/A converters, before the actual controller function can be processed.

The developer targeting an FPGA as processing resource for the controller has to be aware of the questions, how far towards the platform-specific properties the controller implementation model can be refined, and how the resulting generated component has to be integrated into the embedded platform.

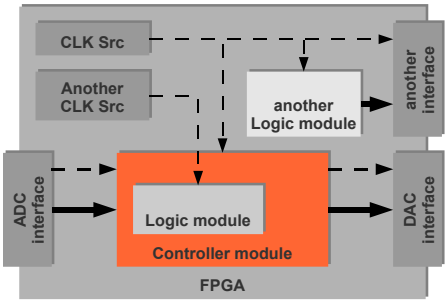


Fig. 4. A generated controller component embedded in a possible FPGA design.

As stated e.g. in Xilinx, Inc. (2011), some parts of a chip design require a level of design control, that cannot easily abstracted from and therefore cannot be adequately dealt with on modelling level. This includes the connection to restricted on-chip resources like peripheral and memory interfaces and the clock management. Figure 4 illustrates the generic structure of an FPGA design, which includes a component generated from a model. This component is likely to reside on the chip with other logic modules, side by side or in hierarchical relations - each will have to be connected to a clock source, but there might exist different clocks driving different components. The modules may also be connected to predefined interfaces to off-chip resources, with which they exchange data and synchronisation signals like triggers.

In some cases it might be possible to manage the complete chip design from the modelling level, but the general case will be to generate a component to be integrated in a chip design. This integration, however, will be the more seamless, the more the model of the controller component is refined to correspond to the structural and behavioural requirements of the embedding design.

2.3 Simulink model properties and HDL code generation

At this point a closer look will be taken at a couple of properties of a Simulink model, that need special consideration when aiming on generating HDL code from this model and implementing this HDL description on a chip.

A Simulink “time-based block diagram" model resembles a synchronous data flow graph, the signal outputs and states of which are computed every simulation time step. One simulation time step defines the time interval in which a significant change of values is to be recorded. In principle, simulation modes with variable time steps can be used in order to decrease the time intervals in cases of rapidly changing values, so that the discretization error can be minimized. For details on the block diagram and time simulation semantics the reader is referred to The Mathworks (2011a) Simulink User Guide. For this control design, the simulation time resolution is determined by the fixed sample period of the time-discrete controller, which here will be referred to as `T_Control` with $100\mu s$. For a hardware-oriented model, the simulation time interval resembles the fixed period of the clock signal (`T_Clk`) driving the generated logic. Due to this fact, the notion of time resolution has to be changed - from the sample time of the controller, the interval, in which new input values are acquired, to the execution cycle period, the time interval a logic stage computes one cycle.

MATLAB and Simulink allow heterogeneous modelling and the integration of models with different levels of functional abstraction. This is majorly supported by the MATLAB functions and Simulink blocks being polymorph. That means, they work on scalar, vector and matrix data alike, as well handle different data types transparently to the engineer. The library functions/blocks are designed to easily create complex models from reusable, parametrizable blocks, hiding the actual implementation details. But these details are significant when considering a hardware design, because some function, as e.g. division operators, cannot be trivially mapped to binary logic. As a consequence, only a subset of the Simulink libraries feature corresponding synthesizable HDL constructs. For any non-synthesizable blocks adequate substitute constructs have to be modelled or the block has to be treated as an abstract black-box, that is to be replaced by a pre-implemented external HDL-component during code generation.

By default, MATLAB uses the data type `DOUBLE` for all its algorithm descriptions, be it in MATLAB script language or Simulink block models. `DOUBLE` represents a 64 bit wide floating-point number, which covers a range of up to $\pm 2^{1024} \approx 2 \times 10^{308}$ and a precision of up to $2^{-52} \approx 2 \times 10^{-16}$, which is supposed to cover the needs of any modelling and simulation task. Any code generation without interference would map the model signals to variables of a supported floating-point type. Difficulties could arise when trying to deploy this code to a processing hardware - floating-point units are quite complex and comparatively slow in general-purpose processors, some support only single-precision (32 bit), some micro-controllers and DSPs do not even feature any, and in custom hardware designs they can only be used in very limited numbers. So double-precision floating-point arithmetic has to be time-intensively emulated, limited hardware units have to be reused in a time-multiplexed fashion, requiring a complex execution flow, or the arithmetic functionality cannot be realized at all. The hardware description language VHDL supports the double-precision floating-point data type `REAL`, which allows direct code generation from Simulink. The VHDL simulation of floating-point arithmetic is possible, but the `REAL` data type is not synthesizable, as stated e.g. by Rushton (2011). Therefore, any platform-specific Simulink model, that is intended to resemble a hardware description, has to feature integer or fixed-point arithmetic.

The following sections will exemplify several activities to create a HDL compliant and hardware implementable model from the controller design introduced above. Since the 2010 version of MATLAB introduced the HDL Workflow Advisor for the Simulink HDL Coder, a tool is featured that guides the code generation, and via utilization of vendor-specific synthesis tools, also allows the rapid-prototyping synthesis and FPGA implementation of Simulink models. The model transformations will be described with regard to the Simulink HDL Coder, since no change in the basic block set will be required. First, structural changes to the data path of the model will be discussed, followed by behavioural modifications, that transform the model behaviour to the hardware execution behaviour.

3. Data path design

The structural changes to a model in order to facilitate both successful HDL code generation and successful hardware implementation, that are considered in this section, include

- the insertion of HDL-compatible substitutes,
- the utilization of black-box blocks to incorporate predefined HDL components, and
- the fixed-point data type conversions.

3.1 HDL-Coder supported substitute constructs

The substitution of blocks is necessary, since only a subset of the modelling blocks and block configurations available in Simulink supports the generation of HDL code. The blocks that do are compiled into the so called "Supported Blocks Library" by the `hdllib` command - refer to The Mathworks (2011b) HDL Coder user guide for details. Still some blocks can feature modes or configurations, that are not synthesizable. Common examples are the arithmetic function blocks that polymorphically work on scalars, as well as on vectors and matrices. Since matrix operations are not supported these blocks have to be substituted by explicit constructs.

Regarding the controller model displayed in Figure 3, the first block to consider is the *Discrete Integrator* block. This block can be configured to a mode of computation, e.g. Forward-Euler,

Backward-Euler, etc. In this case the integrator is required to be resettable, since the internal value has to be set to zero in case of de- and reactivation of the controller module. The *resettable* feature is documented as not being HDL compatible, so the block has to be substituted by the self-tailored subsystem, that is depicted in Figure 5. It resembles a forward-Euler integrator with the discrete sample time interval of `T_Control`, the internal state of which can be reset by an external signal.

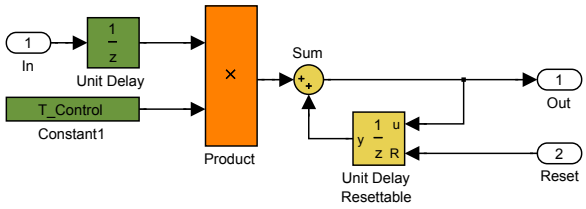


Fig. 5. Substitute for the Forward-Euler Discrete Integrator block with external reset.

A further aspect for consideration is the use of *Look-up tables* (LUT), like the one determining the variable *I*-gain in Figure 3. Simulink LUT blocks can be configured to determine an output value according to a set of sampling points and an interpolation method, like in this case linear *Interpolation/Extrapolation*.

Starting with MATLAB Release 2011a the HDL Coder supports the code generation from LUTs configured to linear interpolation, but only based on evenly distributed sampling points. For certain LUT mappings the minimal distance between sampling points would require a large number of them to cover the input data value range, resulting in a large memory requirement for the on-chip LUT implementation.

A method to substitute a Simulink look-up table with linear interpolation/extrapolation between sparsely and arbitrarily distributed sampling points is presented in Figure 6. Here, the input value range is subdivided into intervals according to the sampling points. The input value x is compared to the interval boundaries, the comparison results are accumulated to an index value. A switch assigns a respective set of coefficients A, B to the actual linear interpolation $y = Ax + B$.

3.2 Fixed-point conversion

To effectively implement an algorithm into hardware, it is advised to utilize an integer or fixed-point representation of both data and algorithm. Since the initially designed executable specification model uses double-precision floating-point data representation, a conversion to fixed-point data types has to be performed.

The key steps are to determine the maximum/minimum values to be covered, and the minimal discretization step width. From these information, the fixed-point format consisting of *sign bit*, *word length* and *position of radix point* can be derived, which in MATLAB is denoted as e.g. `fixdt(1,16,4)` [1].

MATLAB features the Fixed-Point Tool and the Fixed-Point Advisor process, documented by The Mathworks (2010), to aid the determination of the respective fixed-point formats. The

[1] for a 16 bit wide signed number with the four least significant bits being fraction bits

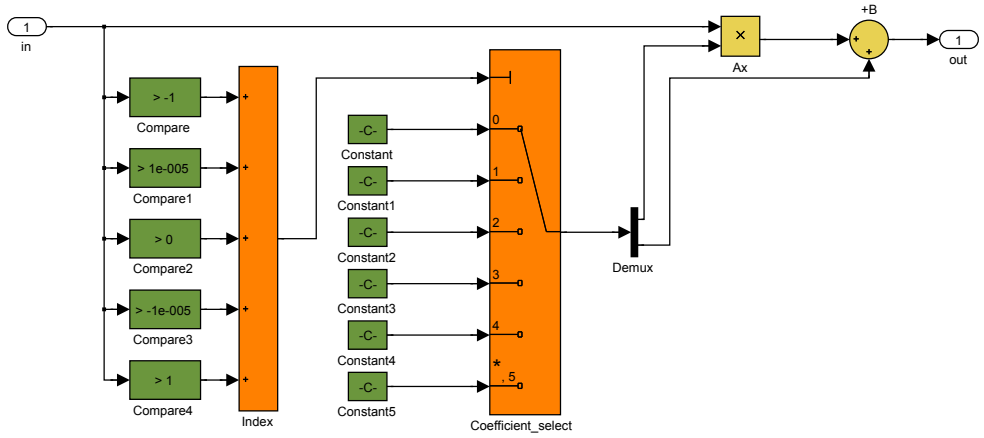


Fig. 6. Substitute for Simulink Look-up table block with linear *Interpolation/Extrapolation* between arbitrarily distributed sampling points.

automated Fixed-Point Advisor conversion process relies on the default data type definitions for *classes* of values, even if the hardware implementation option is set to `FPGA/ASIC`. So, globally fixed word lengths for all the input values, constant values and internal outputs, respectively, have to be set by the designer. Based on the globally set word length definitions the fixed-point tools move the radix point to achieve the maximum precision while covering the maximally necessary value range. Only afterwards can be determined, if the resulting precision meets the specified precision requirements, usually leading to an iterative process of re-adapting the word lengths.

For FPGA implementations there are per se no such restrictions as globally predefined operand word lengths. On the contrary, for each operation the operand and result widths can be determined with bit accuracy, therefore allowing to tailor the word lengths exactly to the algorithms' requirements, especially regarding the fact, that the specified precision can be used as a-priori information.

Regarding this, it should be noted, that a decimal fraction cannot easily be expressed using a binary fraction, since each additional fraction bit only enhances precision to the next negative power of two, resulting in a decimal precision with a maximum numerical error ϵ. Based on the approach presented by Carletta & Veillette (2005), the required positions for the most significant bit (MSB) and the least significant bit (LSB) in a binary representation of the decimal value x with a desired decimal precision ϵ can be computed by $P_{MSB}(x) = \lfloor log_2|x| \rfloor + 1$, and $P_{LSB}(x)s = \lceil log_2(\epsilon * |x|) \rceil - 1$, respectively. The required binary word length L is given by $L = P_{MSB} - P_{LSB}$ plus the sign bit, if required.

A successive fixed-point conversion is conducted based on the minimal fixed-point expression of all inputs, constants and coefficients, and on the following rules:

- for binary Addition of two operands: $L_{result} = max_i(L_{operand\_i}) + 1$,
- for binary Multiplication of two operands: $P_{MSB}(result) = \sum P_{MSB}(operand_i)$ and $P_{LSB}(result) = \sum P_{LSB}(operand_i)$

The Simulink block parametrization option `Data type inheritance via internal rule` works similarly, but can not resolve feed back loops without major restrictions. In those cases definitely manual parametrization based on good knowledge of the process parameters is expedient.

The operand bit widths can be regarded as a measure for chip area utilization, since they influence the complexity of the synthesized register and routing structures on the target chip. The use of the Fixed-Point Advisor will produce valid solutions for hardware implementation, which therefore makes it a valuable tool for rapid prototyping, but the advantage of a manual minimization of the fixed-point representation and its effect on chip resource utilisation becomes obvious.

3.3 Excluding externally provided black-box blocks

The utilization of black-box components is a means to include externally available implementations as substitutes for Simulink modelling blocks, that are not or not feasibly synthesizable. The model element regarded as black box will still be simulated, but neglected during HDL code generation, where an interface-compatible predefined HDL component will be referenced instead. For details on the declaration of a black box subsystem refer to The Mathworks (2011b) HDL Coder documentation.

For the presented controller design these circumstances will be illustrated by the example of the Kalman filter, the design of which has been presented by Amthor et al. (2008). For this Simulink module no feasible hardware implementation could be generated using a model-driven HDL generation process. This is not at least caused by the fact, that for the high-precision application domain this filter has no feasible fixed-point solution in terms of required data widths and resulting chip area requirements. So, the computation of this function on the hardware is conducted using a highly optimized VHDL-designed core, based on the floating-point processing architecture presented by Pacholik et al. (2011). According to its specification, the IP[2] component executing the Kalman filter function has the following properties, which are of significance for the model integration:

- clock port for internal clock (up to 120 MHz),
- latency: 215 clock cycles @ 120 MHz $\approx 1.8\mu s$,
- clock port for external clock (to allow synchronisation with the surrounding design),
- execution triggered on rising edge of `Trigger` signal,
- result availability signalled by `Valid` signal (pulse with width of one external clock period).

In order for the IP component to be correctly referenced during the synthesis process the Simulink black-box subsystem requires the exact replication of the IP components interface and motivates the following adaptions in the HDL-oriented Simulink model.

Figure 7 shows the configuration of the Kalman filter subsystem as a black-box component. A closer look reveals that the data ports of the Simulink subsystem block have been supplemented with a `CLK_120MHz`, a `Trigger` and `Valid` port. The dialog shows options to configure the appearance of the interface of the HDL instance - in addition to the data

[2] "Intellectual property" cores describe pre-designed logic blocks to be reused for FPGA/ASIC design

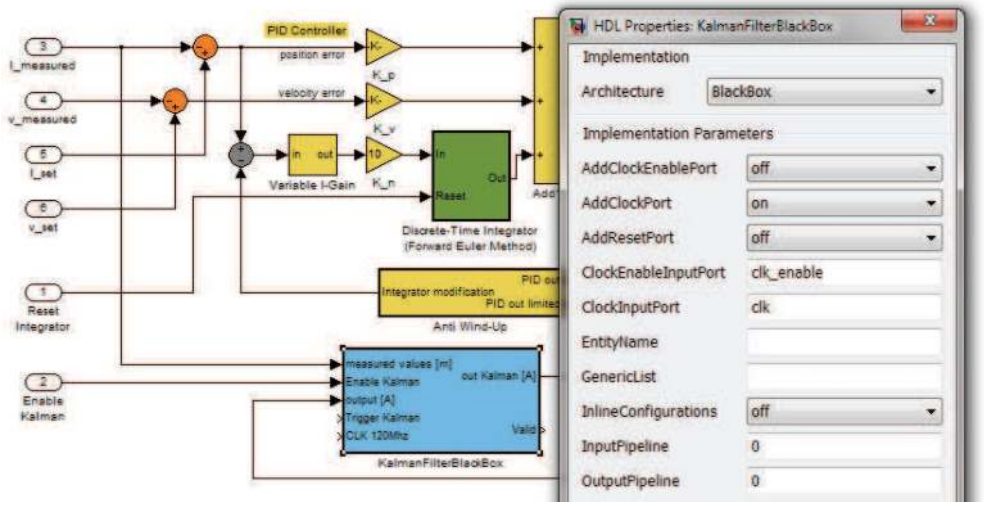


Fig. 7. Declaring a black-box component.

ports from the Simulink block - to fit the predefined component interface, as documented in The Mathworks (2011b) HDL Coder guide. The clock port `clk`, that is added, matches the hardware port connected to the clock of the surrounding design, which completes the replication of the referenced component's interface. At this point the black-box declared subsystem ensures both, the correct simulation using the platform-independent filter model, and the structurally correct instantiation and therefore seamless integration of the Kalman filter component at code level.

3.4 Resulting model

The model of the x-axis controller module after the completed data path design steps is depicted in Figure 8. It includes the substituted discrete integrator and look-up table blocks, the black-box interface of the Kalman block, as well as additional compatibility issues like the resolved bus signals, and shows the resulting fixed-point data widths for the operands. At this point the HDL Coder compatibility check should complete successfully.

4. Behavioral design

The behavioural changes the model has to undergo during its transformation into an implementation model are mainly characterized by a changed notion of time[3]. In the simulation model the base sample rate describes the interval, in which the change of relevant physical values is observed - the control system sample time `T_Control`. The computation of the outputs of the controller and the reaction of the plant is done "timelessly" at one point in simulation time. In reality, as illustrated in Figure 9, the begin of the time interval `T_Control` marks the time, when measured values are acquired (DAQ), before they are submitted to the controller module and the computation of the controller function can begin. The computation

[3] Prerequisites for behavioural compatibility to the HDL Coder code generation process are set by the `hdlsetup` command - refer to the The Mathworks (2011b) HDL Coder documentation.

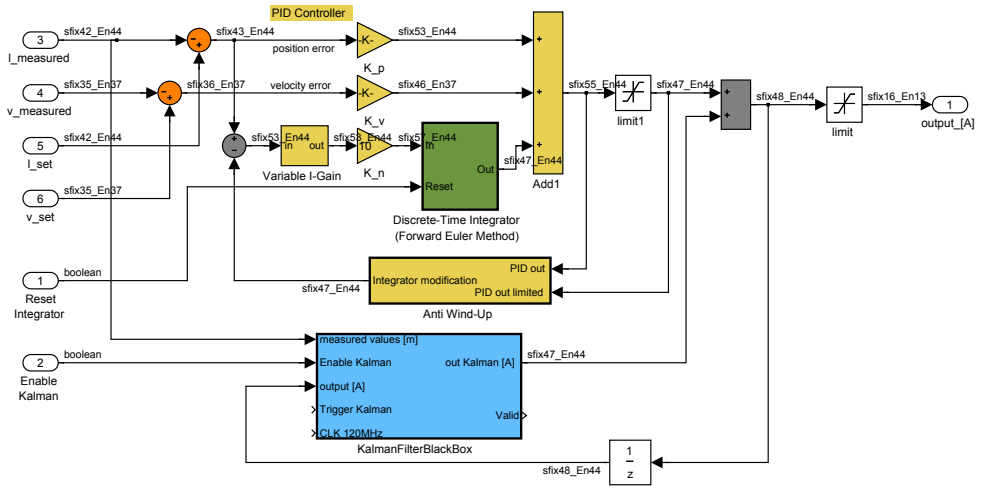


Fig. 8. Structurally HDL compatible controller model.

of the controller function and the control output have to be completed within one period of `T_Control` in order to fulfill the real-time execution restriction for the discrete-time controller design (Caspi & Maler (2005)).

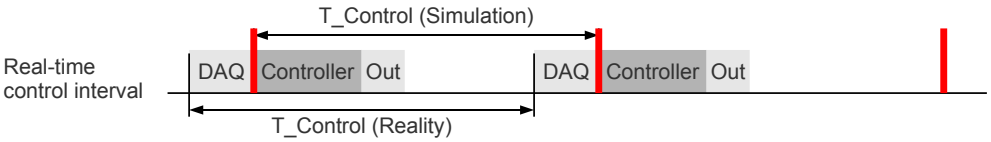


Fig. 9. Execution cycle of a real-time control system in reality and simulation.

Since the platform-independent model abstracts from the DAQ and output behaviour, from the model-based design point of view, the beginning of the `T_Control` interval has to be regarded as the point in time, when the state of the plant has been captured and the controller simulation/computation is initiated. For the platform-specific controller model the question arises, how to regard the hardware execution of the controller and its delay in relation to the real-time interval and how to model it.

In this section first a hardware implementable model is directly derived from a `T_Control`-cycle oriented model to analyze its hardware execution semantics. Afterwards a model with advanced clocking and trigger structures necessary to ensure consistent hardware execution are introduced. Along with the required modifications to the model structure and the simulation semantics the advantages and disadvantages of each modelling and implementation variant are discussed.

4.1 Control cycle oriented design

When applying the HDL Workflow Advisor directly to the model shown in Figure 8 a valid HDL entity of the controller model including the referenced HDL component for the black box will be generated. The data paths from the inputs through the internal arithmetic functions of

this entity will be transformed into *combinational logic*. The unit delays in the feedback path, see Figure 8, and in the integrator block, see Figure 5 storing the control cycle state will be realized as *registers*. The update of these registers will be triggered by a rising edge of the clock signal connected to this entity.

At this level of abstraction, although the Kalman filter subsystem is still simulated as an abstract black box, the integration of the underlying HDL component in the logic path has to be prepared. This includes the connection to a 120 MHz clock source, which is best designed at chip level. Therefore, the according clock port `CLK_120MHz` is linked to the controller component's interface to the top level. Furthermore, the Kalman filter component has to be triggered at a point in time defined by outside prerequisites, which requires the connection of the `Trigger` port to an outside source. Both modifications are shown in Figure 10.

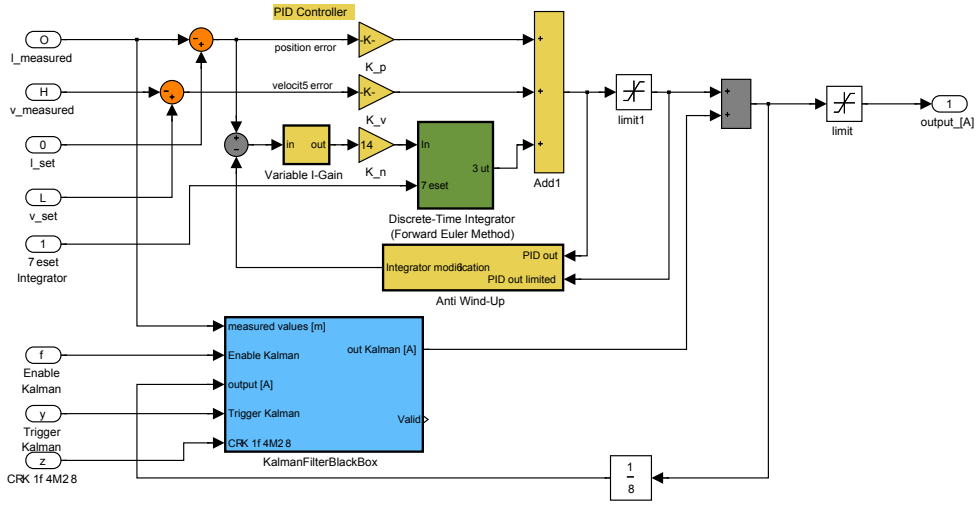


Fig. 10. Model of the controller with an externally clocked Kalman filter component.

Prototyping the implementation with the HDL Coder Workflow Advisor and using the back-annotation feature allows to examine the computational delays of the combinational path of the resulting hardware implementation of the controller. For the example design this value is $45ns$. Regarding the $1.8\mu s$ execution delay of the Kalman filter, which on the FPGA is computed in parallel, it is certain, that the computation is completed well within the `T_Control` $= 100\mu s$ real-time interval.

During code generation the HDL Coder will generate an entity, that, aside from the ports defined in the Simulink model, declares the port `clk` to connect to an on-chip clock source. As the simulation model computed one step per sample interval `T_Control`, the controller implementation will conduct one computation step, if the `clk` port is driven by a clock source with the period of the simulation base sample time, i.e. clock frequency $f = 1/T\_Control$.

But it has to be noted, that the only safe assertion, that can be made for hardware design derived from this model, is that at each rising clock edge the output and integrator values are stored in the respective registers. Due to the combinational logic, the design is sensitive to the *input/output behaviour* of the surrounding hardware design. All value changes of the signals

will be directly propagated through the circuit, only subject to gate delay - an effect that cannot be expressed in the Simulink model with a time resolution of `T_Control`. This makes it very difficult to determine, e.g. when all current input data will be available and stable in relation to the clock edge, when to trigger the Kalman filter component on valid input data and when to submit the result values to the surrounding circuitry.

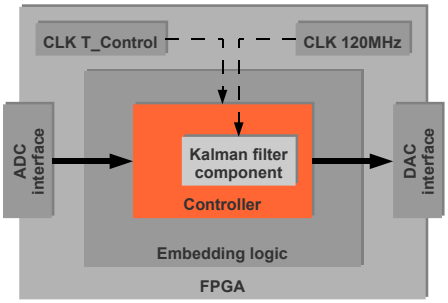


Fig. 11. Integration of the generated controller and Kalman filter components in an FPGA design.

If the logic embedding the controller component, as depicted in Figure 11, is known to provide the required clock and trigger signals, to guarantee for the stability of the input values for the computation time and to correctly time the take-over of the result values, the utilization of a combinationally integrated component should not pose a problem. Nevertheless, the correct design of the embedding logic and the behavioural validation of both, the component and the overall implementation are efforts, that cannot be undertaken on modelling level. To provide a more robust component implementation and to ease the integration of the component in a chip design a more advanced modelling approach should be considered.

4.2 Execution cycle oriented clocked design

Guidelines for good design practice regarding the creation of reusable IP components have been formulated by Keating & Bricaud (2002). A key feature is the independence from the influence of embedding logic regarding the data propagation timings, especially when the timing behaviour is unknown or cannot be guaranteed at component design time.

The method to ensure the simultaneous submission of input values to the component design and the stability of those inputs for a defined time period is to provide input registers. These registers are triggered to store the input values at defined points in time, when the validity of the values can be assumed. This decouples the timing of the component from any surrounding issues, so the internal behaviour of the component can be designed based on this trigger event.

On modelling level the Simulink controller design has to be equipped with input registers, that are triggered to store the input values at the sample points defined by `T_Control`. Additionally, the output has to be provided with a register to submit a stable and valid value to the embedding logic after the computation has been completed. These port registers are modelled by additional *Enabled Unit Delays*, as depicted in Figure 12.

Now the data propagation along the forward path of the data flow has been divided in multiple logic stages and is therefore delayed by multiple simulation cycles. Still, the simulation of the controller has to be accomplished within the period `T_Control`, i.e. within

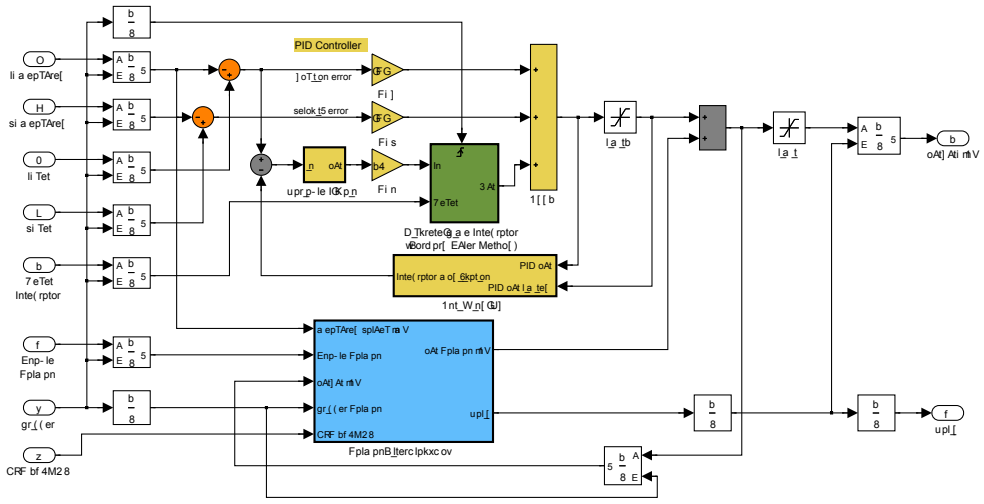


Fig. 12. Model of a controller design with registered I/O ports.

one simulation interval of the plant and environment model. This requires the introduction of a *new simulation base sample time* with a resolution high enough to fit the respective controller simulation cycles. In the following, this simulation sample period will be referred to as `T_Clk`.

The controller design shown in Figure 12 is now simulated with `T_Clk` - each unit delay only adds one `T_Clk` latency. Still, there are values, that have to be updated only once every `T_Control`. Since this can no longer implicitly accomplished by using unit delays based on `T_Control`, it has to be expressed by unit delays enabled for one `T_Clk` every `T_Control` interval by an explicitly modelled trigger signal. The relation of the control interval `T_Control`, the execution cycle interval `T_Clk`[4] and the trigger signal, representing the beginning of the control cycle in the execution time domain, is displayed in Figure 13.

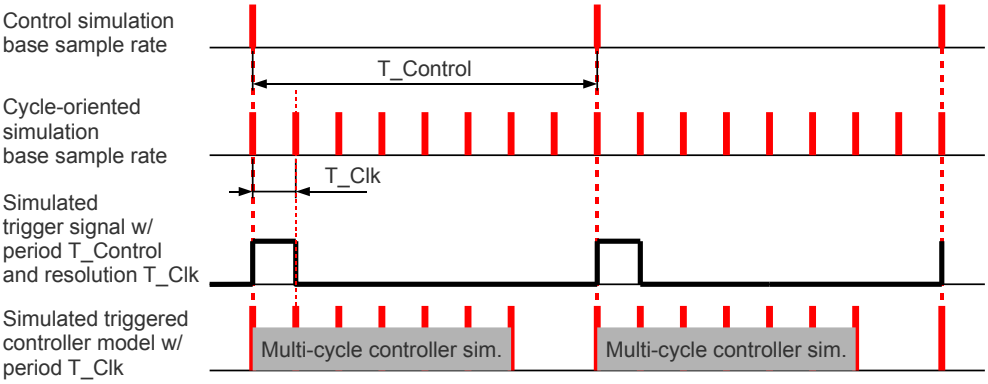


Fig. 13. Simulation rate multiplication for computation cycle oriented simulation.

[4] Simulation solver restrictions still require `T_Control` being an integer multiple of the base sample time `T_Clk`

This trigger signal is used to enable the following elements for one cycle of `T_Clk` in order to store a value at the beginning of each `T_Control` interval. First, the controller's input unit delays take over the current input values. Second, the unit delay storing the feedback output value is enabled. Third, the Discrete Integrator block has to store the last accumulated value. For this purpose, the integrator block from Figure 5 has been turned into a triggered subsystem. Until the next enable phase these values will be kept stable by the registers. Within one `T_Clk` the results are propagated through the design and on a further trigger submitted to the output register. Since the result depends on the output of the Kalman filter component, the trigger for the output register is derived from the `Valid` signal of this component.

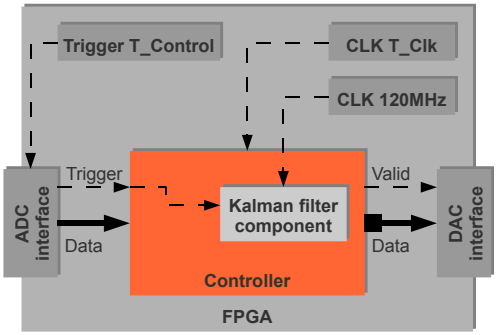


Fig. 14. Integration of the generated synchronized controller into an FPGA design.

Figure 14 depicts the integration of the I/O-synchronized controller into an FPGA design. The component is independent from surrounding logic, except the clock sources. Thanks to the registered I/Os and the trigger signal, the execution of the controller can be started upon the event of input data availability. This is best signalled by an upstream component like an ADC, which in turn relays its own trigger signal with period `T_Control`. The execution flow is continued in a deterministic way further downstream to trigger the output, constituting a real-time behaviour as illustrated in Figure 9.

The introduction of `T_Clk` as simulation base sample time allows to simulate all significant controller computation steps within the control system sample period. An decrease of the simulation sample period always increases the simulation complexity and duration, since now also the plant and environment would have to be simulated with increased rate, although they only produce a significant value change with a period of `T_Control`. To minimize this effect, the construction of a *multi-rate model* is recommended, allowing to simulate the controller partition with `T_Clk` while only executing the plant model once every `T_Control`, as in the platform-independent model.

Figure 15 shows a section of the model surrounding the x-axis controller module, illustrating the interface adaptions necessary to accommodate a HDL compatible, cycle-oriented partition. The data lines for values coming from the plant and environment model have been supplemented with *Data type conversions* (Convert) to provide the fixed-point values and *Rate transitions* (RT) to transition from the `T_Control` to the `T_Clk` time domain. Furthermore, the pulse generator block producing the execution trigger pulse with a width of `T_Clk` and period `T_Control` is shown.

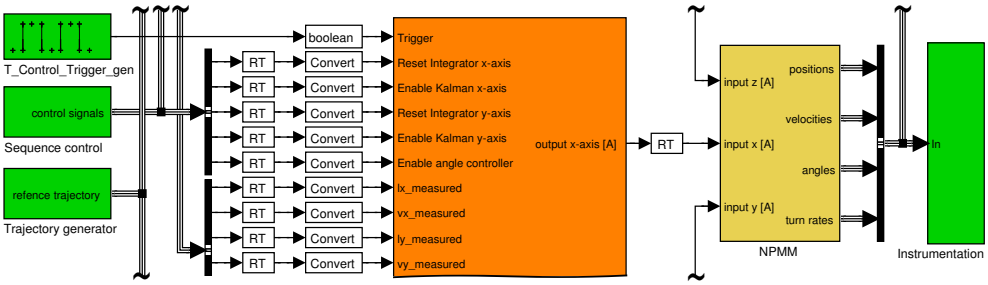


Fig. 15. Section of the multi-rate model containing the controller design simulated with higher time resolution.

4.3 Remarks

This section introduced two different implementable designs of the example controller - first, a largely combinational design with feedback registers to be clocked with `T_Control`; and second, an I/O registered design to be clocked with at least `T_Clk` and triggered with period `T_Control`. Both represent different styles of hardware implementation, depending on the grade of robustness, decoupling, universality or reusability, that is required for the component under development with regard to the system it is to be embedded in.

The introduction of the sample time `T_Clk` allows to simulate the controller execution cycles within the control system sample steps. In principle, `T_Clk` does not need to be more fine-grained, than necessary to simulate the N designed computation steps - i.e. $T\_Clk \leq 1/N \times T\_Control$. Of course, on the chip the component will be clocked with a much higher frequency, but a true cycle-accurate simulation of a multi-MHz clocked controller design is not feasible in the early design and validation stages of a control design with a sample rate of some kHz. Scaling up the resolution of `T_Clk` with continuing refinement of the execution path and logic stages is a way to find the balance between behavioural validation - "as accurate as necessary" - and simulation time - "as short as possible".

The controller model with registered I/O resembles only the first step into the direction of *synchronous component* design. In principle it is possible to insert registers after each operation step, thus assuming the total control over the execution of the algorithm in terms of timing and determinism. The subdivision of the data path by registers shortens the combinational paths and allows to drive the hardware design with a higher clock - in turn, the overall latency will increase due to the higher cycle count. Aiming on either, the smaller overall latency of combinational designs, or the higher frequency of synchronous designs, is possible by adapting advanced hardware design principles on modelling level, but exceeds the focus of this chapter.

5. From simulation model to implementation model

To summarize the activities discussed in this chapter, Figure 16 shows a detailed view of the *PIM-to-PSM transformation* as a part of the Simulink-based development flow from the introduction. Based on a controller partition embedded in a plant and environment model the refinement process towards a platform-specific execution model starts with the *data path design*. This iterative process substitutes the algorithmic blocks with constructs supported

by the code generation facilities. Additionally, all signals carrying the data are converted to fixed-point representations with as minimal bit widths as possible.

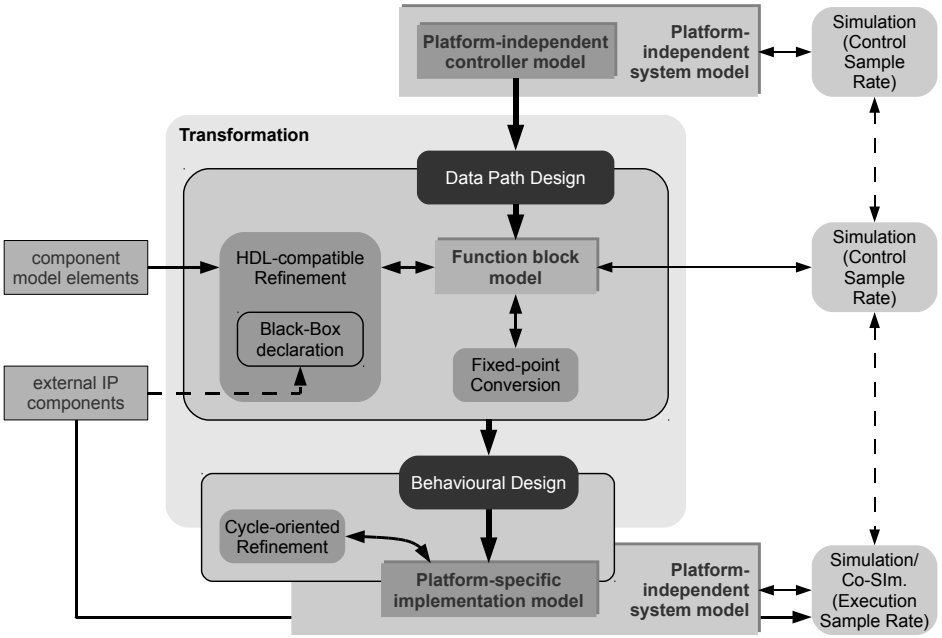


Fig. 16. The platform-specific controller modelling and validation flow in detail.

The substitutions and conversions necessary to fulfill the HDL-compatibility can produce deviations from the behaviour of the specified controller design. Therefore, a *simulative validation* of each modification towards a more platform-specific representation against the initial platform-independent specification is necessary, which might motivate an iteration of the initial controller design and parametrization. Validation of the fixed-point solution is a critical factor. Both the automated conversion using the Fixed-Point tools as well as the manual conversion rely on the specified data ranges for all operations. For operators with a-priori unknown output data ranges, the tools offer the option to determine the minima and maxima from the simulation results. Obviously, the consistency of this solution is based on the completeness of the simulation scenario - not covering the real minima/maxima of certain values will result in falsely scaled fixed-point representations. As a side note, as pointed out by The Mathworks (2011b), MATLAB has a restriction on simulation data widths of 128 bit, for larger word lengths a bit-accurate simulation is not possible, and the simulation results will deviate from the actual execution results.

The data path design approach also applies to the utilization of the vendor specific tools Xilinx, Inc. (2011) System Generator or Altera Corporation (2011) DSP Builder. But, when using these tools, from some point onwards during refinement process the model will have to be completely (re-)designed using the vendor-specifc block sets to create a platform-specific model, that is compatible to the respective code generation tool chains.

The activity of *behavioural design* applies for all mentioned tools in a similar fashion and concerns the design of the execution behaviour of the algorithm on the hardware.

Two possible designs have been introduced along with their properties concerning chip integration. Depending on the number of logic stages incorporated in the design, the platform-specific model has to be simulated with a more fine-grained simulation time resolution. The Mathworks (2011b) HDL Coder provides the possibility to create a cycle-accurate model of the HDL code derived from Simulink blocks configured to more advanced architectural implementations, which might introduce additional execution cycles not reflected in the original model. In those cases the execution control path of platform-specific model has to be further refined, regarding both the logic stages and the, simulation time resolution, to cycle-accurate simulation.

The simulative validation of the platform-specific model including a black-box component can also be conducted using HDL co-simulation, via links to external HDL simulators, like e.g. ModelSim. This method allows to analyse the behaviour of the black-box implementation and its interaction with the surrounding controller design on a cycle-accurate level. HDL co-simulation is also supported by the Xilinx System Generator. Without the HDL co-simulation option the validation of the black-box behaviour can be conducted only after leaving the modelling level by simulating the complete HDL generated design with instantiated black-box component with an HDL simulator. This course of action is aided by the HDL Coder by giving the option to generate an HDL testbench from the recorded Simulink simulation scenario. A method for validation of the implemented chip-design within the environment of the simulated plant model is the hardware-in-the-loop (HIL) simulation, that is supported by the vendor-specific tools DSP Builder and System Generator.

6. Conclusion

In conclusion it can be stated that MATLAB/Simulink supports the seamless top-down development flow for control designs with the intention of implementing the digital real-time controller on a reconfigurable logic platform. Within the extends of the Simulink block set and associated tools, and under consideration of the issues discussed in this chapter, the engineer can conduct the process of data path design by successive refinement and transformation of the controller model under constant validation against the surrounding plant model. The construction of the correct control flow of the algorithm's execution is a decisive engineering step, but it is well supported by the model-based approach. From a certain level of refinement, the usage of the HDL Coder supported block set or the the FPGA vendors' block sets provide the ability to construct and validate a hardware-oriented execution model. While the former allows a more integrated top-down modelling process, and the latter provide more platform-specific design and validation abilities, either tool facilitates in closing the gap between model-based design, automatic code generation and FPGA synthesis.

7. Acknowledgements

The work presented here is related to the research within the Collaborative Research Centre 622 "Nano-Positioning and Nano-Measuring Machines", funded by the German Research Council (DFG) under grant SFB 622 / CRC 622.

8. References

Altera Corporation (2011). DSP Builder Handbook Volume 1 : Introduction to DSP Builder.

Amthor, A., Zschack, S. & Ament, C. (2008). Position control on nanometer scale based on an adaptive friction compensation scheme, *2008 34th Annual Conference of IEEE Industrial Electronics*, IEEE, Orlando (USA), pp. 2568–2573.

Auger, D. (2008). Programmable hardware systems using model-based design, *2008 IET and Electronics Weekly Conference on Programmable Hardware Systems* .

Carletta, J. & Veillette, R. (2005). A methodology for FPGA-based control implementation, *IEEE Transactions on Control Systems Technology* 13(6): 977–987.

Caspi, P. & Maler, O. (2005). From control loops to real-time programs, *Handbook of networked and embedded control systems*, Birkhäuser, pp. 395–418.

Keating, M. & Bricaud, P. (2002). *Reuse Methodology Manual*, 3rd edn, Springer.

Krasner, J. (2004). Model-based design and beyond: Solutions for today's embedded systems requirements, *Analyst report, American Technology International* (January): 1–12.

Monmasson, E. & Cirstea, M. (2007). FPGA Design Methodology for Industrial Control Systems - A Review, *IEEE Transactions on Industrial Electronics* 54(4): 1824–1842.

Müller, M., Fengler, W., Amthor, A. & Ament, C. (2009). Model-driven development and multiprocessor implementation of a dynamic control algorithm for nanopositioning and nanomeasuring machines, *Journal of Systems and Control Engineering* 223: 417–429.

Pacholik, A., Klöckner, J., Müller, M., Gushchina, I. & Fengler, W. (2011). LiSARD: LabVIEW Integrated Softcore Architecture for Reconfigurable Devices, *2011 International Conference on Reconfigurable Computing and FPGAs (ReConFig '11), 30. Nov.- 02. Dec. 2011, Cancun (Mexico)*, IEEE Computer Society CPS, pp. 442–447.

Rushton, A. (2011). *VHDL for Logic Synthesis*, 3rd edn, John Wiley & Sons.

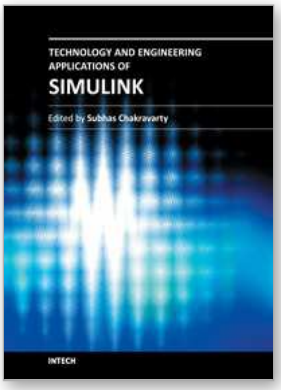
The Mathworks (2010). Simulink Fixed Point 6 User Guide.

The Mathworks (2011a). MATLAB/Simulink Online Documentation. URL: *http://www.mathworks.com/help/toolbox/simulink/*

The Mathworks (2011b). Simulink HDL Coder User's Guide R2011b.

Xilinx, Inc. (2011). Xilinx System Generator for DSP User Guide.

Zschäck, S., Amthor, A., Müller, M., Klöckner, J., Ament, C. & Fengler, W. (2010). Integrated system development process for high-precision motion control systems, *2010 IEEE International Conference on Control Applications*, IEEE, pp. 344–350.



Technology and Engineering Applications of Simulink

Edited by Prof. Subhas Chakravarty

ISBN 978-953-51-0635-7
Hard cover, 256 pages
Publisher InTech
Published online 23, May, 2012
Published in print edition May, 2012

Building on MATLAB (the language of technical computing), Simulink provides a platform for engineers to plan, model, design, simulate, test and implement complex electromechanical, dynamic control, signal processing and communication systems. Simulink-Matlab combination is very useful for developing algorithms, GUI assisted creation of block diagrams and realisation of interactive simulation based designs. The eleven chapters of the book demonstrate the power and capabilities of Simulink to solve engineering problems with varied degree of complexity in the virtual environment.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821