Development of a Real-Time Simulation Environment for Heating Circuits on the Basis of the UDOO developing board utilizing Scilab / Xcos

Master's Thesis

by

Patrick Kohl

Faculty of Life Sciences
Department of Renewable Energies
&
Department of Process Engineering

Patrick Kohl

Development of a Real-Time Simulation Environment for Heating Circuits on the Basis of the UDOO developing board utilizing Scilab / Xcos

Master's thesis handed in as part of the master's examination for the degree programme of *Renewable Energy Systems* (M.Eng.) at the departments of *Umwelttechnik* and *Verfahrenstechnik* in the faculty of *Life Sciences* at the university *Hochschule für Angewandte Wissenschaften Hamburg*.

Hand in date:	June 9, 2015
Submitted by:	Patrick Kohl (B.Sc.), Student number: 1867164
1^{st} supervisor:	Dr.-Ing. G. Lichtenberg
2^{nd} supervisor:	Dipl.-Ing. Nico Mock

Keywords

hardware-in-the-loop (HIL), real-time simulation, heating system, PI-controller, UDOO, DDC4200, Scilab, Xcos, Arduino

Abstract

This thesis outlined the development of a real-time, hardware-in-the-loop (HIL) simulation of a heating system for a non-residential building on the basis of the UDOO development board. The result was a low-cost HIL simulation of an open-source software modelled heating system, controlled by a PI-controller in a feedback loop. The components of the HIL included the DDC automation station and the software heating model running on an OS Ubuntu laptop computer. A discrete-time model of the heating system was developed in the open-source Scilab/Xcos programming language on the basis of an existing continuous-time Matlab/Simulink® model. An Arduino IDE sketch was implemented, managing the I/O and serial-USB data transfer. An electrical interface was developed for the physical signal transfer. First simulation results prove the operational readiness of the HIL. The control variable is plotted over the simulation time showing a curve progressions converging towards a set-point temperature. The modular technical platform created enables future developers to extend the existing controlling functions of the HIL simulation.

For Mom & Dad

Acknowledgement

I thank Kai Kruppa for the outstanding support he has offered me, whilst developing this thesis. I wish to express my gratitude to my Professor Dr. G. Lichtenberg for his valuable guidance during my studies. My gratitude also goes to the team of the electronics lab of the HAW-Hamburg, especially to Dipl.-Ing. Nico Mock and Dipl.-Ing. Jan-Klaas Böhmke for their help in solving many practical problems and helping me navigate through the shallows of Linux. My special thanks goes to Lydia Schulze Heuling, Jona Schwarz, Fabian Kording, Lea Drolshagen and and Erik Pietsch for their constructive feedback, proofreading and moral support. I am most grateful for the love, tolerance, patience and abiding support of Sandra Janßen, without it I could not have finished this thesis. Finally, my heart-felt gratitude goes to my parents for their selfless support during my studies.

I hereby declare that I produced the present work myself and only with the help of the indicated aids and sources.

Hamburg, June 9, 2015 Patrick Kohl

Contents

List of Figures

List of Tables

Nomenclature

Symbol	Units	Description
α	modulates heat input (0...1)	$[-]$
$boilerInit$	starting temperature for T_{boil}	$[K]$
$onOff$	boiler switch (1=on, 0=off)	$[-]$
c	specific heat capacity of water	$[\frac{J}{kgK}]$
$emerStop$	indicator for emergency shutdown	$[-]$
$emerStopTime$	emergency stop time	$[s]$
$emerTempStop$	emergency shut-down temperature	$[K]$
$InitTempCon$	initial temp. of consumer	$[K]$
k_b	building heat capacity	$[\frac{J}{K}]$
$k_{b,a}$	heat transfer coeff. build. $\rightarrow$ ambient	$[\frac{W}{m^2K}]$
k_{boil}	power loss coeff.	$[\frac{kW}{K}]$
$k_{r,b}$	heat transfer coeff. radiator $\rightarrow$ building	$[\frac{W}{m^2K}]$
P_{min}	min. boiler output	$[kW]$
P_n	rated boiler output	$[kW]$
Q	thermal energy	$[J]$
$\dot{Q}_{in}$	heat power input	$[kW]$
$\dot{Q}_{out}$	heat power ouput	$[kW]$
ρ	density of water	$[\frac{kg}{m^3}]$
ts	sampling time	$[s]$
T_a	ambient temperature	$[K]$
T_{amb}	ambient temperature of boiler	$[K]$
T_b	building temperature	$[K]$

$T_{b,set}$	set building temperature	$[K]$
ΔT_b	temperature difference	$[K]$
T_bInit	starting value build. temperature	$[K]$
T_r	return flow temperature	$[K]$
T_s	supply flow temperature	$[K]$
$\dot{V}$	input and/or output volume flow	$[\frac{m^3}{s}]$
$\dot{V}_a$	volume flow amplitude	$[\frac{m^3}{s}]$
V_b	volume boiler	$[m^3]$
$\dot{V}_m$	mean volume flow	$[\frac{m^3}{s}]$
V_{Ver}	volume consumer	$[m^3]$
w	set-point temperature	$[^\circ C]$

1 Introduction

In striving to develop more energy efficient heating systems the motivation for developing model-based methods for heating solutions in non-residential buildings, compare favourably high to the heuristic methods, which often exist to date [11]. Often, the current practise in controlling a heating system is that the boiler settings and thermostat programming are determined arbitrarily [14]. If the boiler is oversized, even a modulated boiler may turn to an on / off system. Furthermore, when sudden changes to the system occur, like a rapid dropping of the outside temperature, more intelligent controllers in the control loop like MPC (model predictive control) is needed [16]. The challenge is that heating systems are individually planed, so that a component-based modelling platform would be desirable.

Currently, an existing continuous-time heating system model derived from first principle equations exists [10]. Set parameters of this model were generated from measurement data of a real non-residential building [12]. However, for effective MPC computing a discrete-time model is needed [5]. A numerical programming language for calculating dynamic systems, like the Matlab/Simulink clone Scilab/Xcos masters this task. Furthermore, an automation station with set-point controller functionality is provided by the company Kieback & Peter GmbH & Co.KG to the HAW-Hamburg. Thus a software model simulating thermal behaviour of a heating system may be controlled by a PI-controller. Using the functionality of an I/O-board, like an UDOO single board computer to transfer the data across the interfaces, a simplified hardware-in-the-loop (HIL) simulation is set-up. This real-time environment is ready to be used for modular, component orientated development of heating systems. As it is advantageous for such a modular platform to be cost-efficient, the software implemented is open source and the UDOO single board computer, open-hardware. This thesis discusses a set-up of such a real-time HIL simulation.

In chapter 2 the motivation for the HiL is described, introducing the components and the controlling signals for data transfer across the interface. This is followed by a description of the applied theory for the controller and the thermodynamic principles of a heating system model in chapter 3. Firstly, an overview of the control system is presented and the PI-controller theory formulated. Secondly, the physical principle equations of the heating system are outlined and the continuous-time model discretized, resulting in difference equations for the system states of each component for the heating system model.
The following chapter 4 gives a detailed description of the hardware and software components needed for the HIL simulation. The Arduino IDE for serial and I/O data management, the programming language Scilab/Xcos and the OS Ubuntu is introduced. The

UDOO development board, DDC4200 automation station and laptop computer specifications are presented.
Chapter 5 outlines the I/O-signal route between the HIL simulation components, as well as the serial and I/O functions of Scilab and the Arduino IDE. Furthermore, the electrical interface for the data transfer and the calculations for the required circuity is specified.
In chapter 6 the results of the working HIL simulation are presented and discussed, accomplishes underlined and further investigations proposed.
A troubleshooting guide for errors and bugs identified during projecting is presented in chapter 7, while chapter 8 rounds off the discussion with the conclusion and an outlook.

2 HIL system

2.1 Motivation

A HIL simulation is a technique to test complex real-time embedded systems [8]. In this case the thermal behaviour of a heating system model is simulated, controlled by an external PI-controller within a feedback control loop. The objective of this project is to create a low-cost, open source developing platform which may be modularly extended. Therefore, the UDOO single board computer was chosen for its high processing performance and relative low cost. Similarly, for the simulation software the numerical programming language Scilab/Xcos on 32-bit Ubuntu OS was selected. Scilab/Xcos is a popular, free-of-cost Matlab/Simulink clone and like the implemented open source Ubuntu OS, has a large web-based development community [20]. Thus an open source platform is created for developing individual model based solutions for specific heating systems within the scope of the Observe research project [11].

2.2 Overview

The HIL simulation reproduces the thermal behaviour of a heating system in real-time by software and hardware components. A DDC4200 automation station is connected via a UDOO single board computer to a Xcos software model of a heating system running on a laptop computer, as described in chapter 4.2.
The data transfer between the DDC4200 automation station and the UDOO development board is handled by an electrical interface, as described in chapter 5.3. The development software Scilab/Xcos and Arduino IDE are outsourced on the laptop, further detailed in chapter 4.1.
Figure 2.1 shows the data transfer between the HIL components, specifically the generated control variable T_s and the actuating variable α as described in chapter 5.2. The Xcos heating system model generates T_s and accepts the actuating variable α from the PI-controller. The signal transfer between the UDOO and the laptop is processed via the USB to UART-serial bridge, whilst the data between the UDOO and the DDC4200 is processed via the UDOO's I/O-pins. The I/O-transfer of data is described in chapter 5.

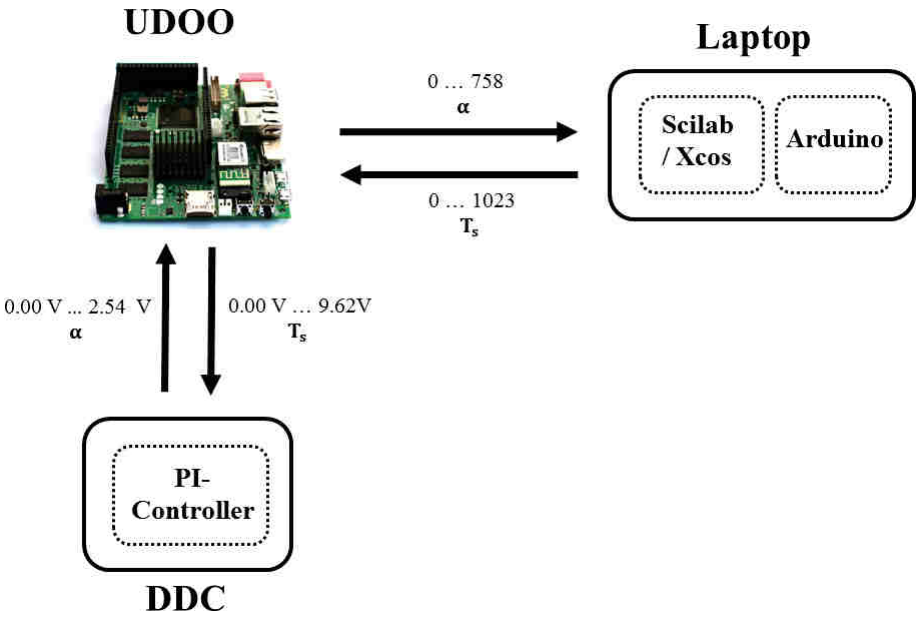


Figure 2.1: Schematic of HIL simulation, including the signal route of control variable T_s and actuating variable α (Source: own sketch, photo: [24])

3 Theory: controller and heating system model

This chapter outlines the control system under discussion and introduces the theoretical principles of the controller and the heating system. A detailed description of each component of the Xcos heating system model is given and the calculation for the system state variables defined.

3.1 Control system

The control system under investigation is made up of the plant and the controller. The plant or heating system consists of a *boiler*, a *pump* and the *building*, within which the *consumer* in shape of a radiator is installed. The heating system model is programmed in Xcos and generates the control variable *supply flow temperature* T_s (also state variable) which is transmitted to the controller. The equations for this non-linear system are described in chapter 3.5.
The controller is a software module installed on the DDC4200 automation station as a fixed set-point controller (PI-Controller), referred to in chapter 3.2. It returns a modulated signal (actuating variable) α which regulates the burner of the boiler.
The basis for the following discussion is the continuous-time heating system model (building model, consumer model, pump model and boiler model) developed by K. Kruppa. The continuous-time model components are detailed in the HeatLib [10], a library of Simulink® functions blocks for simplified heating systems. The set parameters for the heating system model described in the following are generated from measurement data from an official building of the district council of Düsseldorf, Germany [12]. The discretization step of the continuous-time model is described in chapter 3.4.

The following state variables for the components of the heating system model under discussion are defined as:

- T_b : building temperature ⇒ calculated by the building model
- T_r : return flow temperature ⇒ calculated by the consumer model
- T_s : input flow temperature ⇒ calculated by the boiler model.

Figure 3.1 shows a schematic overview of the control system including the state variables.

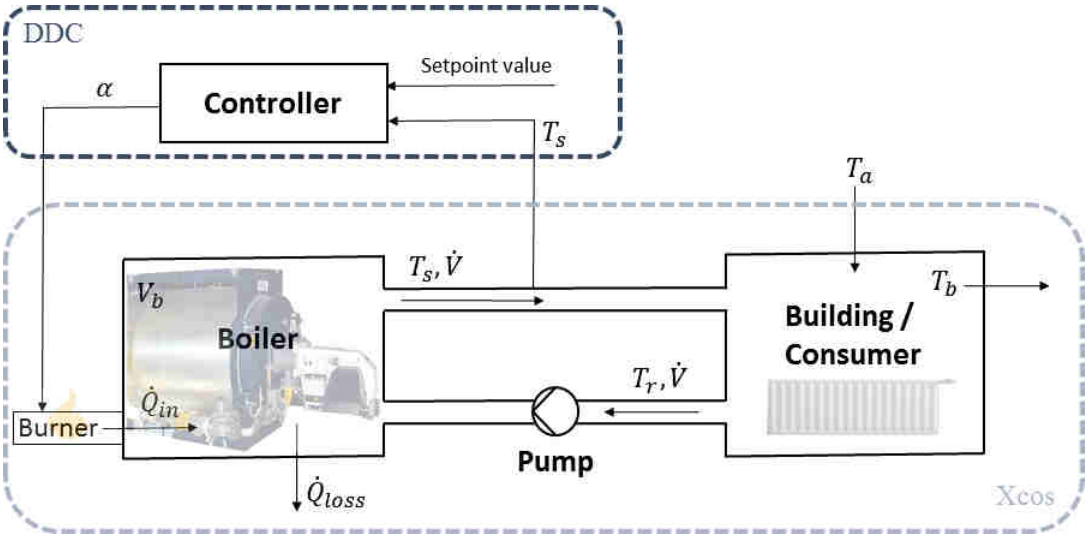


Figure 3.1: Control system schematic including the plant and controller. The dotted line shows the location of the components in the HIL simulation. (Source: own sketch, photo: www.wikipedia.org)

3.2 PI-Controller of the DDC4200 automation station

The controller implemented in the control system in Figure 3.1 of chapter 3.1 is a fixed set-point controller, set to a two term proportional and integral controller (PI-Controller) within the DDC automation station software. Figure 3.2 shows the controller embedded into a feedback control-loop [17]. The control variable T_s and the actuating variable α from the Xcos heating system model are assigned y and u respectively. In reference to chapter 5, the control variable y or T_s is passed from the Xcos heating system via the UDOO to the DDC, while the actuating variable u or α is sent back to the heating system. The set-point variable w in $°C$ is set in the controller software. The difference of the set-point variable w and the control variable y is named control deviation e which is passed to the PI-controller. The actuating variable u is described in the figure 3.3. The disturbance variable d is the ambient temperature T_a of the Xcos building model within the heating system described in chapter 3.5.1.

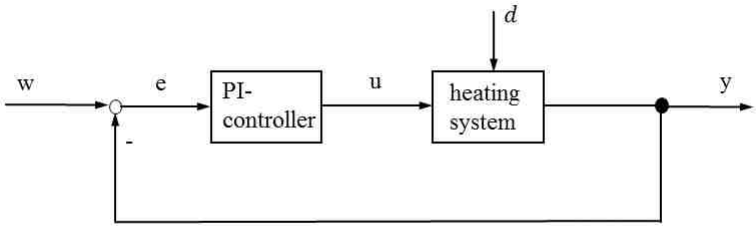


Figure 3.2: Feedback control loop with set-point variable w, control deviation e, actuating variable u, disturbance variable d and control variable y (Source: own sketch)

Figure 3.3 shows the block diagram of a PI-characteristic. The PI-controller combines both positive characteristics of a P- and I-controller. The control deviation e passes through both the proportional and integral component. The respective outputs are superimposed to the actuating variable u.

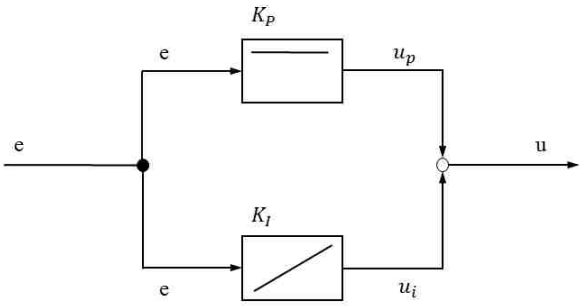


Figure 3.3: PI-controller block diagram (Source: own sketch)

The reset time T_i and the proportional gain K_p are the characteristic parameters of a PI-controller and function as tuning parameters. According to BEIER following relationships are described [2]. The proportional gain:

$$K_P = \frac{u_p}{e} \tag{3.1}$$

and the integral gain:

$$K_I = \frac{\Delta u_i}{e} \cdot \frac{1}{\Delta t} \tag{3.2}$$

with Δt the integration time, define the reset time T_i when the actuating variables u_p and u_i are equated, resulting in:

$$T_i = \frac{K_P}{K_I}. \tag{3.3}$$

The reset time T_i thus provides a weight to the integral gain K_I so that the influence of integral action can be independently adjusted. Therefore a greater reset time minimizes the integral gain K_I of the PI-controller and vice versa, while the relationship of the P-and I-part is fixed.

3.3 Heat power balances

The physical basis of the heating system model is the energy balance described by the first law of thermodynamics, the conservation of energy states [21]. The total energy of an isolated system is constant, therefore energy is not lost, but may change it's form. As no mechanical work is done, one component of the system supplies another with thermal energy. This thermal energy or heat Q is stored in the system and is described by

$$Q = c \cdot \rho \cdot V \cdot T, \tag{3.4}$$

with c being the specific heat capacity, ρ the density at temperature T with volume V. The medium used here is potable water. Taking into account the time dependency of temperature and volume, the first derivative of equation 3.4, whilst applying the product rule, results in

$$\dot{Q} = \rho \cdot c(\dot{V}T + V\dot{T}), \tag{3.5}$$

with $\dot{Q}$ as heat power. Depending on the environment for calculating the heat power $\dot{Q}$, either the temperature T or the volume V is considered constant over time, thus converging towards zero when differentiated. Accordingly, the partial derivatives of equation 3.5 by case are described by

$$\dot{Q} = \begin{cases} \rho \cdot c \cdot \dot{V} \cdot T & \textbf{for } T = const \\ \rho \cdot c \cdot V \cdot \dot{T} & \textbf{for } V = const. \end{cases} \tag{3.6}$$

The heating system model components described in the following chapters do not take into account all the physical effects encountered in reality, but offer a simplified model keeping the core functionality in place.

Equation 3.6 is the starting point for calculating the state variables of each of the heating system components at a nodal point in the Xcos block diagram, described in the following chapters. In the Xcos diagrams the notation e.g. dV is used for $\dot{V}$, as the latter may not be depicted. Likewise x' is used for $\dot{x}$ for the derivative of the state variables.

3.4 Discretization of continuous heating system model

Typically, the Xcos solver is able to numerically solve discrete-time or continuous IVPs (initial value problems). To construct a model, which is able to implement advanced controller models like MPC's (model predictive control) [5], the continuous ODE's from the HeatLib are transformed to difference equations. This is achieved by implementing the 1^{st} order EULER forward discretization method [13]. The resulting finite number of point model problem is then adapted to Xcos code.

In general terms the starting point in each case is an explicit ODE of first order $\dot{x} = f(t, x)$ with initial values $x(0)$ for x, at a start time $t(0)$: $x(t(0)) \stackrel{!}{=} x(0)$ or IVP. The continuous function f has a unique solution to the IVP. The function f is then discretized using EULER forward. The signal $x(t)$ is sampled at a fixed time interval or step size, here labelled the sampling time ts (set throughout to 1 s). Therefore the next approximative function value $x(k+1)$ for the EULER method from time $t(k)$ to $t(k+1) = t(k) + ts$ is

$$x(k+1) = x(k) + ts \cdot \underbrace{f(t(k), x(k))}_{\mathtt{Y}} \quad (k = 1, 2, ..., n) \tag{3.7}$$

with n the value of the final integration time of the simulation. Figures 3.4 and 3.5 show the Xcos continuous-time ($\dot{x}$) and discrete-time ($x(k+1)$) function blocks needed respectively for solving differential or difference equations. The input of each block diagram represents the right hand side of either the continuous-time or discrete-time system describing equation.

Figure 3.4: Continuous-time integral function block in Xcos (Source: Xcos)

Figure 3.5: Discrete-time delay function block in Xcos (Source: Xcos)

In the following chapters the `Y` part of equation 3.7 is replaced with the right hand side of the ODE, describing the state variable for each respective heating system component. Finally, the resulting discretizised Xcos block diagrams of each component are connected together to form a complete heating system model, as shown in figure 3.18 of chapter 3.6.

3.5 Components of the heating system model

3.5.1 Building model

The building model is considered to be a simple so-called 1-zone model, calculating the inside building temperature T_b which is assumed to be uniform throughout. Furthermore, the model calculates the heat capacity demand $\dot{Q}_{out}$, needed by the consumer model to determine the return temperature T_r of the flow back to the boiler. Within the building, radiators adopt the same temperature as the return temperature. It is assumed that the radiators emit heat to the building model, described by the heat transfer coefficient $k_{r,b}$. Heat loss to the outside occurs determined by the heat transfer coefficient $k_{b,a}$ and is dependent on the ambient temperature T_a. The building model itself has a certain heat capacity k_b. Figure 3.6 shows the building model with the radiator and its main input/output parameters.

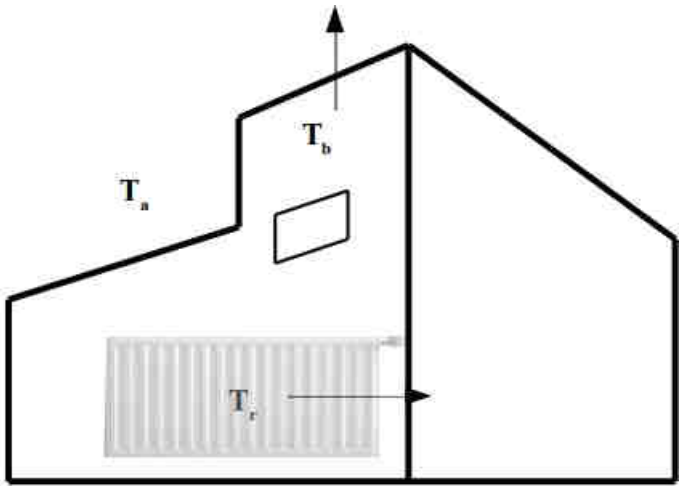


Figure 3.6: The building model with the radiator as consumer (Source:own sketch, photo: www.wikipedia.org)

The tables 2 and 3 show the input and output variables of the building's Xcos model and parameters with set values.

Table 2: Building model input and output variables

Input	Description	Unit
T_a	ambient temperature	$[K]$
T_r	return flow temperature	$[K]$
Output	**Description**	**Unit**
T_b	building temperature	$[K]$
$\dot{Q}_{out}$	heat demand of consumer	$[kW]$

Table 3: Building parameters

Parameter	Description	Set value	Unit
k_b	building heat capacity	10^{10}	$[\frac{J}{K}]$
$k_{r,b}$	heat transfer coeff. radiator $\rightarrow$ building	$2.5 \cdot 10^4$	$[\frac{W}{m^2K}]$
$k_{b,a}$	heat transfer coeff. build. $\rightarrow$ ambient	$4.7 \cdot 10^4$	$[\frac{W}{m^2K}]$
T_bInit	starting value build. temperature	293	$[K]$

The thermal behaviour of the building model over time by calculation of the system state

variable *building temperature* T_b may be described by

$$\dot{T}_b = \overbrace{\frac{k_{r,b}}{k_b}(T_r - T_b)}^{\dot{Q}_{out}} - \overbrace{\frac{k_{b,a}}{k_b}(T_b - T_a)}^{\dot{Q}_{loss}}. \tag{3.8}$$

Application of the EULER method described in equation 3.7 results in the difference equation

$$T_b(k+1) = T_b(k) + Ts\left(\frac{k_{r,b}}{k_b}(T_r(k) - T_b(k)) - \frac{k_{b,a}}{k_b}(T_b(k) - T_a)\right). \tag{3.9}$$

The Xcos block diagram, representing equation 3.9, is shown in figure3.7.

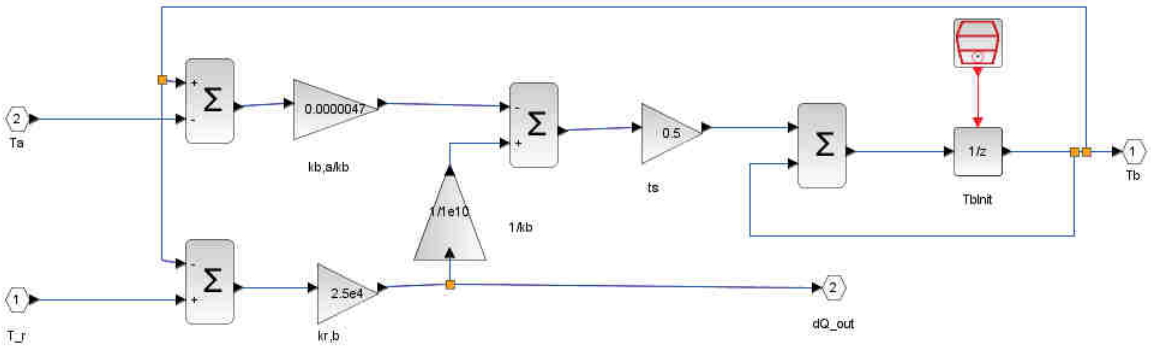


Figure 3.7: Xcos block diagram of building model (Source: Xcos)

3.5.2 Pump model

The pump model determines the volume flow $\dot{V}$ in the heating system model, setting the flow dependant on the difference between the actual building temperature T_b and the desired set building temperature $T_{b,set}$. If the building temperature T_b is below the set building temperature $T_{b,set}$ the pump generates higher volume flow resulting in a higher mean volume flow $\dot{V}_m$. Likewise, if T_b is higher than $T_{b,set}$ the pump provides a volume flow less than $\dot{V}_m$. The ideal pump characteristic shown in figure3.8 has a linear trend. If the absolute deviation between building temperature and set building temperature $|T_b - T_{b,set}|$ is greater than the temperature difference ΔT_b, the volume flow $\dot{V}$ is in saturation on its minimum, respectively maximum value $\dot{V}_m \pm \dot{V}_a$. The volume flow amplitude is described by $\dot{V}_a$. The pump flow $\dot{V}$ is essentially regulated by the thermostatic valves, installed on the radiators (consumer) in the building.

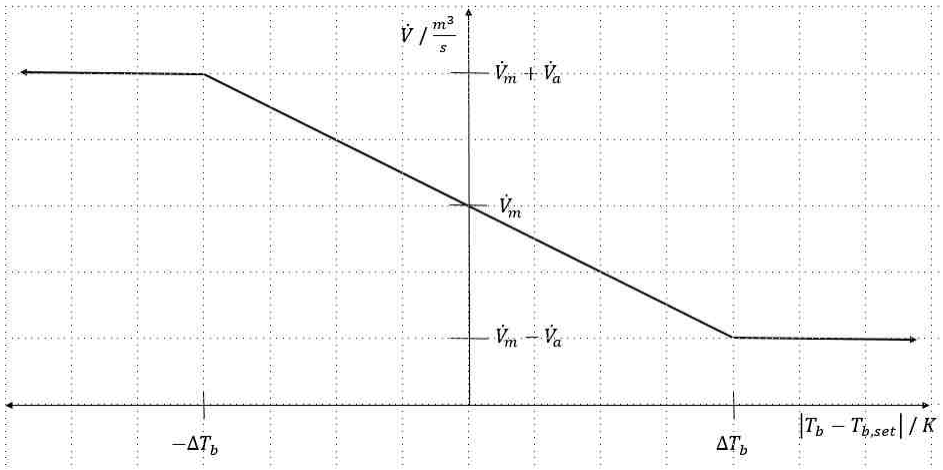


Figure 3.8: Ideal pump characteristic (Source: own sketch)

The tables 4 and 5 show the input and output variables of the pump and the parameters with set values.

Table 4: Pump model input and output variables

Input	Description	Unit
T_b	building temperature	$[K]$
Output	**Description**	**Unit**
$\dot{V}$	volume flow	$[\frac{m^3}{s}]$

Table 5: Pump parameters

Parameter	Description	Set value	Unit
$\dot{V}_m$	mean volume flow	$4.75 \cdot 10^{-3}$	$[\frac{m^3}{s}]$
$\dot{V}_a$	amplitude of volume flow	$3.25 \cdot 10^{-3}$	$[\frac{m^3}{s}]$
$T_{b,set}$	building temperature set-point	294	$[K]$
ΔT_b	temperature difference between set and actual value	4	$[K]$

The case-equations

$$\dot{V} = \begin{cases} \dot{V}_m - \dot{V}_a & \textbf{if } \overbrace{T_b - T_{b,set} \geq \Delta T_b}^{\texttt{min. saturation}} \\ \dot{V}_m + \frac{\dot{V}_a}{\Delta T_b}(T_b - T_{b,set}) & \textbf{if } \overbrace{-\Delta T_b < T_b - T_{b,set} < \Delta T_b}^{\texttt{ideal linear characteristic}} \\ \dot{V}_m + \dot{V}_a & \textbf{if } \overbrace{T_b - T_{b,set} \leq -\Delta T_b}^{\texttt{max. saturation}} \end{cases} \tag{3.10}$$

describe the ideal linear characteristic of the volume flow $\dot{V}$ within set limits. There is no discretization of equation 3.10, as there are no states changing over time in the model. The figure 3.9 shows the Xcos block diagram for the pump model depicting the case equation 3.10.

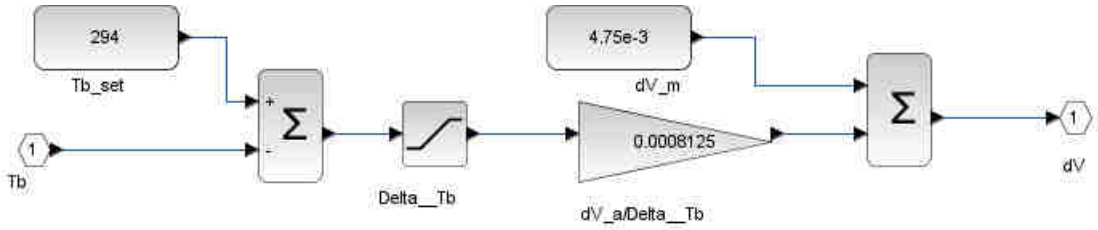


Figure 3.9: Xcos block diagram of pump model (Source: Xcos)

3.5.3 Consumer model

Within the simplified consumer model only the input/output behaviour relative to the heat producer (boiler model) is discussed, neglecting other parasitic influence such as irradiation, window area, ventilation or other heat producers, like electrical appliances. The supply volume flow $\dot{V}$ with temperature T_s from the boiler model flows into the consumer with initial temperature $InitTempCon$ heating the building model by power $\dot{Q}_{out}$ in the process. Therefore, the consumer model (or radiator) provides the heat for the building model. The consumer model's volume V_{Ver} is assumed constant, containing the total water volume of the consumer's heating circuit. The cooled off water then exits the consumer as return flow $\dot{V}$ with temperature T_r. Input and output flow $\dot{V}$ is assumed equal as pressure loss in the piping and valves is neglected.

Figure 3.10 shows the consumer model with the main input and output variables.

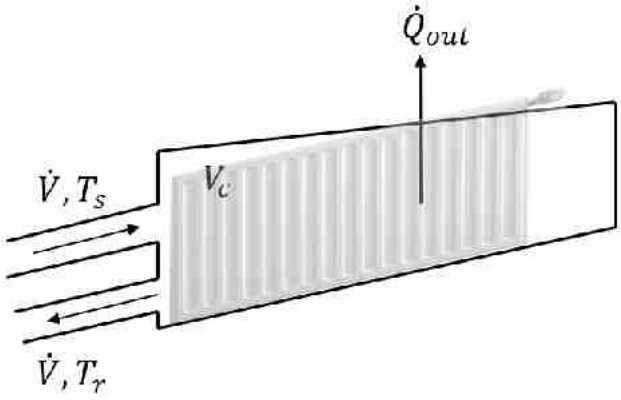


Figure 3.10: Xcos Consumer model with input and output parameters (Source: Xcos)

The tables 6 and 7 show the input and output variables of the consumer model and the parameters with set values.

Table 6: Consumer model input and output variables

Input	Description	Unit
$\dot{Q}_{out}$	heat power input	$[kW]$
T_s	temperature of input volume flow	$[K]$
$\dot{V}$	input volume flow	$[\frac{m^3}{s}]$
Output	**Description**	**Unit**
T_r	return flow temperature	$[K]$
$\dot{V}$	return flow	$[\frac{m^3}{s}]$

Table 7: Consumer parameters

Parameter	Description	Set value	Unit
V_{Ver}	volume consumer	5	$[m^3]$
c	specific heat capacity of water	4182	$[\frac{J}{kgK}]$
ρ	density of water	1000	$[\frac{kg}{m^3}]$
$InitTempCon$	initial temp. of consumer	335	$[K]$

The thermal behaviour of the consumer model over time by calculation of the state variable *return flow temperature* T_r is described by

$$\dot{T}_r = \frac{\overbrace{c \cdot \rho \cdot \dot{V}(T_s - T_r)}^{\texttt{heat input}} - \overbrace{\dot{Q}_{out}}^{\texttt{consumed heat by building}}}{c \cdot \rho \cdot V_{Ver}}. \tag{3.11}$$

Application of the EULER method described in equation 3.7 results in the difference equation

$$T_r(k+1) = T_r(k) + ts \left(\frac{c \cdot \rho \cdot \dot{V}(k)(T_s(k) - T_r(k)) - \dot{Q}_{out}(k)}{c \cdot \rho \cdot V_{Ver}} \right). \tag{3.12}$$

The Xcos block diagram representing the difference equation 3.12 is shown in figure3.11.

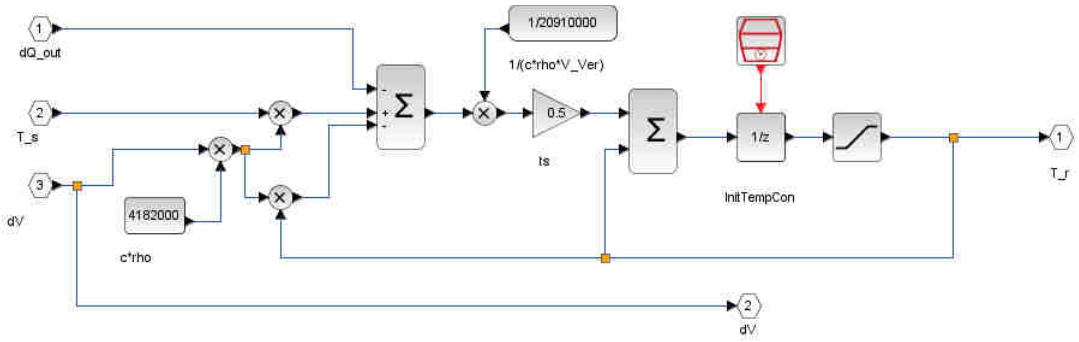


Figure 3.11: Xcos block diagram of consumer model. (Source: Xcos)

3.5.4 Boiler model

The return flow temperature T_r determined by the consumer model is input to the boiler model. The boiler returns the supply volume flow $\dot{V}$ at temperature T_s. The water of the boiler with volume V_b , initial temperature $boilerInit$ and ambient boiler temperature T_{amb}, is heated by multiplying a modulated signal α (0 ... 1) with the boiler's rated power P_n, whilst assuming no stratification in the boiler. The resulting heating power $\dot{Q}_{in}$ is input to the boiler cycle.
The input heating power $\dot{Q}_{in}$ is limited to the rated boiler power P_n and may not fall under a certain minimum P_{min}. The $onOff$ switch activates or deactivates power input. Heat loss is summarized as power loss $\dot{Q}_{loss}$ which describes the heat transfer from the boiler to the environment. The specific power loss coefficient k_{boil} is a proportional constant and is

set here to zero, thus disregarding any power loss. The boiler ambient temperature T_{amb} is set to fixed value.
A safety mechanism prevents the boiler of taking damage when the supply flow temperature T_s exceeds an emergency shut-down temperature $emerTempStop$. In this event, the boiler is switched off, the emergency stop mode indicator $emerStop$ is set to 1 for the period of the emergency stop time $emerStopTime$. As soon as $emerStopTime$ has passed, heating input power with the previous modulation signal, from the previous step, is reactivated. Figure 3.12 shows the boiler model with its main input/output variables.

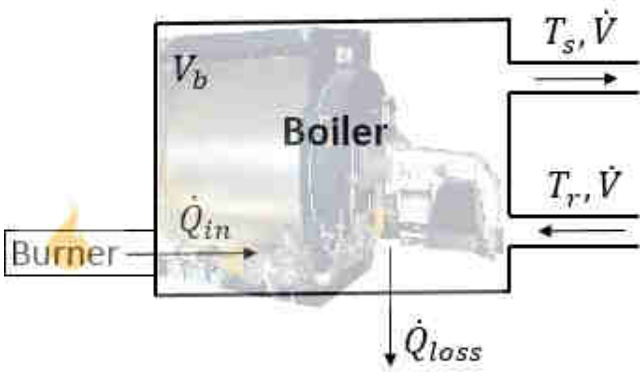


Figure 3.12: The boiler model with the input and output parameters. (Source:own sketch, photo: www.wikipedia.org)

The tables 8 and 9 show the input/output variables of the boiler model and parameters with set values.

Table 8: Boiler model input and output variables

Input	Description	Unit
T_r	return flow temperature	$[K]$
$\dot{V}$	return volume flow (build.)	$[\frac{m^3}{s}]$
α	modulates heat input (0...1)	$[-]$
$onOff$	switch (1=on, 0=off)	$[-]$
Output	**Description**	**Unit**
T_s	supply flow temperature	$[K]$
$\dot{V}$	outgoing volume flow	$[\frac{m^3}{s}]$
$emerStop$	indicator for emergency shutdown	$[-]$

Table 9: Boiler parameters

Parameter	Description	Set value	Unit
V_b	volume boiler	1.05	$[m^3]$
c	specific heat capacity of water	4182	$[\frac{J}{kgK}]$
ρ	density of water	1000	$[\frac{kg}{m^3}]$
$emerTempStop$	emergency shut-down temperature	368	$[K]$
$emerStopTime$	emergency stop time	10	$[s]$
$boilerInit$	starting temperature for T_s	338	$[K]$
T_{amb}	ambient temperature of boiler	303	$[K]$
k_{boil}	specific power loss coeff.	0	$[\frac{kW}{K}]$
P_n	rated boiler output	$1.1 \cdot 10^3$	$[kW]$
P_{min}	min. boiler output	200	$[kW]$

As previously described, the input flow is assumed equal to the output flow, thus assigned one variable $\dot{V}$, as pressure drop through the piping and valves is disregarded. The thermal behaviour of the boiler model over time by calculation of the state variable *supply flow temperature* T_s is described by

$$\dot{T}_s = \frac{\overbrace{\dot{Q}_{in}}^{\text{power input}} \quad \overbrace{-k_{boil}(T_s - T_{amb})}^{\text{power loss}} + \overbrace{c \cdot \rho \cdot \dot{V}(T_r - T_s)}^{\text{power difference between flow and return}}}{c \cdot \rho \cdot V_b}. \tag{3.13}$$

Application of the EULER method described in equation 3.7 results in the difference equation

$$T_s(k+1) = T_s(k) + ts \left(\frac{\dot{Q}_{in}(k) - k_{boil}(T_s(k) - T_{amb}) + c \cdot \rho \cdot \dot{V}(k)(T_r(k) - T_s(k))}{c \cdot \rho \cdot V_b} \right). \tag{3.14}$$

The Xcos block diagrams representing the terms of the difference equation 3.14 are shown in figure3.13 to figure 3.17.

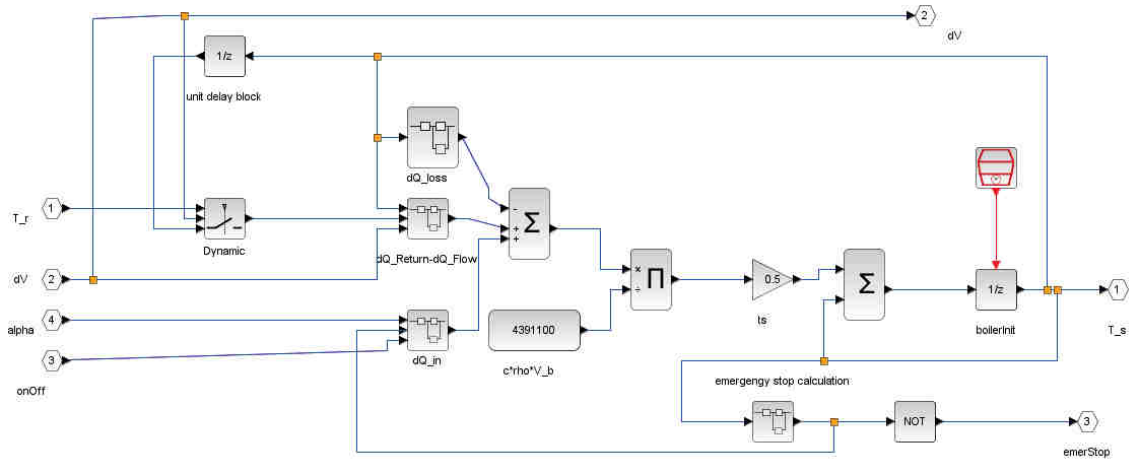


Figure 3.13: Xcos block diagram of boiler model (Source: Xcos)

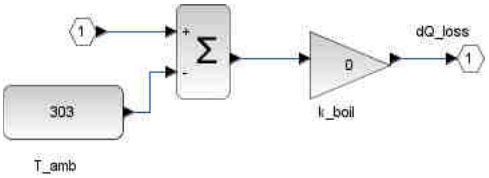


Figure 3.14: Xcos block diagram of thermal power loss (Source: Xcos)

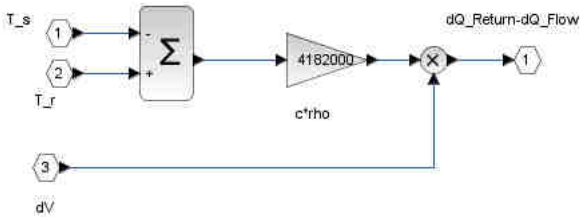


Figure 3.15: Xcos block diagram of power difference between flow and return (Source: Xcos)

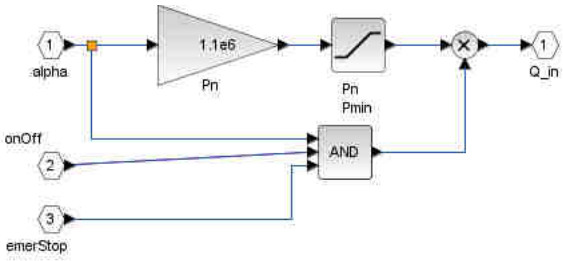


Figure 3.16: Xcos block diagram of input power (Source: Xcos)

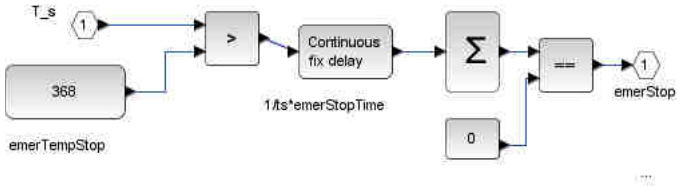


Figure 3.17: Xcos block diagram of emergency stop calculation (Source: Xcos)

3.6 Block diagram of the heating system model

The figure 3.18 shows the Xcos block diagram of chapters 3.5.1 to 3.5.4 adjoined together to form a simplified heating system model. Included is the serial communication super-block discussed in chapter 5.2.1.

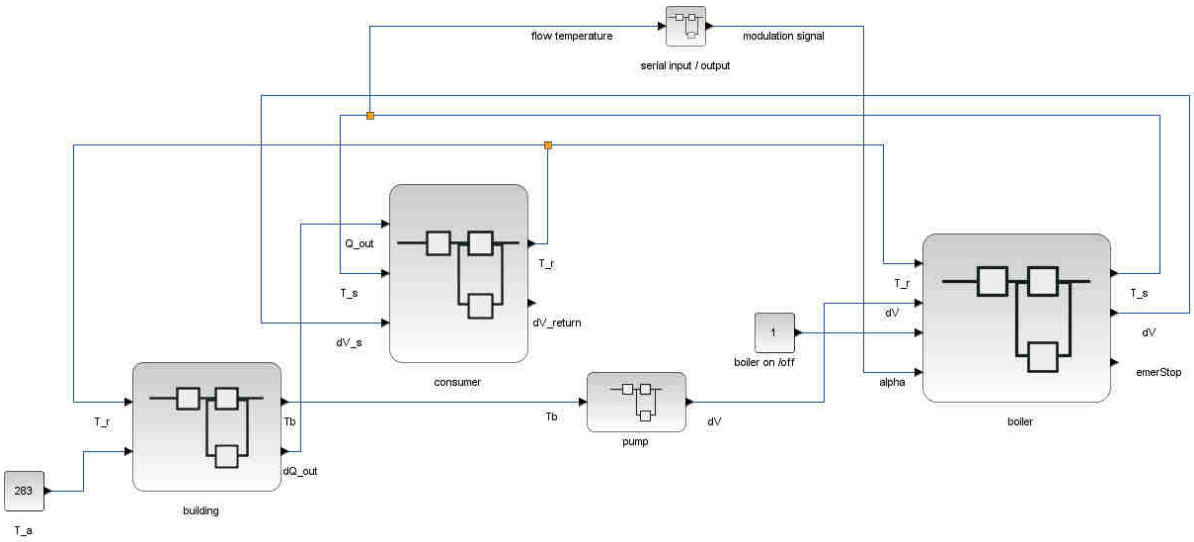


Figure 3.18: Xcos block diagram of heating system model including serial communication super-block (Source: Xcos)

4 Hard- and Software used in the HIL

This chapter describes the hard- and software compontents need for the HIL-simulation, introducing Arduino IDE, Scilab/Xcos, Linux Ubuntu, UDOO and DDC4200.

4.1 Software in the HIL

4.1.1 Arduino IDE

The Arduino IDE® (integrated development environment) is a cross-platform application written in a C/C++ dialect using a specific JAVA programming environment [4]. The free installation files may be downloaded from the official Arduino web-page [1]. To use the Arduino IDE from an external laptop computer, connected to the UDOO via UART-serial bridge, a program patch must be downloaded and installed as described on said web-page. The code produced in the Arduino IDE is called a "sketch". The language includes pre-defined analog and digital I/O functions for serial data processing.
The basic structure of a cyclic executive Arduino sketch comprises of two functions, *void setup* and *void loop* [18]. The former is used to initialize settings like opening a serial line with a set baud rate (data bits per second) for serial data transmission and is called once. The latter is called repeatedly once uploaded until the UDOO computer is switched off. The sketch itself is uploaded to the SAM3x flash memory. Within the *loop* function, script code is placed for reading and writing to the GPIO-pins or the UART-serial line as described in chapter 5.2.2. For reading in- and output from the serial line buffer the Arduino IDE's inbuilt serial monitor is useful for debugging. Figure 4.1 shows the opening screen of the Arduino IDE with an "empty" sketch, including only the initial *setup* und *loop* functions, as well as indicators for important icons.

Figure 4.1: Arduino sketch - basic structure (Source: [1])

In the Arduino IDE menu "Tools" tab, under the sub-menu item "Board", the Arduino Due (Programming Port) is selected for uploading the sketches via the mini USB-to-serial converter as described in chapter 4.2.1). Once the laptop computer is connected to the UDOO, the correct physical serial port */dev/ttyUSB0* is selected on the "Port" sub-menu, the same which is set in the *openserial_udoo* serial function of Scilab as described in chapter 5.2.1. Refer to chapter 7 for troubleshooting a connection problem occuring while upload sketches.

4.1.2 Scilab/Xcos

Scilab is an open source, numerically orientated programming language [20]. The optional free package Xcos is part of Scilab and is used for calculations of dynamical systems, continuous or discrete. Both packages use the same syntax, whereas the Xcos block diagram sketches have direct access to the Scilab functions and variables. The 5.5.1 version is implemented here, compiled from the source code for 32-bit GNU/Linux and installed on both the laptop computer and the UDOO single board computer. Hence, the Scilab/Xcos script programmed may be transferred from one platform to another without any data loss or compatibility issues. Scilab is compiled from the source files, as to date no readily compiled binary package for the latest Scilab version is available for the ARM Cortex-A9 architecture, as described in chapter 4.2.1. Refer to appendix 9.1.2 for a step-by-step

instructions on the compilation process for Ubuntu OS.

All the time dependent Xcos function blocks used are discrete-time. The sampling time ts sets the cycle clock pulse of each simulation iteration in XCOS. It is set in the *Sample-CLK* function blocks of the Xcos heating system block diagrams, *boiler model*, *consumer model* and *building model*, as described in chapter 3.5. *SampleCLK* is used instead of $CLOCK\_c$ as all events triggered by this block run synchronous due to the specific internal compilation method. Additionally a 1 s offset is set in the *SampleCLK* block to delay the simulation for better stability results.
The Xcos blocks which need activation all inherent the event activation (*inherent*=1), except the *delay* element used for calculating the system states. It is triggered by the clock pulse set by the sampling time ts of the *SampleCLK* function block.
To have access to Scilabs serial functions for data transfer across the UART-serial line, the Serial Communication Toolbox is installed via the Scilab ATOMS module manager [7] found in the "Applications" tab in Scilab. For installation error troubleshooting refer to chapter 7.

4.1.3 Linux Ubuntu

The operating system (OS) running on the UDOO is an adaptation of Lubuntu 12.04 with LXDE, a lightweight desktop environment designed to save precious processing power and includes readily available "hard-float" hardware support for higher efficiency [9]. The UDOO is ready to use once a bootable microSD (8 GB) is created from an ISO-image file of Lubuntu, downloadable from the official UDOO website [24].
The laptop computer runs on Ubuntu 14.04.02 LTS 32-bit as OS which is freely available to download from the Canonical website [23].
Prior to compiling Scilab 5.x on the laptop (or UDOO) the dependency packages like JAVA etc. need to be installed. Refer to appendix 9.1.1 for a sequence of administrator run commands from a terminal to remove any unmet (broken) dependencies from the Ubuntu OS.

4.2 Hardware in the HIL

4.2.1 UDOO single board development platform

The UDOO is a single board computer designed as an open hardware platform equipped with two processors for prototyping projects [24]: the Freescale i.MX6 ARM Cortex-A9 quad core 1 GHz processor (datasheet: 9.3.3), also used in the Rasberry Pi ™ single board computer and an Arduino Due [18] compatible ATMEL SAM3x ARM processor (datasheet: 9.3.2). The i.MX6 handles the Ubuntu OS and other applications like Scilab/Xcos or the Arduino IDE. The SAM3x gives the UDOO the option to be utitlized as a stand-alone Arduino Due board and manages the I/O (input/output) data transfer via the GPIO (general purpose I/O) pins.

The UDOO features 76 fully available, 3.3 V compliant (max. 3.445 V), GPIO pins. Of these I/O pins, 8 feature as ADC's (analog-to-digital converter) and two as DAC's (digital-to-analog converter) [3]. The default setting for the ADC is a 10-bit resolution, while the DAC's is 8-bit. The resolution of both the ADC and DAC may be changed from within an Arduino sketch as described in chapter 5.2.2. Due to the Atmel SAM3x architecture, the DAC's output signal ranges from a minimum (1/6)*V_{ADVREF} to a maximum (5/6)*V_{ADVREF}, with $V_{ADVREF} = 3.445\ V$. Figure 4.2 shows plotted measurements of output voltages 0.571 V...2.87 V on the DAC0 pin for input values 0...1023, proving a linear characteristic for the DAC.

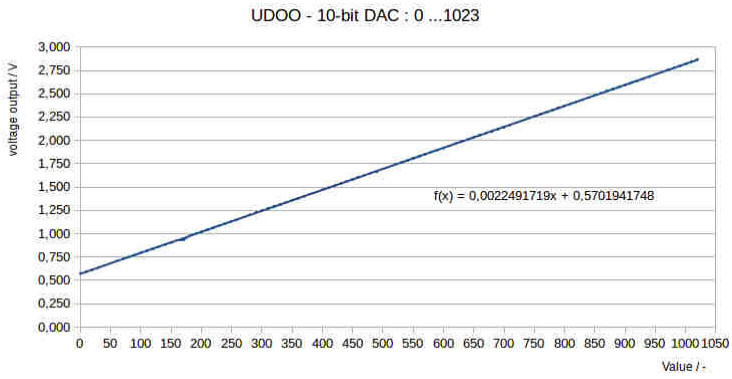


Figure 4.2: 10-bit DAC output voltage on DAC0 between 0.571 V...2.87 V for values 0...1023 (Source: Libre Calc.)

The output signal (DAC0) is adjusted implementing a differential amplifier circuit to up-

scale the output voltage signal, as described in chapter 5.3.
The full potential of the UDOO board is utitlized using the always active UART-serial (universal asynchronous receiver transmitter) line between the two processors. This serial line may be accessed from an external connection via the mini-USB serial bridge as depicted in figure 4.3. The serial data transfer sequentially transmits individual bits of data, sending and receiving only once the previous cycle is finished [19]. Both processors may run a script (or event managers) to read or write data to the serial line. For example, a serial port is opened via a Scilab script running on the i.MX6 writing some data into the serial buffer, while the Arduino sketch processes data from the buffer and forwards it to the I/O-pins via the SAM3x functionality. Figure 4.3 shows a schematic of the two processors of the UDOO with the UART-serial line terminals.

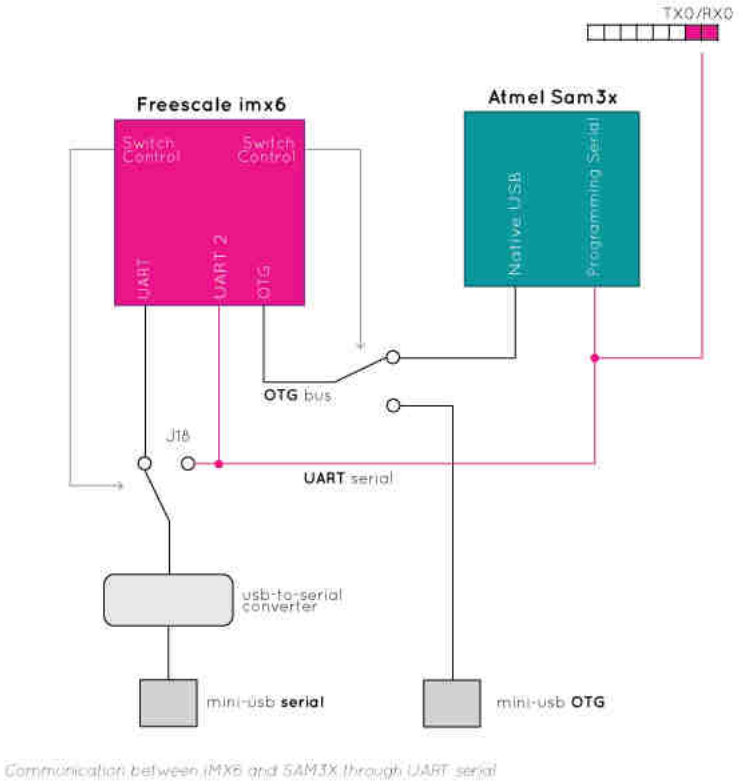


Figure 4.3: Data transfer between SAM3x and i.MX6 via UART-serial (Source: [24])

Other features of the UDOO include a HDMI port, Ethernet RJ45, 1 GB RAM and a GPU Vivante. Refer to appendix 9.3.1 for full specifications of the UDOO.
For the HIL simulation discussed in this thesis, the i.MX6 is essentially bypassed, using only the functionality of the SAM3x. This means the UDOO is implemented as a stand-

alone Arduino Due board [1], for the serial communication to the laptop computer and the data transfer to the DDC across the UDOO's GPIO pins. Running the Scilab/Xcos model simulation on a stand-alone UDOO is generally possible, as processing performance of the i.MX6 is sufficient. However, for development and debugging purposes the above mentioned setup is preferred, in order to utilize the extra processing performance of the laptop. This is to avoid high latency and resulting time loss during simulation runs. Nonetheless, identical versions of Linux 32-bit Scilab are installed on both the UDOO and the laptop computer so that scripts are interchangeable for both systems.
The UDOO is connected to the laptop via the mini-USB port (CN6) with the USB cable provided. To determine the USB-port address from the laptop, whilst the UDOO is connected, the command *dmesg* may be used from an Ubuntu terminal logged in as root user.

4.2.2 DDC4200 automation station and laptop computer

DDC4200

The DDC 4200 is a stand-alone automation station developed by the company Kieback & Peter GmbH & Co.KG for controlling and monitoring functions for non-residential building heating systems [15]. It implements the BACnet (building automation and control networks) protocol for building automation. Kieback & Peter has provided one unit to the HAW-Hamburg for research purposes in line with the Observe project [11].
The DDC features a variety of bus connection, interfaces and I/O terminals, refer to appendix 9.3.4 for full specifications. For the purpose of the HIL implemented, two of the available 24 analog input/output pins are used for sending/receiving signals. They are assigned the pin-object P.02 and P.01 respectively in the software. The pins are switchable to various sensor types (Pt100, Ni100 etc.). For both input an output the sensor type is selected to 0 V...10 V DC. The input 0...10 V signal correlates to a temperature signal of 0°C ...100 °C. Figure 4.4 shows a plotted measurement result of voltage signals in 5 V increments, generated by a voltage source, input on pin-object P.02 and the resulting temperature readings in °C. The result proves the linear characteristic of the DDC's 0 V...10 V input sensor.

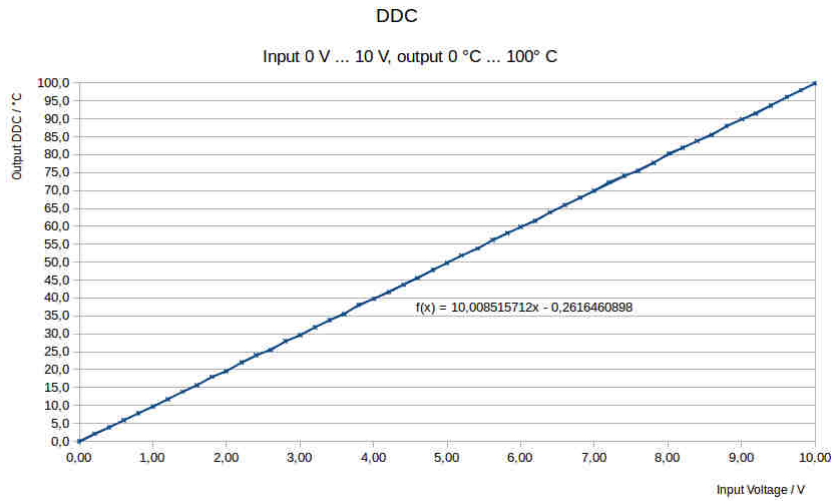


Figure 4.4: Temperature readings from the DDC menu in $°C$ with 0 V ...10 V input on P.02, in 5 V increments (Source: Libre Calc.)

A set-point controller software module is installed in the DDC and set to a PI-controller as described in chapter 3.2. Within the controller software, the input pin-object P.02 must be set to the source of the set-point controller for the control variable, whereas the actuating signal is set as the source for the outgoing pin-object P.01.

Laptop computer

The laptop computer used is a Lenovo T420s model running on an Intel® quad Core™ i5-sandy-bridge gen. series chip with 4 GB RAM.

5 I/O data transfer in the HIL

This chapter outlines the I/O signal route between the HIL-components and gives an introduction to the software functions for I/O and UART-serial data-transfer. The electric interface including calculations is detailed.

5.1 I/O signal route

In reference to figure 2.1 of chapter 2.2, the control variable T_s is calculated by the Xcos heating system model to a value of 0...1023 which is passed via the serial line to the UDOO. The 10-bit DAC maps this signal to a 0.57 V... 2.87 V signal which is amplified by the differential amplifier circuit and passed on to the pin object P.02, 0-10V input sensor, of the DDC.
The pin object P.01 of the DDC outputs a 0 V... 10 V signal, or actuating variable α from the PI-controller. It is downscaled to a max. 2.54 V signal by a voltage divider and passed onto the UDOO as described in chapter 5.3. The 10-bit ADC maps the signal from the voltage divider to a 0...758 value and passes it via the UART-serial line to the Xcos heating system model on the laptop computer.

5.2 I/O Software functions

5.2.1 Scilab/Xcos functions

The Scilab/Xcos serial functions realize the data transfer via the UART-serial line between the UDOO and the laptop computer, whilst the signal processing via the UDOO's I/O-pins is performed via the functions of the Arduino IDE.

The Scilab serial functions are accessible once the *Serial Communication Toolbox* [20] is installed via the Atoms module manager as mentioned in chapter 4.1.2. The opening and closing of a serial port is established from within the Scilab script, while reading and writing to the serial port is performed in the Xcos serial communication super-block of the heating system model shown in figure 3.18. A super-block in Xcos is a simplification block encompassing more than one function block.
The task of the super-block is to write the control variable T_s to the serial port and to read the actuating signal α from the serial port. The control variable T_s is also a system state variable, the modulated signal α regulates the burner of the boiler as described in

chapter 3.5.4.

The following core serial communication functions are implemented within the Scilab

- *openserial_ udoo*
- *xcos_ simulate*
- *closeserial*

and for the Xcos *read / write* communications super-block

- *readserial*
- *writeserial*.

The *openserial_ udoo* function is a modification of *openserial* from the Scilab functions library. It assigns a handle to the serial port, opening serial communication via the mini-USB-serial port to the UART serial line of the SAM3x and i.MX6 on the UDOO, as described in chapter 4.2.1. The full code with the amended line changÃŋng the physical USB-port address, is appended in 9.2.4. The port number and baud rate set in the argument must be identical to the port number set in the Arduino sketch, as described in chapter 5.2.2. The default 9600 bit/s is upheld. A delay of 1 s using the function *xpause* is set after calling *openserial* for a stable serial line connection. To close the serial connection the function *closeserial* is called. The *xcos_ simulate* function calls the Xcos diagram specified in the argument and initiates the simulation.
The Xcos function block *scifunc_ block_ m* is used to embed Scilab code in a Xcos diagram. Within this block the *readserial* and *writeserial* functions read, respectively write a string, one character at a time, in ASCII format into the buffer of the serial port [9]. The cycle is dependant on the set sampling time t_s as a parameter of the *SampleCLK* Xcos function block, as mentioned in chapter 4.1.2. The full code of the *scifunc_ block_ m*is appended in 9.2.2.

Figure 5.1 shows the *read / write* function block handling the data transfer over the serial port connection from within the serial communication super-block of figure 3.18.

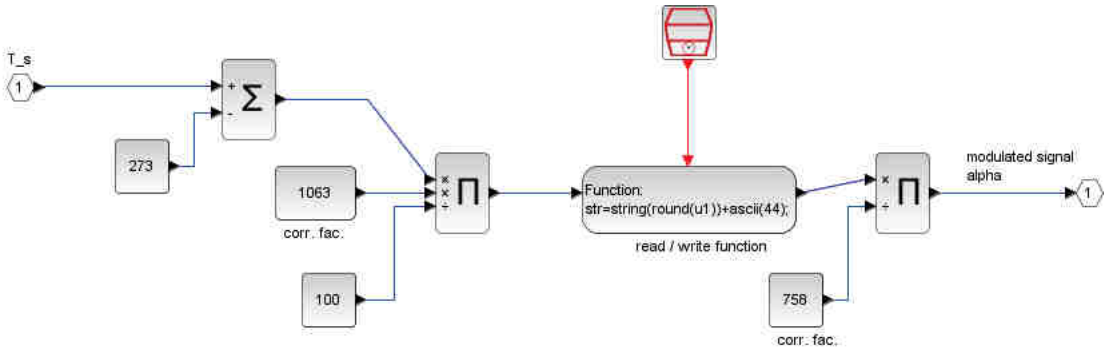


Figure 5.1: Serial communication block diagram (Source: Xcos)

The state variable (or control variable) T_s from the boiler model is converted from K to $°C$ by subtracting 273. This value is then multiplied by the correction factor 1063 and divided by 100. The resulting value between 0...1023 is passed to the *read / write* function block. The proportional calculation to determine the correction factor yields

$$\begin{aligned} 9.62V &= 1023 \\ 10V &= x \\ \Rightarrow x &\approx 1063. \end{aligned}$$

The 0...1023 value is mapped by the UDOO's 10-bit-DAC to 0.57 V...2.87 V which is amplified by the differential amplifier circuit to 0 V...9.62 V, further explained in chapter 5.3.

On the output side of the *read / write* function block, the *readserial* function reads a character string from the serial buffer and converts it to an integer using the function *strtod*. This integer value is divided by 758 resulting in the modulated signal α between 0...1, regulating the heat power input of the boiler, as described in chapter 3.5.4. The value of 758 corresponds to the maximum voltage 2.543 V which is converted by the UDOO's 10-bit ADC, as described in chapter 5.3.

5.2.2 Arduino functions

The Arduino sketch implements functions for the serial communication to the external laptop computer, as well as for the data transfer across the I/O-pins of the UDOO to the DDC. The core of the Arduino IDE library [1] for serial communication are

- *Serial.begin*

- *Serial.available*
- *Serial.parseInt*
- *Serial.println*

and for the data transfer across the UDOO's I/O-pins

- *analogRead*
- *analogWrite*.

For an overview of an Arduino sketch refer to chapter 4.1.1. Within *void setup* the *serial.begin* function initializes the serial connection to a set baud rate specified in the argument, identical to the one set in *openserial_udoo* of the Scilab script described in chapter 5.2.1. The following functions may be placed within the void loop method. The *Serial.available* function listens on the UART-serial line if any data is available in the serial buffer. If so, *Serial.parseInt* reads the next integer, while *Serial.println* writes an integer value received by *analogRead* to the serial buffer. The *analogRead* function reads the value from the analog input pin A0 which is mapped by the ADC to a value 0 ... 758, as described in chapter 5.3. The resolution of the ADC is set to 10-bit by *analogReadResolution*.
The function *analogWrite* writes the parsed integer from *Serial.parseInt* to the DAC output pin DAC0. The received 0...1023 value from the Xcos script is mapped to 0.57 V ... 2.87 V, as described in chapter 5.3. A delay of 20 ms is set with the function *delay* to give the DAC time to reset after each iteration. The DAC's resolution is set to 10-bit with *analogWriteResolution*. The full code of the Arduino sketch is appended in 9.2.1.

5.3 I/O hardware: electrical interface

The DDC's outgoing voltage signal is higher as the compatibility range of the UDOO's I/O pins. Similarly, the outgoing signal range off the DAC-pin of the UDOO is to low for the 0-10 V DDC sensor to map the signal to a 0-100°C temperature range. The following presents a solution to these obstacles and creates an electrical interface.
Figure 5.2 illustrates the circuit diagram of the UDOO connected to external circuitry. The laptop computer is connected to the UDOO via USB as described in chapter 4.2.1. On the input side of the UDOO a voltage divider is connected, whilst on the output side a differential amplifier circuit.

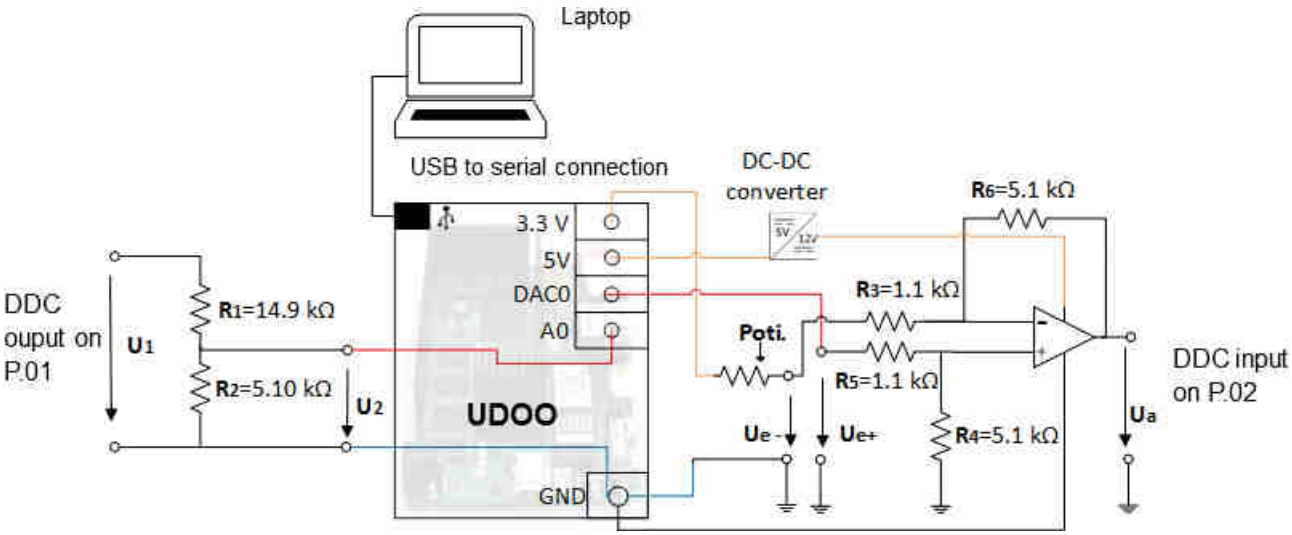


Figure 5.2: I/O interface schematic of the UDOO connected to a voltage divider on input and a differential amplifier on output (Source:own sketch - not to scale, photo: www.udoo.org ©)

The maximum 10 V signal output by the DDC-pin must be downscaled, as the UDOO's analog pin only accepts a maximum 3.445 V without taking damage. For this purpose a voltage divider circuit is soldered onto an external stripboard. On the output side of the UDOO the 10-bit-DAC outputs a 0.57 V ... 2.87 V signal. To eliminate the offset of 0.57 V and amplify the remaining 2.3 V to 9.62 V, a differential amplifying circuit [6] is connected, soldered on an external stripboard. As the sensor in the DDC is set to 0-10 V, 9.62 V is analogous to 96 $°C$.

The outgoing 0 V... 10 V signal U_1 from the DDC pin P.01 is split by the voltage divider to a 0.00 V... 2.54 V signal, refer to chapter 5.4.2 for the calculation. This voltage signal U_2 is input to the UDOO's analog pin A0, where the signal is mapped by the 10-bit-ADC to a value of 0...758. This is the range averaged over 5000 measurements.
On the output side the 0...1023 value is mapped by the 10-bit-DAC on the DAC0 pin of the UDOO to an output voltage signal ranging from 0.57 V to 2.87 V. The DAC's characteristic is due to the Atmel SAM3x specific architecture described in chapter 4.2.1. For the differential amplifier circuit a potentiometer (refer to datasheet 9.3.6) is placed in series with the input resistor of the non-inverting input $(-)$ of the operational amplifier (refer to datasheet 9.3.5). The potentiometer, powered by the 3.3 V UDOO power supply, is adjusted so that the offset U_{e-} of 0.57 V is subtracted and the remaining voltage U_{e+} of 2.3 V amplified. Refer to chapter 5.4.1 for the calculation.
A 2 W DC-to-DC converter (refer to datasheet 9.3.7) is connected to the UDOO's 5 V

power supply, upscaling the 5 V to 12 V to power the operational amplifier. Subtracting the off-set voltage and amplifying the remainder results in a final output voltage U_a of the circuit from 0.01 V...9.62 V.
For setting up and testing the connection it is useful to have readily available: two Fluke^{TM} multimeters, a couple of banana plugs, testing terminals and some cables.

Figure 5.3 shows a photo of the UDOO development board connected to the differential amplifier circuitry, while figure 5.4 shows a photo of the voltage divider circuitry connected to the DDC4200.

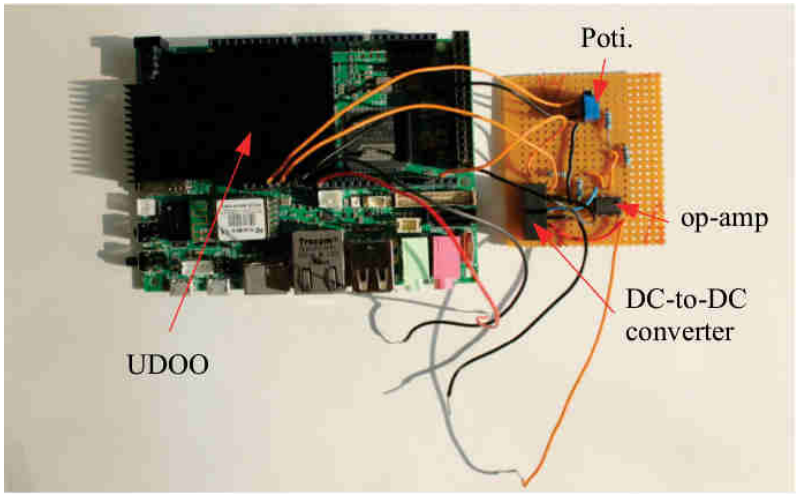


Figure 5.3: Photo of UDOO connected to stripboard with differential amplifier (Source:own photo)

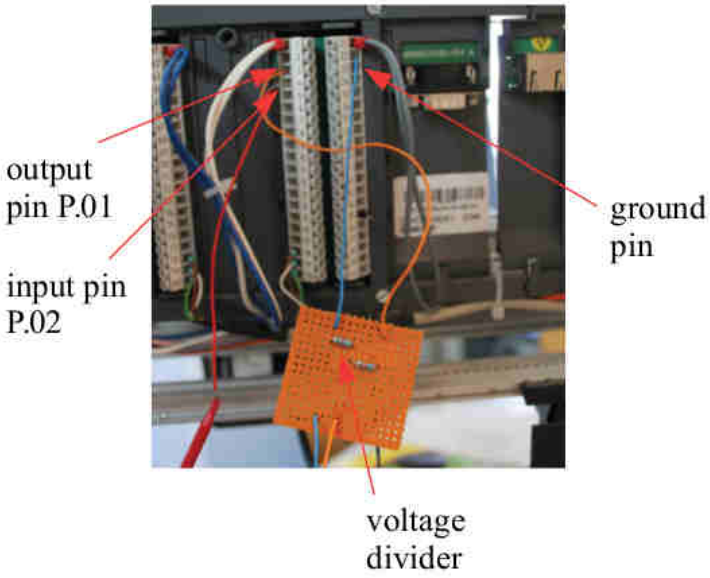


Figure 5.4: Photo of stripboard connected to DDC (Source:own photo)

5.4 Electrical interface: calculations

5.4.1 Differential amplifier

The differential amplifier circuit is connected to the output of the UDOO as described in figure 5.2 of chapter 5.3. In a differential amplifier circuit the operational amplifier is wired to function as an inverting and non-inverting amplifier simultaneously. The relationship between input U_{e+}, difference U_{e-} and output U_a voltage is described by [22]

$$U_a = \frac{(R_3 + R_6)R_4}{(R_5 + R_4)R_3} \cdot U_{e+} - \frac{R_6}{R_3} \cdot U_{e-}. \tag{5.1}$$

If the resistances are selected so that $R_3 = R_5$ and $R_4 = R_6$, then equation 5.1 simplifies to

$$U_a = \frac{R_6}{R_3} \cdot \underbrace{(U_{e+} - U_{e-})}_{X} \tag{5.2}$$

so that U_a is a proportionate to the relation of R_6 to R_3. The potentiometer is adjusted so

that $U_{e-} = 0.57\ V$ removing the offset cause by the DAC. The term X of equation 5.2 thus equals 2.87 V - 0.57 V = 2.3 V. Initial resistances were selected so R_6 = 4.7 $k\Omega$ and R_3 = 1.1 $k\Omega$, corresponding to a gain factor of 4.3. By measurements a gain factor of 3.9 was determined. The lower as expected gain factor is explained by thermal losses occurring in the operational amplifier. Therefore, the resistance R_6 by rule of proportion is increased to 5.1 $k\Omega$, resulting in an output voltage U_a of 9.62 V which suffices for this purpose. Thus the resulting voltage output U_a of 0 V...9.62 V corresponds to a temperature of 0°C... 96.2 °C, when converted by the DDC's 0-10 V sensor.

5.4.2 Voltage divider

Before the input of the UDOO a voltage divider circuitry is connected, as described in figure 5.2 in chapter 5.3. Figure 5.5 shows a magnification of the voltage divider circuit.

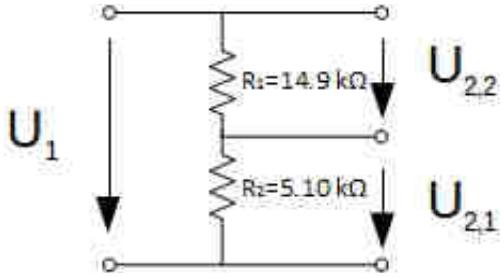


Figure 5.5: Voltage divider reducing the output voltage from the DDC (Source:own sketch)

According to the voltage divider rule [22] under Ohm's law, the relationship of voltage and resistance in the above circuit is described as

$$U_{2,1} = U_1 \cdot \frac{R_2}{R_1 + R_2}. \tag{5.3}$$

Inserting R_1 = 14.92 $k\Omega$ and R_2 = 5.10 $k\Omega$ into 5.3 with U_1 = 10 V, results in $U_{2,1}$ = 2.55 V. Due to heat losses in the resistors, the actual maximum output for U_2 is 2.54 V.

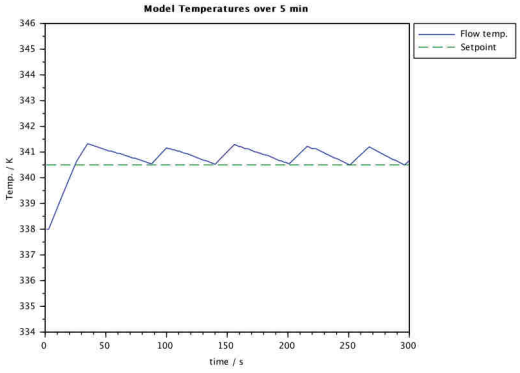
The UDOO's I/O-pin input current is limited to 50 mA [24]. To check if the current stays within the limit, Ohmic law is used,

$$I = \frac{U_{max}}{R_{tot.}}. \tag{5.4}$$

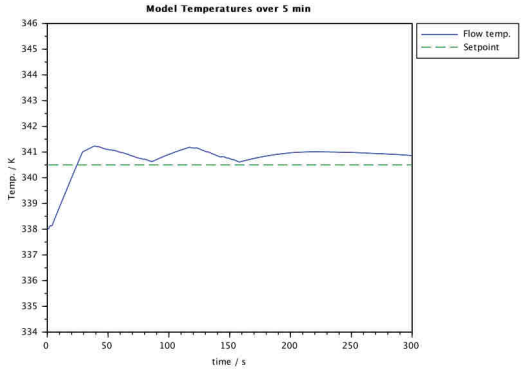
Inserting $U_{max} = 10\ V$ and $R_{tot} = 20.02\ k\Omega$ into 5.4 results in $I = 0.5\ mA$ which is a factor 100 below the limit of 50 mA.

6 Simulation results of test environment and discussion

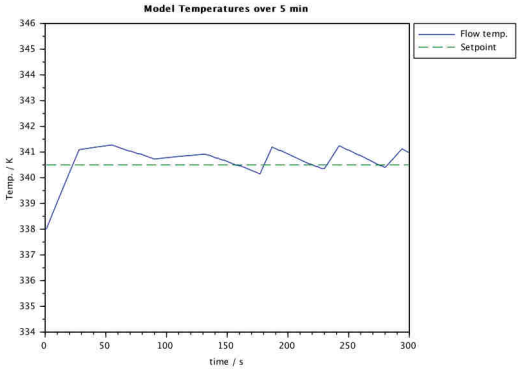
The first set of simulation results for the HIL simulation is described in the following. To simulate the Xcos script in real-time, the *real time scaling* parameter in Xcos via the *setup* submenu of the *simulation* tab has to be set to 1. The *final integration time* parameter sets the total simulation time in s. The solver is set to the default *Sundial/CVODE - BDF - Newton*. Within the menu of the DDC fixed set-point controller software, the reset time (ger. Nachstellzeit) T_i in minutes, the proportional gain (ger. Proportionalbeiwert) K_P and set-point temperature w (ger. Sollwert) may be assigned to set values. The boiler's temperature initial value $boilerInit$ is set to 338 K within the Xcos boiler model, as described in chapter 3.5.4. The *cycling time* (ger. Zykluszeit) in the DDC's controller software is set to 1 s, as is the sampling time t_s in the Xcos heating system model. The following simulation results were selected from more than 13 hours of simulation time, using varying parameter values.
Figure 6.1 shows four example simulation runs of five minutes each, plotting the curve progressions of the control variable T_s converging towards the set-point w in K. All parameters are set to identical values.



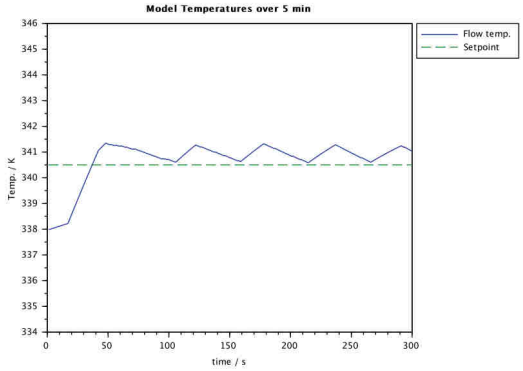
(a) 1. simulation run



(b) 2. simulation run



(c) 3. simulation run

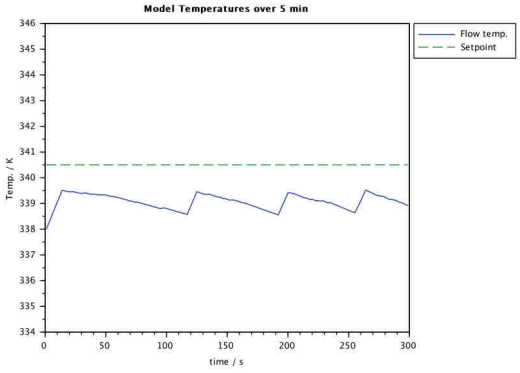


(d) 4. simulation run

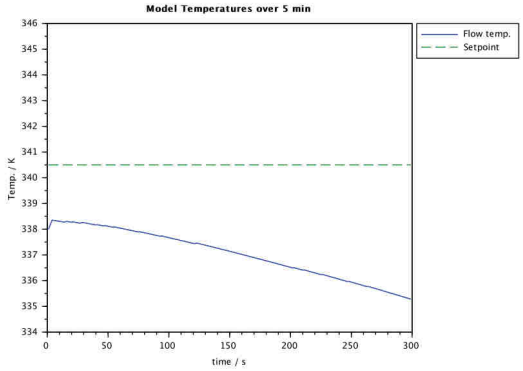
Figure 6.1: Four simulation results for the curve progression of control variable T_s converging towards the set-point $w = 340.5\ K$, initial boiler temperature $boilerInit = 338\ K$, final integration time = 300 s. PI-controller: $K_P = 0.5$, T_i=60 s, cycling time = sampling time = 1 s (Source: Scilab plot)

An assumption for the deviating curve progression in these results is a lack of synchronous behaviour between the Xcos model and the DDC controller affecting data transmission. In addition, the reason might be related to the USB to SERIAL bridge. Compared to a standard RS232 serial port, USB uses a cycle based communication concept with possible delays in the serial communication [25].

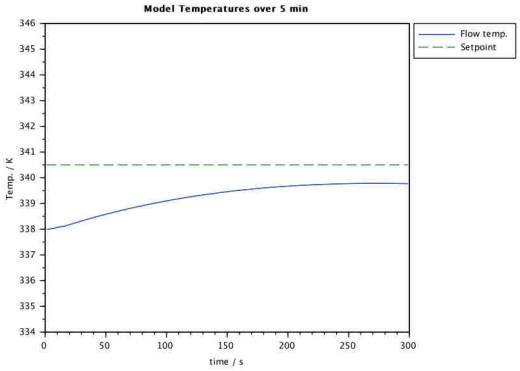
Figure 6.2 shows four additional simulation results with modified parameters to demonstrate further the behaviour of the DDC's fixed set-point controller. Again the curve progressions are plotted of the control variable T_s converging towards the set-point w.



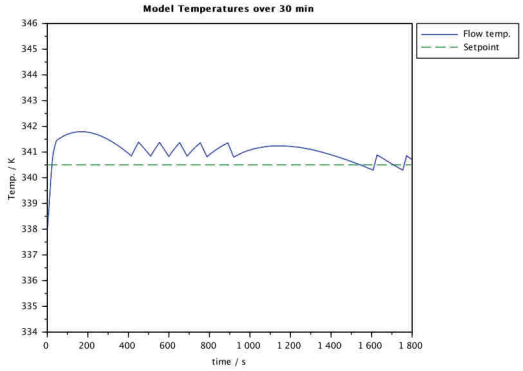
(a) Simulation over 5 min., $K_P = 0.5$, $T_i = 120\ s$, $w = 340.5\ K$, $boilerInit = 338\ K$.



(b) Simulation over 5 min., $K_P = 1$, $T_i = 60\ s$, $w = 340.5\ K$, $boilerInit = 338\ K$.



(c) Simulation over 5 min., $K_P = 65$, $T_i = 60\ s$, $w = 340.5\ K$, $boilerInit = 338\ K$.



(d) Simulation over 30 min., $K_P = 2$, $T_i = 0$,$w = 340.5\ K$, $boilerInit = 338\ K$.

Figure 6.2: Four simulation results for the curve progression of the control variable T_s converging towards the set-point $w = 340.5\ K$, initial boiler temperature $boilerInit = 338\ K$, with varying integration times, cycling time $= 1\ s$ (Source: Scilab plot)

The simulation result of figure 6.2 (a) shows a similar curve progression as figure 6.1 (a-d), for a doubling of the reset time T_i induces an overall decrease of the output signal, as expected from equation 3.3.

In figure 6.2 (b) the progression curve of T_s drops off, away from the set-point w. As the proportional gain K_P has been doubled, compared to figure 6.1, this progression is contrary to the expected result.

The curve progression in figure 6.2 (c) seems to converge promisingly towards the set-point line, however it does not coincide with the previous results of a lower K_P value.

Finally, figure 6.2 (d) shows the progression of T_s with a similar result as in figure 6.1. The controller is set to a P-controller, albeit with a substantial greater K_P as in 6.1 and a nonuniform behaviour.

The simulation results of figure 6.2 demonstrate that further simulation and testing is needed, to develop a better understanding of the data transfer in the HIL. At this stage, a degree of randomness in the results may be observed, due to a possible bug or error on the signal route. Further testing of the I/O interface, espcially the serial connection is needed. After a number of simulation runs, occasionally the serial communication is disrupted. Chapter 7 describes a workaround solution for when such an error occurs.

7 Troubleshooting

The following solutions for errors were identified, whilst developing the HIL simulation components and during debugging and simulating.

UDOO

After cycling around 10 simulation runs, it may occur that the serial communication breaks off, or at least does not transfer feasible data (NaN-Error). The causes may be subject of further investigations. The following workaround solution is helpful. Install and setup the serial communication application Minicom as described in appendix 9.1.3. Open Minicom using *sudo minicom -w* as root in an Ubuntu terminal. Make sure the jumper on the UDOO is on J18. After a UDOO reset stop the boot-up process by hitting `any-key`.

- Close the serial monitor. Unplug J18 jumper to allow the communication with the programming port of SAM3x.
- Plug the jumper J22 for 1 second, then remove it (to erase the old sketch programmed in SAM3x).
- Plug the jumper J16 for 1 second, then remove it (to reset the SAM3x).
- Upload the Arduino sketch using the Arduino Due programming port button.
- Press the reset button to restart i.MX6.

Arduino IDE

- When uploading the Arduino sketch, occasionally a "Port not found"-error may occur. Try unplugging the USB cable from the laptop and reinserting it. Using the *dmesg* command in an Ubuntu terminal shows the port address of the connecting device.

Scilab/XCOS

- For the Xcos function blocks it is possible to define variables for the "Variable Browser" similar to the "Workspace" in Matlab®. However it is recommended to hard-code the variables into the function blocks of Xcos, if a math operation is

involved, else the Scilab compiler computes senseless data. However, the "Variable Browser" may be used to define single value variables, which may be used on all levels of the Xcos block hierarchy. Figure 7.1 shows an example of a *const_m* function block for a product.

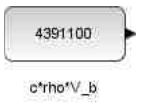


Figure 7.1: A constant Xcos function block with a product as single value

- Use *SampleCLK* function block instead of CLOCK_c function block, as all *SampleCLK* blocks are synchronous. In each of the *SampleCLK* function blocks of the heating system models set an offset of 1 *s* for stable simulation runs.
- The minimum sampling time for the *SampleCLK* function blocks is 0.015 *s* for the serial read/write functions to transfer data across the serial connection in a stable way.
- Occasionally an error occurs while installing the Atoms Module Manager : "*No ATOMS module is available. Please, check your Internet connection or make sure that your OS is compatible with ATOMS.*". This might mean that the system language is not set to "English".
  - Open a terminal in Ubuntu logged on as root-user.
  - Check for the language setting with *locale* (it should be *en_US.UTF-8*). Else, enter the following commands:
  - *sudo locale-gen en_US.UTF-8*
  - *sudo update-locale LANG=en_US.UTF-8*
  - Restart the terminal and / or Ubuntu.

8 Conclusion & Outlook

Within the scope of this thesis a hardware-in-the-loop (HIL) simulation was developed, in which the physical behaviour of a heating system for a non-residential building was reproduced by a software model. The project was regarded in context with the Observe research project. The result was a HIL simulation of a software modelled heating system, controlled by an external PI-controller in a feedback loop. Each physical hardware component for the real-time simulation was described, specifically the functionality of the UDOO single board computer and the DDC4200 automation station including the PI-controller. The UDOO was implemented as an I/O-board, connected to a laptop computer via an USB-serial interface. The software model simulation of the heating system was outsourced onto the laptop. This setup proved to be advantageous in reducing the latency, compared to running the Xcos simulation on the UDOO.
The PI-controller, contained within the DDC4200, forwarded an actuation signal to the programmed heating system model, which returned the control variable via the UDOO I/O-functionality. The physical theory of the heating system was outlined and the existing continuous-time model of a heating system discretized using the EULER forward method. The heating model was thus concentrated to its core physical functionality. A set of difference equations was derived for the state variable of each heating system component. The conversion from a continuous-time to a discrete-time model made the model adaptable to advanced control systems like MPC. The numerical, open source programming language Scilab/Xcos was introduced to realize the discrete-time models. The operating system implemented was a 32-bit Ubuntu OS version for the purpose of creating a cost-free software development platform as an advantage compared to the pricey licenses of Matlab/Simulink® and MS Windows®. A step-by-step instruction on installing Scilab from the source file in Ubuntu OS was included.
In addition, the serial functionality of Scilab was presented for the transfer of data from the heating system model across the serial-USB bridge to the UDOO. The basic structure of an open source Arduinio IDE sketch was specified, detailing the management of data transfer between the serial line and the I/O pins of the UDOO to forward the signal to the DDC4200 automation station. The electrical interface needed for this purpose was specified, including the calculations for the additional circuitry for converting the voltage signals in compliance with the input pins of either the UDOO or the DDC4200.
A troubleshooting guide recorded some of the practical debugging errors and solutions encountered during setup of the HIL.
A set of simulations was carried out of five to thirty minutes each, with set parameter

values, for testing the HIL simulation controllability. The plotted result, shows the curve progressions of the control variable converging towards a set-point temperature.

At first, the simulations compared curve progressions with identically set control parameters. A disparity of the curve characteristic of each result was observed. This deviation between the results was explained due to potential asynchronous data transfer between the Scilab/Xcos model and the controller of the DDC4200. A second set of simulations compared curve progressions with varying controller parameters. The results partly deviate from the expected controller theory with further simulations necessary to explore the causes.

In fact, the simulation results demonstrated the operational readiness of the HIL, proving the controllability of the Xcos heating system model by an external fixed set-point controller. The limitation was the lack of comparability of the results with another controller. The data transfer lacked perfect stability and further simulation tests are needed to garner a better understanding for both the data transfer within the HIL and the controller.

In conclusion, a low-cost, technical platform was created as a basis for future projects. In a next step the DDC's PI-controller may be replaced with a Xcos function block PI-controller to compare the results. The modular nature of the HIL enables the developer to implement other software or hardware components. For instance, more sophisticated controllers may be implemented on a separate UDOO, attached to the existing platform, for advanced controlling like model predictive control. Further developing the existing HIL simulation to include advanced controllers, could be a future approach to develop more energy efficient heating systems.

9 Appendix

9.1 Software setup

9.1.1 Ubuntu: removing and updating package repositories

Using the Ubuntu APT (advanced packaging tool) *apt-get* [9] for removing and updating package repositories from Ubuntu, the following sequence of commands is useful [26] :

sudo apt-get install -f // fixes broken dependencies if there are any
sudo apt-get clean // deletes download installation files from the cache
sudo apt-get autoclean // deletes all downloaded package files that aren't existent in the sources file.
sudo apt-get update // all package-sources from the /etc/apt/sources.list are read again checking the signature
sudo apt-get upgrade // updates the already installed packages from the package sources
sudo apt-get dist-upgrade // updates the already installed packages from the package sources and installs new packages if required

Possibly the APT package repositories sources in the */etc/apt/sources.list* file needs to be edited (using *vi* or *nano*).

9.1.2 Compiling & configuring Scilab

To be able to compile Scilab 5.X some core dependencies for your linux distribution are mandatory, see
http://wiki.scilab.org/Dependencies%20of%20Scilab%205.X for further reading.

Open a terminal in Ubuntu and enter following commands as root-user:
sudo apt-get update // refreshes packages source information in the /etc/apt/sources.list
sudo apt-get upgrade // upgrades package from package source to latest release
sudo apt-get dist-upgrade // same as "upgrade", also installs new packages replacing old ones which are obsolete through new dependencies

Create a new folder:
mkdir /usr/local/src // makes new directory

sudo chmod -R 777 /usr/local/src // change user rights to read/write/execute for created directory
sudo chown -cR $USER /usr/local/src // change owner of directory to root

Install the following required packages:
sudo apt-get install libcurl3 // installs the libcurl3 package
sudo apt-get install libcurl3-dev // installs the libcurl3-dev package

Download the "libmatio" package from:
http://sourceforge.net/projects/matio/matio-1.5.2.tar.gz
Extract the files using the *tar* command to usr/local/src:
tar -xvzf filename.tar.gz
According to the README file install the package in /usr/local .

Download the "libarpack" package from:
http://forge.scilab.org/index.php/p/arpack-ng/arpack-ng_3.1.5.tar.gz
Extract the files using the *tar* command to usr/local/src .
According to the README file install the package in /usr/local .

Download the "libxml2" package from:
ftp://xmlsoft.org/libxml2/libxml2-sources-2.9.2.tar.gz
Extract the files using the *tar* command to /usr/local/libxml2.
According to the README file install the package by typing the following commands into the terminal:
./configure –prefix=/usr/local/libxml2
make
sudo make install

Next download the source files for Scilab 5.5.1 and the prerequiremtents from:
http://www.scilab.org/development/sources/stable

In a terminal switch to the folder: /usr/local/src:
Extract the compressed files of the prerequirements and Scilab using the *tar* command.

All required dependencies for Scilab 5.5.1 should be installed, switch to folder:
/usr/local/src/scilab-5.5.1 and enter the following commands:
./configure –prefix=/usr/local –with-libxml2=/usr/local/libxml2 // builds the makefile
make all // builds the program
sudo make install //install program set in the makefile
DONE!

To run type ./bin/scilab

To make the script universally available in the terminal. Using the example of Scilab:

echo $SHELL // check which shell you're running

for BASH SHELL enter:
echo 'export PATH=$PATH:/usr/local/src/scilab-5.5.1/bin' » /.bashrc // this adds the path of Scilab to .bashrc config-file
Now type "scilab" from anywhere in a terminal to run the program.

9.1.3 Connecting to the USB serial interface

To install Minicom, text-base modem control for serial communication from the Linux console, log on as root and do the following steps:

To install minicom:
sudo apt-get update
sudo apt-get install minicom

To configure use the following command:
sudo minicom -sw

Go to *Serial port setup* and edit as follow:
Serial Device: /dev/ttyUSB0
Hardware Flow Control: No
Software Flow Control: No

Press *exit* and *Save setup as dfl* and exit from Minicom.

Change access permissions to the serial port with:
sudo chmod 666 /dev/ttyUSB0

To start listening on the serial line use:

sudo minicom -w

Restart the UDOO and press-any-key to connect to the serial console shell.

9.2 Software code

9.2.1 Arduino Sketch

```
//declare integer variables
int inNum = 0;
int outPin = DAC0;
int inPin = A0;
int val = 0;
// setup serial connections
void setup() // setup function
{
Serial.begin(9600); // set baud rate to 9600 Bd
delay(2000); // wait 2000ms
analogReadResolution(10); // set ADC to default 10 bit returns values betwenn 0-1023
analogWriteResolution(10); // change resolution of DAC from default 8 bit to 10 bit
}

//main method of sketch
void loop()
{
val = analogRead(inPin); // reads value form ADC, here converts 0.002 V - 2.543 V to
0.5 - 758
Serial.println(val); // prints the value to the serial interface
delay(20); //waits 20 ms
if (Serial.available() > 0) { // if values in serial buffer
inNum = Serial.parseInt(); // read first integer from serial buffer
analogWrite(outPin, inNum); // maps integer value to 0-1023 and writes to pin
delay(20); //wait 20 ms
}else{
delay(20);//waits 20 ms
}
}
```

9.2.2 Code in the XCOS serial communication block

The following code is embedded in the *scifunc_block_m* Xcos function block within the *read / write* super-block.

str=string(round(u1))+ascii(44); // round the float, create a string and add a ,
writeserial(h,str); // write the string one char at a time to the serial port
data_temp=strsubst(readserial(h),ascii(10),','); // read char from serial port to form string replacing the ASCII control character 'linefeed' with a ','
data_temp2=strtod(strsplit(data_temp,',')); // split string add ',' as a denominator; create an integer from the string
y1=data_temp2(1); // only use first row in the vector

9.2.3 Scilab serial communication script

The following code is a Scilab script to open the serial connection on set port number and to simulate the XCOS diagram.

clc; //clear console
getd('/home/patrick/Desktop/Scilab/2015/xcos'); // get access on the directory specified
h = openserial_udoo(0,"9600,n,8,1"); // open serial port 0, 9600 baud rate, n parity bits, 8 data bits, 1 stop bit
xpause(1000000); //pause 1 sec
importXcosDiagram('/home/patrick/Desktop/Scilab/2015/xcos/building/Streckensimulator_mar
// import the xcos code to this script
xcos_simulate(scs_m,4); // run import xcos script
// create a vector setpointf of size tkessel_out.values for set-point value
for i=1:length(tkessel_out.values)
setpointf(i)=340.5;
end
x=1:length(setpointf); // create vector the same size as setpointf
plot(x,tkessel_out.values','-',x, setpointf','–'); // plot simulated boiler temp. values
a=gca(); //get am axis handle
a.data_bounds=[0,334;300,346]; //set range of axis
hl=legend(['Flow temp.';'Setpoint'],[-1]); // set the legend

title('Model Temperatures over '+string(300/60)+ ' min','fontname',4); // set the title with the number of simulated minutes
xlabel('time / s'); // label the x-axis
ylabel('Temp. / K'); // label the y-axis
closeserial(h); // close the serial port

9.2.4 Scilab adapted openserial function

The following function is a customization of the openserial function from the Serial Communication Toolbox in Scilab. The amended line is bold-faced.

```
h=openserial_udoo(p,smode,translation,handshake,xchar,timeout)
//port name if exists("p","local") then p=1; end
if type(p)==1 | type(p)==8 then
// if p<=0 then error("port number must be greater than zero"); end
if getos() == "Windows" then
port="COM"+string(p)+":"
else
port="/dev/ttyUSB"+string(p); // assign the serial port to the USB-port end
elseif type(p)==10
port=p
else
error("port to open must be either a number or a string")
end
TCL_EvalStr("set porthandle [open "+port+" r+]");
h=TCL_GetVar("porthandle");
// parsing communication modes: via fconfigure: -translation
// -mode,-handshake, -xchar, -timeout, -blocking
// translation=["auto","binary","cr","crlf","lf"]
// handshake=["none","rtscts","xonxoff","dtrdsr"]
// xchar=[,]
if exists("smode","local") then
TCL_EvalStr("fconfigure "+h+" -mode "+smode)
end
//default translation is binary to avoid input character count skew
if exists("translation","local") then
translation="binary"
end
TCL_EvalStr("fconfigure "+h+" -translation "+translation)
//does nonblocking work the way I'd expect?
if exists("blocking","local") then
blocking=%f
```

```
end
TCL_EvalStr("fconfigure "+h+" -blocking "+string(bool2s(blocking)))
if exists("handshake","local") then
TCL_EvalStr("fconfigure "+h+" -handshake "+handshake)
end
if exists("xchar","local") then
TCL_EvalStr("fconfigure "+h+" -xchar "+string(xchar)+"")
end
if exists("timeout","local") then
TCL_EvalStr("fconfigure "+h+" -timeout "+string(timeout)+"")
end
endfunction
```

9.3 Datasheets

9.3.1 UDOO specifications

2-page extract from the UDOO starting manual (beta V.04):

1. Introduction

1.1. What's UDOO?

UDOO is a mini PC that can be used both with Android and Linux OS, with an embedded Arduino-compatible board. It is a powerful prototyping board for software development and design; it' s easy to use and allows developing projects with minimum knowledge of hardware. UDOO merges different computing worlds together: each one has its proper strenght- and weak- points, but all of them are useful in today's life for educational purposes as well as Do-It-Yourself (DIY) and quick prototyping. UDOO is an open hardware, low-cost platform equipped with an ARM i.MX6 Freescale processor, and an Arduino Due compatible section based on ATMEL SAM3X ARM processor, all this available on the same board!

1.2. UDOO goals

- Develop an innovative product for a growing market
- Give a new vision to the educational framework, with the idea of training up a new generation of engineers, designers and software developers skilled in digital technology: physical computing, multi-media arts, interactive arts, IoT...
- Give a boost to the DIY world
- Offer a low cost embedded platform for interactive arts with powerful tools: Processing, OpenCV, PureData, openFramework
- Provide companies with a great tool for fast prototyping

1.3. Specifications

UDOO retail line up consists of three models, sharing most of the featuresand different only for connectivity and i.MX6 processor used. All three models feature an embedded Arduino compatible section based on Arduino Due schematic. UDOO' s size are 4.33 inch x 3.35 inch (11 cm x 8.5 cm).
Warning: The UDOO I/O pins are 3.3V compliant. Higher voltages (like 5V) would damage the board.

1.3.1. GPIO features

Current version, UDOO rev. D, has these additional features:

- S/PDIF digital audio in & out through pin headers;
- I2S/AC97/SSI digital audio multiplexer through pin headers;
- FlexCAN (Flexible Controller Area Network) through pin headers, it is possible to switch this function' s management between i.MX6 processor and SAM3X8E processor;
- External SD card support through pins header: plug an external controller for an additional SD card slot or for an eMMC module.

1.3.2. UDOO Quad

- Freescale i.MX6Quad, 4 x ARM® Cortex™-A9 core @ 1GHz with ARMv7A instruction set

- GPU Vivante GC 2000 for 3D + Vivante GC 355 for 2D (vector graphics) + Vivante GC 320 for 2D (composition)
- Atmel SAM3X8E ARM Cortex-M3 CPU (same as Arduino Due)
- RAM DDR3 1GB
- 76 fully available GPIO with Arduino-compatible R3 1.0 pinout
- HDMI and LVDS + Touch
- 2 Micro USB (1 OTG)
- 2 USB 2.0 type A and 1 USB 2.0 internal pin header (requires adapter cable)
- Analog Audio and Mic jacks
- CSI Camera Connection
- on board Micro SD card reader (boot device)
- Power Supply (6-15V) and External Battery connector
- Ethernet RJ45 (10/100/1000 MBit)
- WiFi Module
- SATA connector with power header

1.3.3. UDOO Dual

- Freescale i.MX6DualLite, 2x ARM® Cortex™-A9 core @ 1GHz with ARMv7A instruction set
- GPU Vivante GC 880 for 3D and 2D (vector graphics) + Vivante GC 320 for 2D (composition)
- Atmel SAM3X8E ARM Cortex-M3 CPU (same as Arduino Due)
- RAM DDR3 1GB
- 76 fully available GPIO with Arduino-compatible R3 1.0 pinout
- HDMI and LVDS + Touch
- 2 Micro USB (1 OTG)
- 2 USB 2.0 type A and 1 USB 2.0 internal pin header (requires adapter cable)
- Analog Audio and Mic jacks
- CSI Camera Connection
- on board Micro SD card reader (boot device)
- Power Supply (6-15V) and External Battery connector
- Ethernet RJ45 (10/100/1000 MBit)
- WiFi Module
- SATA connector with power header

1.3.4. UDOO Dual Basic

- Freescale i.MX6DualLite, 2x ARM® Cortex™-A9 core @ 1GHz with ARMv7A instruction set
- GPU Vivante GC 880 for 3D and 2D (vector graphics) + Vivante GC 320 for 2D (composition)
- Atmel SAM3X8E ARM Cortex-M3 CPU (same as Arduino Due)
- RAM DDR3 1GB
- 76 fully available GPIO with Arduino-compatible R3 1.0 pinout
- HDMI and LVDS + Touch

9.3.2 Atmel Sam3x

Table 41-40. Dynamic Performance Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
SNR	Signal to Noise Ratio			71		dB
THD	Total Harmonic Distortion			-71		dB
SINAD	Signal to Noise and Distortion			68		dB

Note: DAC Clock (f_{DAC}) = 50 MHz, f_s = 2 MHz, f_{IN} = 127 kHz, IBCTL = 01, FFT using 1024 points or more, Frequency band = [10 kHz, 500 kHz] – Nyquist conditions fulfilled.

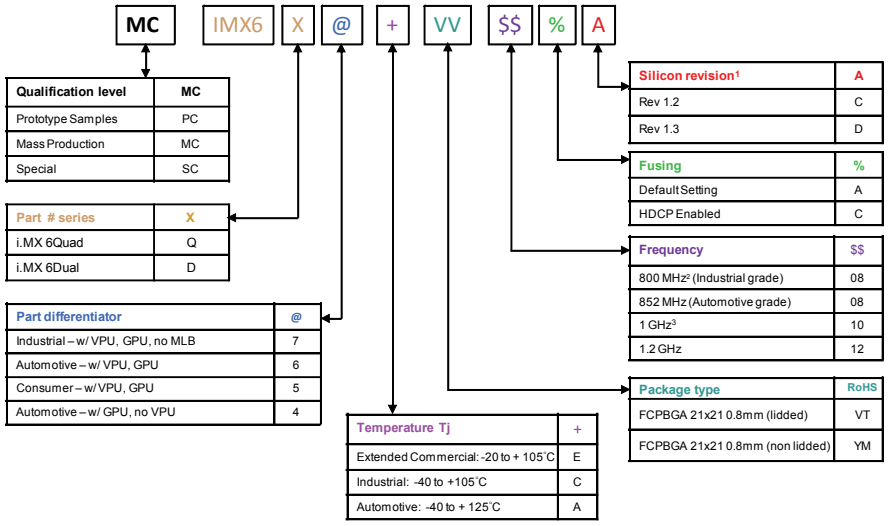
Table 41-41. Analog Outputs

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{OR}	Voltage Range		(1/6) V_{ADVREF}		(5/6) V_{ADVREF}	V
SR	Slew Rate	Channel Output Current vs Slew Rate (IBCTL for DAC0 or DAC1, noted IBCTLCHx) R_{LOAD} = 5kΩ, 0pF < C_{LOAD}< 50 pF IBCTLCHx = 00 IBCTLCHx = 01 IBCTLCHx = 10 IBCTLCHx = 11		 2.7 5.3 8 11		V/µs
	Output Channel Current Consumption	No resistive load IBCTLCHx = 00 IBCTLCHx = 01 IBCTLCHx = 10 IBCTLCHx = 11		 0.23 0.45 0.78 0.9		mA
t_{sa}	Settling Time	R_{LOAD} = 5kΩ, 0pF < C_{LOAD}< 50pF			0.5	µs
R_{LOAD}	Output Load Resistor		5			kΩ
C_{LOAD}	Output Load Capacitor				50	pF

9.3.3 Freescale i.MX6 ARM Cortex-A9 quad core

Introduction

Ensure that you have the right data sheet for your specific part by checking the temperature grade (junction) field and matching it to the right data sheet. If you have questions, see freescale.com/imx6series or contact your Freescale representative.



1. See the freescale.com\imx6series Web page for latest information on the available silicon revision.
2. If a 24 MHz input clock is used (required for USB), the maximum SoC speed is limited to 792 MHz.
3. If a 24 MHz input clock is used (required for USB), the maximum SoC speed is limited to 996 MHz.

Figure 1. Part Number Nomenclature—i.MX 6Quad and i.MX 6Dual

1.2 Features

The i.MX 6Dual/6Quad processors are based on ARM Cortex-A9 MPCore platform, which has the following features:

- ARM Cortex-A9 MPCore 4xCPU processor (with TrustZone®)
- The core configuration is symmetric, where each core includes:
  - 32 KByte L1 Instruction Cache
  - 32 KByte L1 Data Cache
  - Private Timer and Watchdog
  - Cortex-A9 NEON MPE (Media Processing Engine) Co-processor

The ARM Cortex-A9 MPCore complex includes:

- General Interrupt Controller (GIC) with 128 interrupt support
- Global Timer
- Snoop Control Unit (SCU)

- 1 MB unified I/D L2 cache, shared by two/four cores
- Two Master AXI (64-bit) bus interfaces output of L2 cache
- Frequency of the core (including Neon and L1 cache) as per Table 6
- NEON MPE coprocessor
  - SIMD Media Processing Architecture
  - NEON register file with 32x64-bit general-purpose registers
  - NEON Integer execute pipeline (ALU, Shift, MAC)
  - NEON dual, single-precision floating point execute pipeline (FADD, FMUL)
  - NEON load/store and permute pipeline

The SoC-level memory system consists of the following additional components:

  - Boot ROM, including HAB (96 KB)
  - Internal multimedia / shared, fast access RAM (OCRAM, 256 KB)
  - Secure/non-secure RAM (16 KB)
- External memory interfaces:
  - 16-bit, 32-bit, and 64-bit DDR3-1066, LVDDR3-1066, and 1/2 LPDDR2-1066 channels, supporting DDR interleaving mode, for 2x32 LPDDR2-1066
  - 8-bit NAND-Flash, including support for Raw MLC/SLC, 2 KB, 4 KB, and 8 KB page size, BA-NAND, PBA-NAND, LBA-NAND, OneNAND™ and others. BCH ECC up to 40 bit.
  - 16/32-bit NOR Flash. All EIMv2 pins are muxed on other interfaces.
  - 16/32-bit PSRAM, Cellular RAM

Each i.MX 6Dual/6Quad processor enables the following interfaces to external devices (some of them are muxed and not available simultaneously):

- Hard Disk Drives—SATA II, 3.0 Gbps
- Displays—Total five interfaces available. Total raw pixel rate of all interfaces is up to 450 Mpixels/sec, 24 bpp. Up to four interfaces may be active in parallel.
  - One Parallel 24-bit display port, up to 225 Mpixels/sec (for example, WUXGA at 60 Hz or dual HD1080 and WXGA at 60 Hz)
  - LVDS serial ports—One port up to 165 Mpixels/sec or two ports up to 85 MP/sec (for example, WUXGA at 60 Hz) each
  - HDMI 1.4 port
  - MIPI/DSI, two lanes at 1 Gbps
- Camera sensors:
  - Parallel Camera port (up to 20 bit and up to 240 MHz peak)
  - MIPI CSI-2 serial camera port, supporting up to 1000 Mbps/lane in 1/2/3-lane mode and up to 800 Mbps/lane in 4-lane mode. The CSI-2 Receiver core can manage one clock lane and up to four data lanes. Each i.MX 6Dual/6Quad processor has four lanes.
- Expansion cards:
  - Four MMC/SD/SDIO card ports all supporting:

  - 1-bit or 4-bit transfer mode specifications for SD and SDIO cards up to UHS-I SDR-104 mode (104 MB/s max)
  - 1-bit, 4-bit, or 8-bit transfer mode specifications for MMC cards up to 52 MHz in both SDR and DDR modes (104 MB/s max)
- USB:
  - One High Speed (HS) USB 2.0 OTG (Up to 480 Mbps), with integrated HS USB PHY
  - Three USB 2.0 (480 Mbps) hosts:
    - One HS host with integrated High Speed PHY
    - Two HS hosts with integrated HS-IC USB (High Speed Inter-Chip USB) PHY
- Expansion PCI Express port (PCIe) v2.0 one lane
  - PCI Express (Gen 2.0) dual mode complex, supporting Root complex operations and Endpoint operations. Uses x1 PHY configuration.
- Miscellaneous IPs and interfaces:
  - SSI block capable of supporting audio sample frequencies up to 192 kHz stereo inputs and outputs with I^2S mode
  - ESAI is capable of supporting audio sample frequencies up to 260kHz in I2S mode with 7.1 multi channel outputs
  - Five UARTs, up to 4.0 Mbps each:
    - Providing RS232 interface
    - Supporting 9-bit RS485 multidrop mode
    - One of the five UARTs (UART1) supports 8-wire while others four supports 4-wire. This is due to the SoC IOMUX limitation, since all UART IPs are identical.
  - Five eCSPI (Enhanced CSPI)
  - Three I2C, supporting 400 kbps
  - Gigabit Ethernet Controller (IEEE1588 compliant), 10/100/1000[1] Mbps
  - Four Pulse Width Modulators (PWM)
  - System JTAG Controller (SJC)
  - GPIO with interrupt capabilities
  - 8x8 Key Pad Port (KPP)
  - Sony Philips Digital Interconnect Format (SPDIF), Rx and Tx
  - Two Controller Area Network (FlexCAN), 1 Mbps each
  - Two Watchdog timers (WDOG)
  - Audio MUX (AUDMUX)
  - MLB (MediaLB) provides interface to MOST Networks (150 Mbps) with the option of DTCP cipher accelerator

1. The theoretical maximum performance of 1 Gbps ENET is limited to 470 Mbps (total for Tx and Rx) due to internal bus throughput limitations. The actual measured performance in optimized environment is up to 400 Mbps. For details, see the ERR004512 erratum in the i.MX 6Dual/6Quad errata document (IMX6DQCE).

The i.MX 6Dual/6Quad processors integrate advanced power management unit and controllers:

- Provide PMU, including LDO supplies, for on-chip resources
- Use Temperature Sensor for monitoring the die temperature
- Support DVFS techniques for low power modes
- Use Software State Retention and Power Gating for ARM and MPE
- Support various levels of system power modes
- Use flexible clock gating control scheme

The i.MX 6Dual/6Quad processors use dedicated hardware accelerators to meet the targeted multimedia performance. The use of hardware accelerators is a key factor in obtaining high performance at low power consumption numbers, while having the CPU core relatively free for performing other tasks.

The i.MX 6Dual/6Quad processors incorporate the following hardware accelerators:

- VPU—Video Processing Unit
- IPUv3H—Image Processing Unit version 3H (2 IPUs)
- GPU3Dv4—3D Graphics Processing Unit (OpenGL ES 2.0) version 4
- GPU2Dv2—2D Graphics Processing Unit (BitBlt)
- GPUVG—OpenVG 1.1 Graphics Processing Unit
- ASRC—Asynchronous Sample Rate Converter

Security functions are enabled and accelerated by the following hardware:

- ARM TrustZone including the TZ architecture (separation of interrupts, memory mapping, etc.)
- SJC—System JTAG Controller. Protecting JTAG from debug port attacks by regulating or blocking the access to the system debug features.
- CAAM—Cryptographic Acceleration and Assurance Module, containing 16 KB secure RAM and True and Pseudo Random Number Generator (NIST certified)
- SNVS—Secure Non-Volatile Storage, including Secure Real Time Clock
- CSU—Central Security Unit. Enhancement for the IC Identification Module (IIM). Will be configured during boot and by eFUSEs and will determine the security level operation mode as well as the TZ policy.
- A-HAB—Advanced High Assurance Boot—HABv4 with the new embedded enhancements: SHA-256, 2048-bit RSA key, version control mechanism, warm boot, CSU, and TZ initialization.

1.3 Updated Signal Naming Convention

The signal names of the i.MX6 series of products have been standardized to better align the signal names within the family and across the documentation. Some of the benefits of these changes are as follows:

- The names are unique within the scope of an SoC and within the series of products
- Searches will return all occurrences of the named signal
- The names are consistent between i.MX 6 series products implementing the same modules
- The module instance is incorporated into the signal name

9.3.4 DDC 4200

Technical data sheet 2.60-10.200-01-en

DDC4200 DDC Central Unit **Device Description**

Technical data

Bus connection	Ethernet	99 DDC4000 Central Units can be administrated, networkable worldwide on active network components, 10/100 Mbit/s
	2 CAN buses switchable as Field or Control Cabinet Bus	Field Bus; F Bus: 63 Field Bus Module FBM, 2000m; 20kBaud, CAN, J-Y(St) Y 2x2x0.8mm² Control Cabinet Bus; SBM Bus: 16 Control Cabinet Bus Modules SBM or BMA/D; 200m; 40kBaud, CAN
	Field and Control Cabinet Bus	At the farthest removed point from the central unit a terminator resistance of 180 Ohms must be attached between "BUS+" and "BUS-".
Interfaces	Serial RS232	Modem, printer
	RS485	decoupling for J Y(St)-Y connection
	CompactFlash	for CompactFlash card; update, data backup / file recovery (behind the front panel)
Inputs and outputs	32 digital inputs DI / digital outputs DO switchable	Transistor output potential-free contact against 0V=24V DC; 50mA, 8 BI of these for pulse count to 80Hz
	24 analog inputs AI / analog outputs AO switchable	Sensor type — Value range and unit 0..10V — 0 to 100% KP10 — -50 to +150°C Pt100 — -50 to +850°C Pt1000 — -50 to +150°C Ni100 — -50 to +150°C Ni1000 (DIN) — -50 to +150°C Ni1000 (L&G) — -50 to +150°C KP250 — -50 to +150°C
	Output	10V/50mA
Operating voltage	for DDC central unit	24V AC +/-10%; 50..60Hz; 33VA; 1.4A or 24V DC +/-10%; 14.4W; 0.6A or 12V DC +/-10%; 15.6W; 1.3A
	for digital inputs and outputs	24V DC +/-10% / 50mA
Fuses	Mains fuse, T 3.15A	
Display	TouchScreen with active back-lighted ¼ VGA color TFT display 14 cm diagonal (5.7 inches)	
Memory	128 MByte Flash Disc; 48MByteSDRAM; 1 MByte Flash-PROM (boot)	
Operating system	Embedded Linux	
Power outage data backup	7 years, clock components battery-buffered	
Degree of enclosure protection	IP30	
Ambient temperature	0..45°C	
Environmental humidity	In operation: 20..80% r. h., not condensing Out of operation: 5..90% r. h., not condensing	
Housing	19" short cassette out of plastic, 4-fold cassette with a base and separated connections for Ethernet and RS232 B x H x T; 202mm x 132mm x 137mm	
Front panel cutout	200.4mm x 112.0mm	
Weight	1.225kg	
Designation	CE	

kieback&peter
Technology for building automation

9.3.5 Operational amplifier

Internally-compensated dual low noise operational amplifier

NE/SA/SE5532/5532A

DESCRIPTION

The 5532 is a dual high-performance low noise operational amplifier. Compared to most of the standard operational amplifiers, such as the 1458, it shows better noise performance, improved output drive capability and considerably higher small-signal and power bandwidths.

This makes the device especially suitable for application in high-quality and professional audio equipment, instrumentation and control circuits, and telephone channel amplifiers. The op amp is internally compensated for gains equal to one. If very low noise is of prime importance, it is recommended that the 5532A version be used because it has guaranteed noise voltage specifications.

FEATURES

- Small-signal bandwidth: 10MHz
- Output drive capability: 600Ω, 10V_{RMS}
- Input noise voltage: 5nV/ $\sqrt{Hz}$ (typical)
- DC voltage gain: 50000
- AC voltage gain: 2200 at 10kHz
- Power bandwidth: 140kHz
- Slew rate: 9V/ s
- Large supply voltage range: ±3 to ±20V
- Compensated for unity gain

PIN CONFIGURATIONS

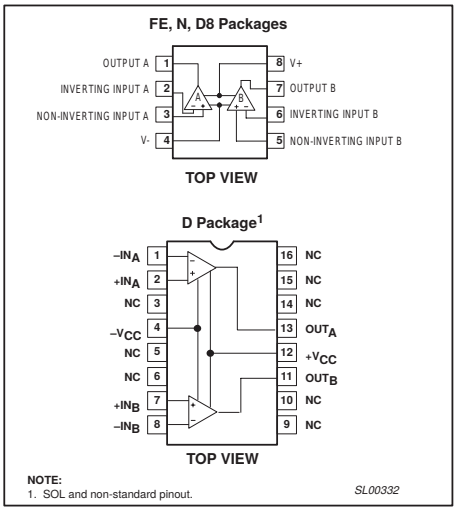


Figure 1. Pin Configurations

ORDERING INFORMATION

DESCRIPTION	TEMPERATURE RANGE	ORDER CODE	DWG #
8-Pin Plastic Dual In-Line Package (DIP)	0 to 70°C	NE5532N	SOT97-1
8-Pin Plastic Dual In-Line Package (DIP)	–40°C to +85°C	SA5532N	SOT97-1
8-Pin Plastic Dual In-Line Package (DIP)	–40°C to +85°C	SA5532AN	SOT97-1
8-Pin Ceramic Dual In-Line Package (CERDIP)	0 to 70°C	NE5532FE	0580A
8-Pin Plastic Dual In-Line Package (DIP)	0 to 70°C	NE5532AN	SOT97-1
8-Pin Ceramic Dual In-Line Package (CERDIP)	0 to 70°C	NE5532AF	0580A
8-Pin Ceramic Dual In-Line Package (CERDIP)	-55°C to +125°C	SE5532FE	0580A
8-Pin Ceramic Dual In-Line Package (CERDIP)	-55°C to +125°C	SE5532AF	0580A
8-Pin Small Outline Package (SO)	0 to 70°C	NE5532AD8	SOT96-1
8-Pin Small Outline Package (SO)	–40°C to 85°C	SA5532D8	SOT96-1
8-Pin Small Outline Package (SO)	–40°C to 85°C	SA5532AD8	SOT96-1
8-Pin Small Outline Package (SO)	-55°C to +125°C	SE5532AD8	SOT96-1
8-Pin Small Outline Package (SO)	0 to 70°C	NE5532D8	SOT96-1
8-Pin Small Outline Package (SO)	–40°C to 85°C	SA5532D8	SOT96-1
8-Pin Small Outline Package (SO)	–40°C to 85°C	SA5532AD8	SOT96-1
8-Pin Small Outline Package (SO)	-55°C to +125°C	SE5532D8	SOT96-1
16-Pin Plastic Small Outline Large (SOL) Package	0 to 70°C	NE5532D	SOT162-1
16-Pin Plastic Dual In-Line Package (DIP)	-55°C to +125°C	SE5532N	SOT38-4

9.3.6 Potentiometer

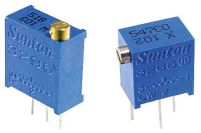
Square Trimming Potentiometer – 3296 & 3266

Suntan®

TSR-3296 & 3266

瑯 釉 微 調 電 位 器

- 多圈的/金屬陶瓷/工業用/密封
- (Multiturn/ Cermet/ Industrial/ Sealed)
- 5 種引線的安裝尺寸
- (5 Terminal Styles)



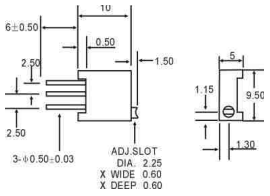
TSR-3296 TSR-3266

	TSR-3296	TSR-3266
標稱阻值範圍 Standard Resistance Range	10Ω - 5MΩ	10Ω - 5MΩ
阻值允許偏差 Resistance Tolerance	±10%	±10%
終端電阻 Absolute Minimum Resistance	≤1% R 或 10n	≤1% R 或 10Ω
接觸電阻變化 Contact Resistance Variation	CRV≤3%或 5n	CRV≤3%或 5Ω
絕緣電阻 Insulation Resistance	R1≥1GΩ(100Vac)	R1≥1GΩ(100Vac)
耐電壓 Withstand Voltage	640Vac	500Vac
有效行程 Effective Travel	30±2 turns nom	12±2 turns nom

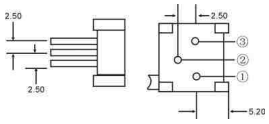
	TSR-3296	TSR-3266
額定功率 Power Rating (315 volts max)	0.5W@70℃, 0W@125℃	0.25W@70℃, 0W@125℃
溫度範圍 Temperature Range	-55℃ ~ +125℃	-55℃ ~ +125℃
溫度系數 Temperature Coefficient	±250 或 ±100ppm/℃	±250 或 ±100ppm/℃
溫度變化 Temperature Variation	-55℃,30min, +125℃ 30min, 循環 5 次 5 cycles ΔR≤5%R,Δ(Uab/Uac)≤5%	-55℃,30min, +125℃ 30min, 循環 5 次 5 cycles ΔR≤5%R,Δ(Uab/Uac)≤5%
振動 Vibration	10 – 500Hz, 0.75mm, 6h ΔR≤5%R,Δ(Uab/Uac)≤7.5%	10 – 500Hz, 0.75mm, 6h ΔR≤5%R,Δ(Uab/Uac)≤7.5%
碰撞 Collision	390m/s², 4000cycles, ΔR≤5%R	390m/s², 4000cycles, ΔR≤5%R
70℃耐久性 Electrical Endurance at 70℃	0.5W@70℃ 1000h, ΔR≤10%R CRV≤3% or 5Ω	0.25W@70℃ 1000h, ΔR≤10%R CRV≤3% or 5Ω
機械壽命 Rotational Life	200 周 (200 cycles) ΔTR≤±10%R,CRV≤3% or 5n	200 周 (200 cycles) ΔR≤10%R,CRV≤3% or 5Ω

TSR-3296

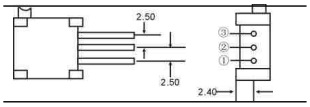
共有尺寸 Common Dimensions



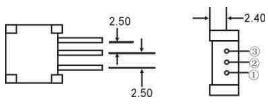
TSR-3296P



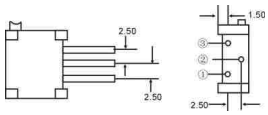
TSR-3296X



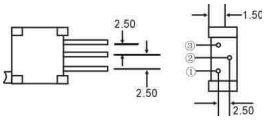
TSR-3296W



TSR-3296Z



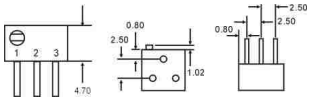
TSR-3296Y



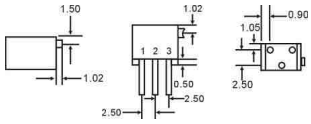
TSR-3266

共有尺寸 Common Dimensions

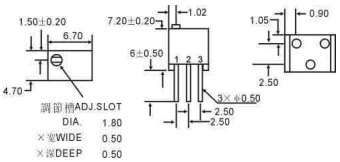
TSR-3266P



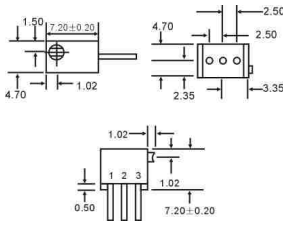
TSR-3266X



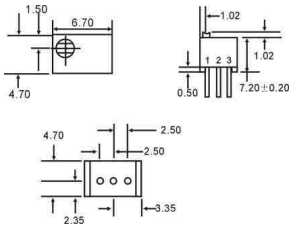
TSR-3266W



TSR-3266Z



TSR-3266Y



阻值代碼表

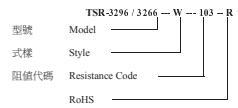
Standard Resistance table

10	100
20	200
50	500
100	101
150	151
200	201
250	251
300	301
470	471
500	501
680	681
1,000	102
1,500	152
2,000	202
2,200	222
2,500	252
3,000	302
4,700	472
5,000	502
6,800	682
10,000	103
15,000	153
20,000	203
22,000	223
25,000	253
30,000	303
33,000	333
47,000	473
50,000	503
68,000	683
100,000	104
150,000	154
200,000	204
220,000	224
250,000	254
300,000	304
330,000	334
470,000	474
500,000	504
680,000	684
1,000,000	105
2,000,000	205
2,200,000	225
5,000,000	505

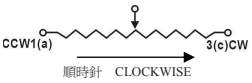
可按用戶要求提供特殊規格

Special resistances available.

訂購指南 How to Order



電刷 WIPER 2(d)



圖中公差 ：除注明外均爲±0.25

Tolerance is ±0.25 if no identification

起動力矩 Starting Torque	≤35mN·m
標誌 Marking	阻值允許偏差(±10%不標注) 阻值代碼/產品型號 Resistance Tolerance (When no identification, it is of ± 10%) Resistance Code, Model
標準包裝 Standard Packaging	50 只/管 (50pcs, per tube)

Note: Specification are subject to change without notice. For more detail and update, please visit our website.

Suntan® Technology Company Limited Website: www.suntan.com.hk Email: info@suntan.com.hk Tel: (852) 8202 8782 Fax: (852) 8208 6246

9.3.7 DC-to-DC converter

SIM 2 - SIL 7

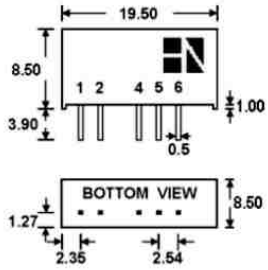
2 Watt Ultra-Miniatur SIL-Modul-Serie

2 W DC/DC SIM-SIL MODUL

Bestell-Information / Order Information

Modell	Eingang/Input V	Ausgang 1/Output 1 V / mA	Ausgang 2/Output 2 V / mA
SIM2-0505S-SIL7	5	5/400	
SIM2-0512S-SIL7	5	12/166	
SIM2-0515S-SIL7	5	15/132	
SIM2-0905S-SIL7	9	5/400	
SIM2-0912S-SIL7	9	12/166	
SIM2-0915S-SIL7	9	15/132	
SIM2-1205S-SIL7	12	5/400	
SIM2-1212S-SIL7	12	12/166	
SIM2-1215S-SIL7	12	15/132	
SIM2-1505S-SIL7	15	5/400	
SIM2-1512S-SIL7	15	12/166	
SIM2-1515S-SIL7	15	15/132	
SIM2-0512D-SIL7	5	12/82	-12/82
SIM2-0515D-SIL7	5	15/66	-15/66
SIM2-0905D-SIL7	9	5/200	-5/200
SIM2-0912D-SIL7	9	12/82	-12/82
SIM2-0915D-SIL7	9	15/66	-15/66
SIM2-1205D-SIL7	12	5/200	-5/200
SIM2-1212D-SIL7	12	12/82	-12/82
SIM2-1215D-SIL7	12	15/66	-15/66
SIM2-1505D-SIL7	15	5/200	-5/200
SIM2-1512D-SIL7	15	12/82	-12/82
SIM2-1515D-SIL7	15	15/66	-15/66

PIN-Belegung und Zeichnung / Pin assignments & drawing, mm (inch)



PIN CONNECTION		
	SINGLE	DUAL
1	Vin +	Vin +
2	Vin -	Vin -
4	Vout-	Vout-
5	OMITTED	COMMON
6	Vout +	Vout +

Tolerance XX.XX +/- 0.25
All Dimensions in m. m.

MEMO:

HN Electronic Components GmbH • Birkenweiherstr. 2 • D-63505 Langenselbold
Tel. (06184) 92790 • Fax (06184) 62316 • http://www.hn-electronic.de

D B

Glossary

ADC	analog-to-digital converter
ASCII	American Standard Code for Information Interchange
BACnet	building automation and control networks
DAC	digital-to-analog converter
DDC	automation station for building facilities
GPIO	general purpose intput / output
HIL	hardware-in-the-loop
IDE	integrated development environment
I/O	input / output
IVP	initial value problem
MPC	model predictive controller
ODE	ordinary differential equation
OS	operating system
UDOO	single-board computer

References

[1] Arduino: *Arduino official web page*. http://www.arduino.cc/en/Reference/HomePage. – Accessed: 31.05.2015

[2] Beier, T. ; Wurl, P.: *Regelungstechnik - Basiswissen, Grundlagen, Anwendungsbeispiele. 1.* Carl Hanser, München, 2013

[3] Bernstein, Herbert: *Grundlagen der Hard- und Software der Mikrocontroller ATtiny2313, ATtiny26 und ATmega32*. Wiesbaden: Springer, 2015. – ISBN 978–1–4200–9157–1. – Accessed: 31.05.2015

[4] Dembowski, K.: *Mikrocontroller - Der Leitfaden für Maker - Schaltungstechnik und Programmierung von Raspberry, Arduino & Co. 1.* dpunkt.verlagwe, 2014

[5] Ding, Bao-Cang: *Modern Predictive Control*. CRC Press, 2010. – ISBN 978–1–4200–9157–1. – Accessed: 31.05.2015

[6] Dugge, K. ; EiSSner, A.: *Grundlagen der Elektronik*. 6. Vogel, Würzburg, 1997

[7] Faes, G.: *Eine Einführung in das Mathematikprogramm Scilab*. 1. Books on Demand, Norderstedt, 2014

[8] Hwang, T. ; Rohl, J. ; Park, K.: Development of HIL Systems for active Brake Control Systems. In: *SICE-ICASE International Joint Conference* 30 (2006), December, Nr. 1, S. 205–231. http://dx.doi.org/10.1137/060676489. – DOI 10.1137/060676489. – Accessed: 31.05.2015

[9] Kofler, M.: *Linux das umfassende Handbuch*. Galileo Press, Bonn, 2014

[10] Kruppa, K.: Dokumentation Matlab HeatLib - Zur Vewendung mit Matlab / Hochschule für Angewandte Wissenschaften Hamburg. TU Hamburg-Harburg, 2014. – Dokumentation

[11] Observe: *official project webpage*. http://ob-serve.de/. – Accessed: 31.05.2015

[12] Pangalos, G. ; Eichler, A. ; Lichtenberg, G.: Tensor Systems: Multilinear Modeling and Applications. In: *3rd Int. Conference on Simulation and Modeling Methodologies, Technologies and Applications*, 2013. – submitted

[13] Papula, L.: *Mathematik für Ingenieure und Naturwissenschaftler*. 12. Vieweg+Teubner, Wiesbaden, 2009

[14] Peeters, L. ; Veken, J. V. ; Hens, H. ; Helsen, L. ; D?haeseleer, W.: Control of heating systems in residential buildings: Current practice. In: *Energy and Buildings* 40 (2008), Nr. 8, 1446 - 1455. http://dx.doi.org/http://dx.doi.org/10.1016/j.enbuild.2008.02.016. – DOI http://dx.doi.org/10.1016/j.enbuild.2008.02.016. – ISSN 0378–7788

[15] Peter, Kieback : *company webpage*. http://www.kieback-peter.de. – Accessed: 31.05.2015

[16] Privara, Samuel ; Siroky, Jan ; Ferkl, Lukas ; Cigler, Jiri: Model predictive control of a building heating system: The first experience. In: *Energy and Buildings* 43 (2011), Nr. 2?3, 564 - 572. http://dx.doi.org/http://dx.doi.org/10.1016/j.enbuild.2010.10.022. – DOI http://dx.doi.org/10.1016/j.enbuild.2010.10.022. – ISSN 0378–7788

[17] Reuther, K.-H.: *Grundlagen der Regelungstechnik - Eine Einführung in die Theorie und die analoge und digitale Realisierung*. 2. Shaker, Aachen, 2007

[18] Schmidt, M.: *Arduino - Ein schneller Einstieg in die Microcontroller- Entwicklung*. 2. dpunkt.verlag, Heidelberg, 2015

[19] Schmitt, G.: *Mikrocomputertechnik mit Controllern der Atmel AVR-RISC-Familie*. 5. Oldenbourg Verlag, München, 2010

[20] Scilab, INRIA: *Scilab official web page*. http://www.scilab.org/resources/documentation. – Accessed: 31.05.2015

[21] Stephan, K. ; Mayinger, F.: *Thermodynamik - Grundlagen und technische Anwendungen*. 12. Springer, Berlin, Heidelberg, New York, 1986

[22] Tkotz, K.: *Fachkunde Elektrotechnik*. 26. Europa-Lehrmittel, Norney, 2008

[23] UBUNTU: *official webpage*. https://ubuntu.com/. – Accessed: 31.05.2015

[24] UDOO: *official webpage*. http://udoo.org/. – Accessed: 31.05.2015

[25] Weichinger, K.: *Scicos Serial-Interface-Block Manual V12.05*. www.bioe.at.tt. – Accessed: 31.05.2015

[26] WIKI, UBUNTU: *User-platform for german speaking Ubuntu users*. https://wiki.ubuntuusers.de/. – Accessed: 31.05.2015