H-Storm System Manual

Revision: 1.0

Introduction

H-Storm is a modular system level design approach targeted towards home-brew and hobby CPU-based projects, like home automation, robotics, measurement and control applications and so on. It's designed to be versatile, yet easy to work with even for people with limited experience in high-speed digital design.
The goal of H-Storm is to bring the power of modern 32-bit processors within the reach of the hobbyist. It is intended to be a more capable replacement of BasicStamp™, SimmStick™ and the like.
Unlike however those proprietary solutions, H-Storm is an open design environment where you can (and encouraged) to add your own designs and changes to the ever growing set of common knowledge.

A little bit of history

H-Storm is based on the SIMM-Sys project, started at RCS ltd. a couple of years ago. I used many of the ideas of SIMM-Sys when laid down the foundations of the H-Storm but I also incorporated the experiences I had during the years since I've stopped actively working on SIMM-Sys. The two projects are very similar electrically but there are some important differences. Interfacing between the two design concepts is easy. The mechanical design of the two projects are also different, as well as the target audience. The copyright model for the two initiatives is somewhat different too.
If you are familiar with the SIMM-Sys concept or just interested in the details of the differences, please read the chapter: H-Storm and SIMM-Sys: a comparison

Table of Contents

Legalese

All of the documentation and software included on the H-Storm project website (including this document) is copyrighted by Andras Tantos and Modular Circuits.
The documents on this website are covered by various licenses, depending on their content, origin, or intent. Information on H-Storm in general, like pin-outs, electrical specifications, and so on is free for almost any kind of use, provided that you don't claim that you created that information. This document falls into that category as well. Documents of this kind are covered by a BSD-like license, called H-Storm Open License. According to this license, for example, you can use this documentation to build a gigabit-Ethernet interface card with an H-Storm interface. You can publish the design, if you wish or you can keep it for yourselves if that's what you like. You can sell it or you can build it into your own bigger solutions. Almost the only thing you can't do is to claim that H-Storm was your idea. Content falling into this category is either not marked or marked with this symbol: HSOL
There are also designs (schematics, PCB layouts) presented on the H-Storm project website. These designs are free for non-commercial use i.e. as long as you don't make any money out of them. You are free to re-build the circuits in their original or in a modified form; you can derive your own design from them. You are not required to publish your work based on these designs but you might do so, again, as long as you don't make money off of them. If you however would like to 'productize' some work based on the designs on this website, you have to get my prior written approval for that. I reserve the right to deny such an approval or request some form of compensation for it, so please contact me at andras_tantos@yahoo.com! This type of information is covered under the H-Storm Non-Commercial License. Content falling into this category is marked with this symbol: HSNCL. Also, any circuit diagram, PCB layout, automatically falls into this category unless marked otherwise.
Source code and binary software presented on this web-page is usually covered under some form of open-license (one of the above two , GPL, LGPL, BSD, etc.) depending on what is the source of the software. The type of license is marked on the web-page by the download link to the particular piece of code and in the source code as well.
And as always: if in doubt, ask!
This document is covered by the H-Storm Open License (HSOL).

H-Storm Open License (HSOL)

Copyright 2004 Andras Tantos and Modular Circuits. All rights reserved.
Redistribution and use in source or binary forms, or incorporated into a physical (hardware) product, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in any other form must contain in printed or electronical format the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- All advertising materials mentioning features or use of this technology must display the following acknowledgment:
  This product includes H-Storm technology developed by Andras Tantos and Modular Circuits.
- Neither the name of Andras Tantos or Modular Circuits may be used to endorse or promote products derived from or using this technology without specific prior written permission.

ALL THE INFORMATION, TECHNOLOGY, AND SOFTWARE IS PROVIDED BY THE AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL ANDRAS TANTOS, MODULAR CIRCUITS OR

CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE OR TECHNOLOGY, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

H-Storm Non-Commercial license (HSNCL)

Copyright 2004 Andras Tantos and Modular Circuits. All rights reserved.
Redistribution and use in source or binary forms, or incorporated into a physical (hardware) product, with or without modification, are permitted for **non-commercial use only**, provided that the following conditions are met:

- **The redistribution doesn't result in financial gain.**
- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in any other form must contain in printed or electronical format the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- All advertising materials mentioning features or use of this technology must display the following acknowledgment:
  This product includes H-Storm technology developed by Andras Tantos and Modular Circuits.
- Neither the name of Andras Tantos or Modular Circuits may be used to endorse or promote products derived from or using this technology without specific prior written permission.

ALL THE INFORMATION, TECHNOLOGY, AND SOFTWARE IS PROVIDED BY THE AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL ANDRAS TANTOS, MODULAR CIRCUITS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE OR TECHNOLOGY, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Other licenses

No information in this document is covered by either of the following licenses. Some software, hardware or documentation however on the site can be covered by some of these (or another) license. For this reason a reference to these licenses are included here:
The GNU General Public License (GPL):
http://www.opensource.org/licenses/gpl-license.php
GNU Lesser General Public License (LGPL):
http://www.opensource.org/licenses/lgpl-license.php
BSD License (BSD):
http://www.opensource.org/licenses/bsd-license.php

Features

The H-Storm is a modular system-level development environment targeted towards the hobby and home-user.

High-level features

- Highly integrated system components for simple system integration
- Flexible design to accommodate special needs
- Common base architecture for interoperability and for common knowledge base
- Reasonable level of compatibility: compatible where practical but allows for incompatibilities where necessary.
- Rich configuration record for each module that allows for automatic system discovery
- Low-cost solutions for fast adoption
- Expandable to accommodate future needs
- Targeted towards hobbyist and low-volume manufacturing
- Open technology
- Free sample designs
- Simple to use and easy to manufacture mechanical design

Technology details

- 16-bit bus design for moderate-speed communication between modules
- Single-master design with optional DMA
- Optional external wait-state generation
- Interrupt support
- Two-wire interface side-band configuration and plug-and-play bus
- Low-power operation with 3.3V signaling
- Triple power supply support (3.3V, 2.5V, 1.8V)
- 100mm (3.94”) wide standard modules with module-dependent height
- System-controller on peripherial and mother-boards can isolate components from the rest of the system.

Goals and non-goals

It’s important to state the goals of a project so that we can keep on focusing on the things we actually trying to achieve. It’s equally important though to state what we don’t try to achieve to so that we can keep the design simple and targeted. Some of the items in the lists below I think pretty are straightforward, others need more explanations. Those will be detailed latter on, but for now, let’s see the shopping-list:

The goals

- Provide a versatile, easy to use modular development environment
- Primary target audience are hobbyists, home robot builders, and small electronic workshops
- Provide significantly higher performance than BasicStamp™, SimmStick™ and similar solutions while keeping the price comparable
- Provide a fast, 16-bit interconnect between the system modules
- Keep the interface simple so that stock semiconductors would be easy to use
- Keep the design simple, straightforward and cheap
- Give an upgrade path to higher performance CPU boards
- Support 16-32 bit CPU architectures
- Make system-integration as simple as possible
- Provide reasonable compatibility between the modules of the system

- Make the system completely software-discoverable

The non-Goals

- Don't support 8-bit microprocessors
- Don't support microcontrollers without external data- and address-buses
- Target mid-range data-transfer rates: don't try to push to the territory of PCI or above
- Target mid-range CPU performance: keep I/O and processing power in sync. Don't try to integrate multi-gigahertz CPUs
- Don't try to make every module 100% compatible with every other module
- External bus is for peripherial connections: external memory, especially for program execution is not a primary target.

High-level system walk-through

System components

H-Storm identifies three types of components.

- CPU cards, that contain some kind of a micro-processor, in most cases a powerful 32-bit micro-controller, ROM and RAM memory and some integrated peripherials.
- Peripherial cards, that provide some kind of interface to the outside world. These cards are connected to the CPU card using a 16-bit parallel bus to provide high-speed data-transfer. They interface to the outside world by standard or application-specific connectors.
- System-boards, that provide the application-specific functionality of the design. They contain the power supply and distribution network, additional application-specific interfaces to the outside world as well as mechanical support for the CPU and Peripherial boards.

Interconnect

In H-Storm there are two buses connect each element to each other. The 16-bit parallel asynchronous **system bus** is used for most of the data-transfers between the peripherial devices (including the system-board) and the CPU card.

There is a secondary two-wire side-band bus, called **PnP bus** that is used primarily to download configuration information from each of the peripherial cards and the system-board to the CPU card. This rich plug-and-play information provides full software discoverability of the system. This bus can however be used as a general, low-bandwidth side-band bus to provide an additional communication channel between the peripherials and the CPU card.

System integration

The H-Storm system-architecture is built around the idea of generic CPU and Peripherial cards and application-specific mother-board designs. While these rules don't fit all cases, usually the system-board has to be be tailored for the specific application. Into this special system-board plug the generic CPU and peripherial cards. This is the opposite in concept of the traditional PC approach where you expand a generic mother-board with application-specific extension cards. The H-Storm approach fits better the world of embedded design (including hobby electronics) since it gives more freedom in the implementation while retains the advantages of using stock components for common functions.

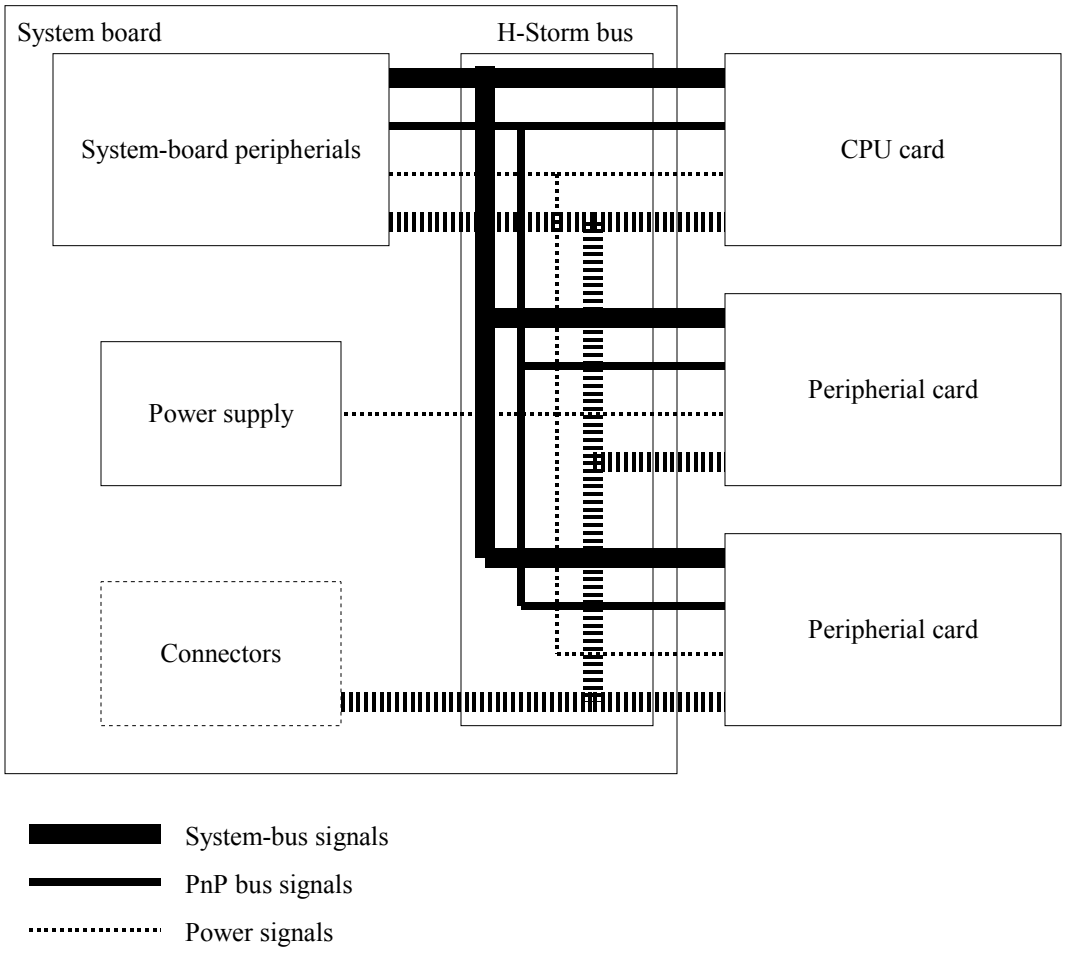

Figure 1: Typical H-Storm system

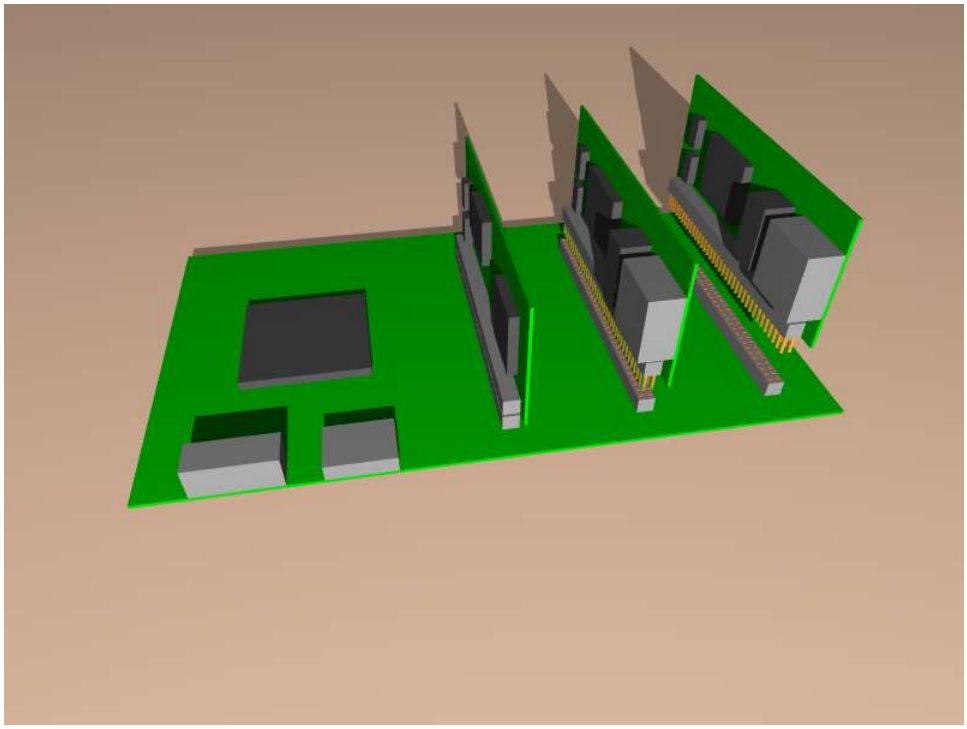

Figure 2: 3D rendering of a typical H-Storm system

Module compatibility

The H-Storm concept doesn't provide or require 100% compatibility between all modules possible. That would require unreasonable amount of effort and would add up to the price-tag as well. Instead it requires that the functionality and compatibility issues are clearly stated in the module data-sheet and in the configuration PROM content as well. It is therefore possible to design a system where the components work together nicely since the designer can check the timing, power and other requirements for possible incompatibilities. It is also possible for the CPU card to detect the requirements of each peripherial cards attached to the system as well as those of the system board and make sure that the actual system components can operate together as a whole.
As an example, the H-Storm specification permits both edge- and level-driven IRQ signals. It would not be reasonable to discredit a component just because it doesn't support for example edge-driven interrupts. It would add up to the price of the total system if it would require conversion of level-driven IRQs to edge-driven ones. Besides, many modern CPUs have support for both types so in many cases such a requirement would be unnecessary. Instead H-Storm requires that the type of interrupt a peripherial card generates and a CPU card can handle is clearly stated in the data-sheet and in the configuration PROM. This ensures that at component selection time the right devices can be designed in the whole system. It also ensures that the CPU can configure its interrupt lines correctly (if its possible) for each peripherial card or at least make sure that the peripherial cards are compatible with the CPU card.
Another example would be the nWAIT signal. Its not reasonable to lock CPUs out from H-Storm concept that doesn't support external wait-states (ADSP218x series is a good example here). It would also unreasonable to disallow any peripherial cards that can't operate in an environment where the CPU cannot accept external wait-states. Instead, in H-Storm, the configuration PROM contains fields that define the nWAIT requirement of each peripherial card. There's a field that states that the card can generate wait-states, and another that states that the card can't operate without wait-states. This latter basically says, that there's no predefined maximal time over which an access must complete on the peripherial. Such peripherials would be incompatible with CPU cards

without external wait-state generation support, but that case can be discovered by matching the data-sheets or at system configuration time by the CPU by examining the configuration PROM content.

Mechanical design

The central mechanical piece of each H-Storm system is the system-board. CPU and peripherial cards plug into this part of the system. CPU and peripherial cards cannot be plugged into each other. No stackable design is supported. All standard modules in the system are connected to each other through a standard 72-pin 100mil dual-line header. The CPU and Peripherial cards have the male part, while the system-board contains the female part. Extended modules have one or two additional 20pin 100mil dual-line headers. All CPU and peripherial cards are 100mm (3.94") wide with a module-dependent height. There are no limitations on the mechanical design of the system-boards other than adequate room must be left around the CPU and peripherial sockets so that the cards can be inserted into them.

System components

CPU cards

In every H-Storm system there's at most (and usually) one CPU card. The card is the central 'brain' of the system and the source of all transactions. No multi-master operation is supported by the design. If multiple CPUs are required by the system, arbitration has to be accomplished by other means, not specified by the bus, or a split-bus design must be used. The CPU card usually contains a 32-bit processor, some RAM and ROM memory and peripherials. The usual case would be that the CPU is integrated into a system-on-chip solution along with the peripherials and some amount of memory. This chip is accompanied by FLASH ROM and/or RAM on the board to provide adequate memory size. The card also contains some support circuitry, like address decoders, bus-drivers, power supplies, reset-generators or clock sources.
The CPU card communicates with the outside world using its on-board peripherials and its system and PnP buses.
The executed program comes from the on-board memory of the card. Code execution from devices connected to the external bus may or may not be supported. Since the standard connector contains only 10 address lines, the accessible external address space is only 2kBytes per module. Though that's more than enough for most peripherials it's most likely inadequate for program execution from those locations. If the CPU card however has the extended connector B, which adds 12 more address lines and extends the addressable memory space to 8MByte per module, external program execution makes more sense.
A CPU card can address three external peripherials. The usual case would be that one of them is the system-board the CPU card is plugged into, while the other two are peripherial cards, also plugged into the same system-board.
The CPU card can use the two-wire PnP bus to discover the configuration of the environment it is working in. It can retrieve detailed information of the characteristics of the modules attached to the bus, so that it can set up its functionality to be compatible with those modules. If such a set up is not possible it can detect that an incompatible card is attached to the system, and either stop execution or isolate the module to prevent erratic behavior.
Physically standard CPU cards are 100mm (3.94") wide modules, with a 72-pin dual-line 100mil 90 degree right-angle mail connector on their side. The height of the cards are not specified but would be in the 3-8cm (1-3") range. Extended CPU cards are 160mm wide with one 72 pin and two 20 pin connectors. See Mechanical Design chapter for details.

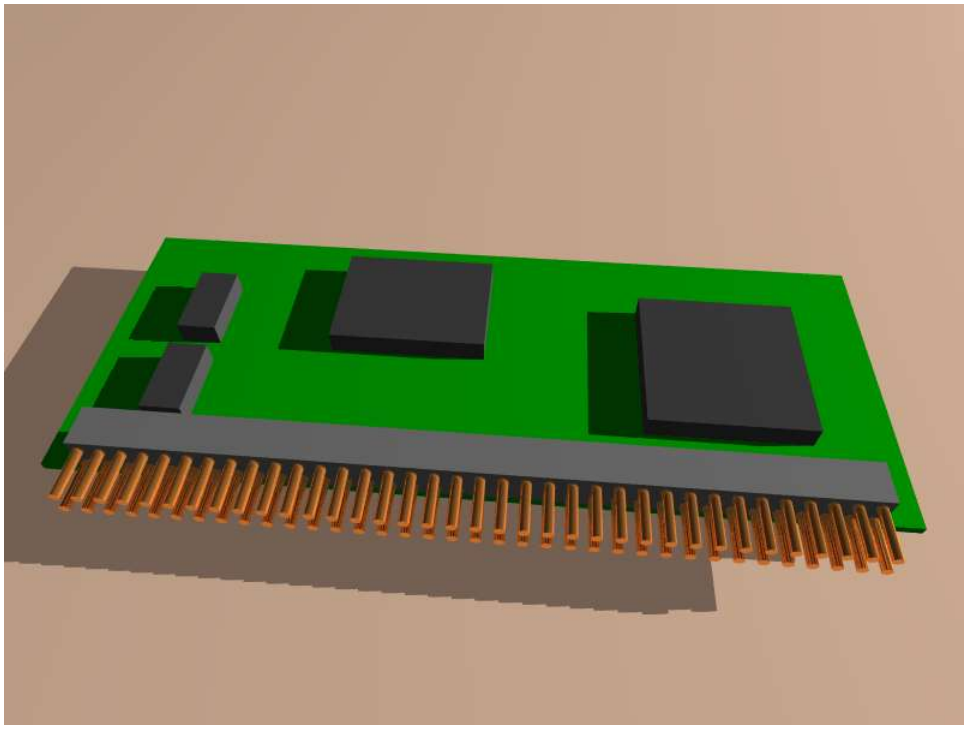

Figure 3: 3D rendering of a typical CPU card

Peripherials

Peripherial cards, according to the main concept of the H-Storm are communication protocol interface cards, that allow for the connection of some sort of standard communication channel. Examples would be RS485, CAN-bus, Ethernet or USB.
This assumption is coming form the observation that these interfaces are usually standard enough throughout various applications that a stock implementation from them would make sense. This isn't however a rule without exceptions. There are other types of peripherial cards and there's no rule against implementing any function in a peripherial card. Examples of non-communication type peripherial cards would be FPGA or CPLD cards, display controller cards, AD/DA converters, isolated digital I/O cards, etc.
Peripherial cards interface to the H-Storm bus in 8- or 16-bit width. It is possible that a peripherial card doesn't support 16-bit transfers or it only support transfers on the lower 8-bits of the bus. This later mode however reduces the addressable locations on the peripherial cards address space to 1024. It also means that (from the operating software's standpoint) registers on the peripherial card appears on even addresses only and are separated by 8-bit wide reserved locations. It is recommended that 8-bit only peripherial cards simply ignore the nHWE signal for writes, and doesn't drive the upper 8 bits of the address bus for reads. This easy to implement technique would still allow 16-bit read or write cycles to be compatible with the device, with of course the upper 8-bit of the data being ignored (in case of writes) or meaningless (for reads).
The application-specific I/O signals on the connector may or may not be used. If the implemented interface has a well-defined physical connector, it is recommended that the card have that connector so that cables can easily be attached to it. So for example an USB target interface would have an USB type B connector on it, while an Ethernet card would have an RJ45 jack. It is also recommended however (unless EMI and signal-integrity consideration suggest otherwise) that those signals are also available on the application-specific part of the H-Storm connector. This would make sure that a system-board device can interface to the peripherial if it wants to. See Mechanical Design chapter for details.

Peripherial cards share the same form-factor as the CPU cards. They are 100mm or 160mm wide, and have dual-line 100mil male connectors to plug them into an H-Storm system-board.

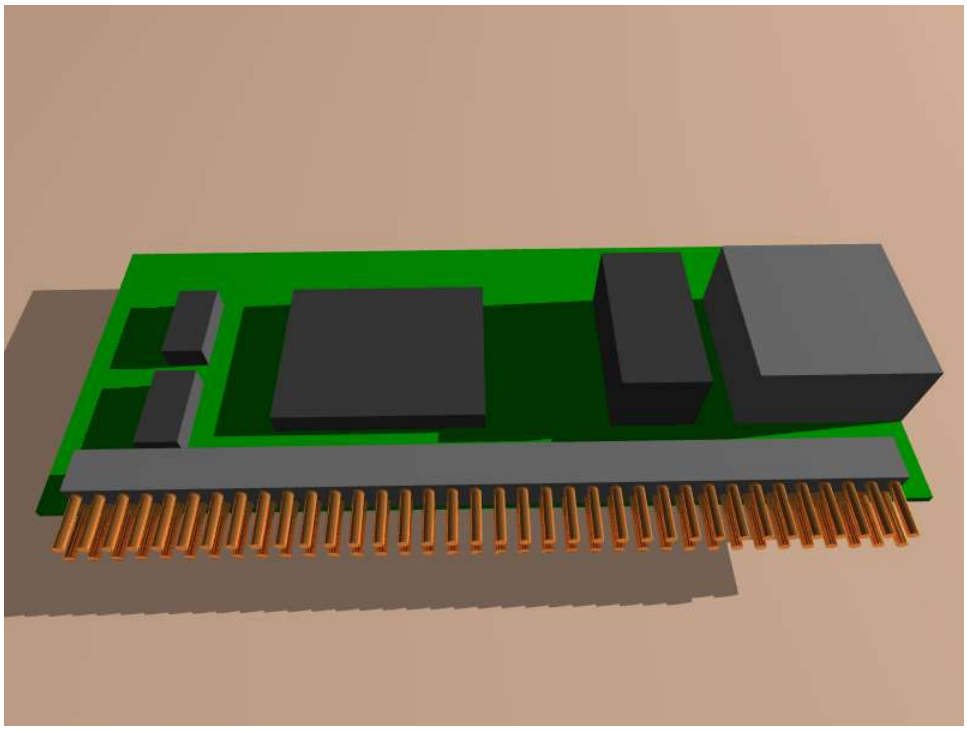

Figure 4: 3D rendering of a typical peripherial card

System-board

For each H-Storm system the system-board provides the framework both electrically and mechanically into which CPU and peripherial modules fit. System-boards are an essential part of each H-Storm system though their functionality is much less strictly defined than the other components of the system. This is because, while other components have a fairly well defined predetermined role, system-boards contain all the application-specific functionality.

Even so, there are a couple of common aspects of system-boards for H-Storm designs that are worth iterating:

- System-boards provide mechanical support for the CPU and peripherial cards. Since CPU and peripherial cards have a 90degree right-angle connector, they will stick out vertically from the system-board. Standard CPU and Peripherial cards are 100mm wide, so enough room should be left on both sides of the socket so that those cards can actually be inserted into the socket. There should be a 5mm clearance on each side of the connector where no component, higher than the female connector should be placed. The clearance between H-Storm bus sockets should be no lower than 20mm to prevent connectors on peripherial cards from touching the adjacent card. See Mechanical Design chapter for details.

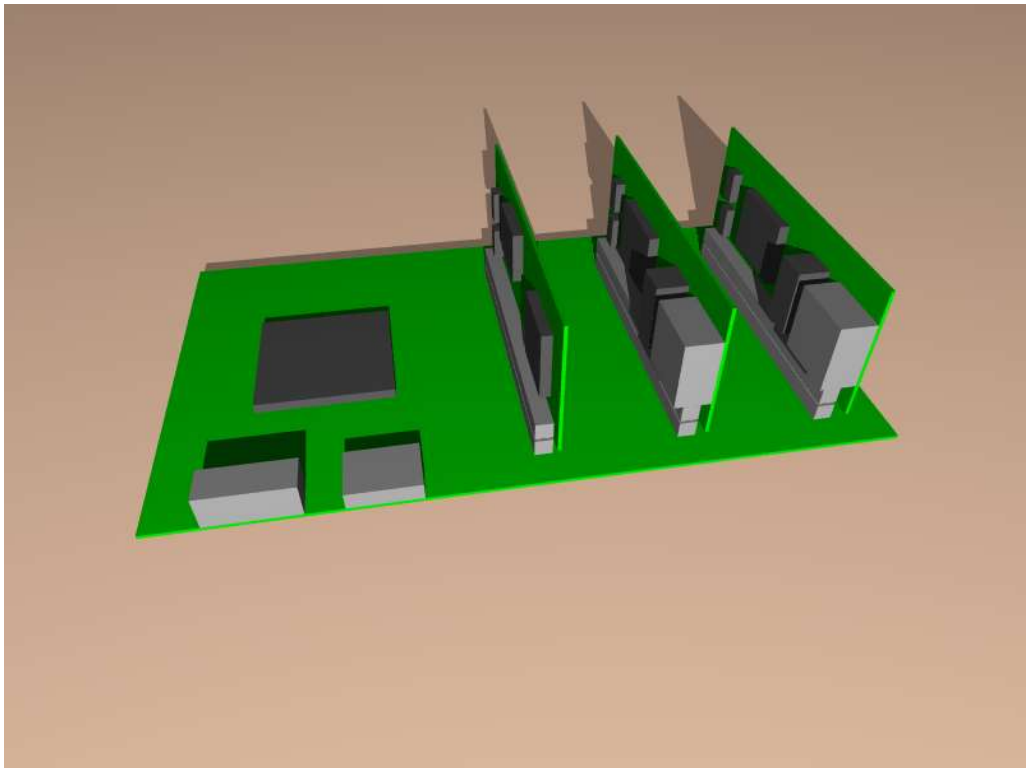

Figure 5: System board with CPU and peripherical cards

- A system-board defines the number of Peripherial and CPU cards that can be used in the design. A full-blown system-board would usually contain one CPU and two peripherial card sockets since a CPU card can address three external peripherials. The usual case would be that one of the peripherial select signals (nSEL0) would enable communication with the components on the system-board. The other two would address peripherial cards, plugged into the system-board. It however is possible that the system-board integrates one or more peripherial functions onto it and maps those to the first or the second peripherial cards. It is also possible that a system-board doesn't allocate any peripherial slots and allows for three peripherial cards to be connected to the CPU card. This type of system-board is called a **passive system-board** even if it contains active components, like a power supply or a PnP EPROM. If a local CPU is built on the system-board it's also possible that no CPU card socket is provided.
- The system-board is also responsible for the electrical connections between the various elements of the system. This means that the common system-bus signals of the H-Storm connectors must be tied together. Care must however be taken to the connection of the nSELx, nIRQx (and in the case of extended connectors nDMAx) signals. These are rotated between each connector. The nSEL0 signal of the CPU card connector is connected to the nSEL2 of the first and the nSEL1 pin of the second peripherial card. The nSEL1 signal is connected to the nSEL0 of the first and the nSEL2 of the second peripherial card. Finally the nSEL2 signal of the CPU card is connected to nSEL1 of the first and nSEL0 of the second peripherial card connector. nIRQx and nDMAx signals are rotated in the same fashion. Usually nSEL0, nIRQ0 and nDMA0 are the signals that the system-boards built-in peripherials use to communicate with the CPU card. This technique makes it possible that the CPU card can select each peripherial independently while all the peripherial functions respond to the select signal nSEL0 on their socket.

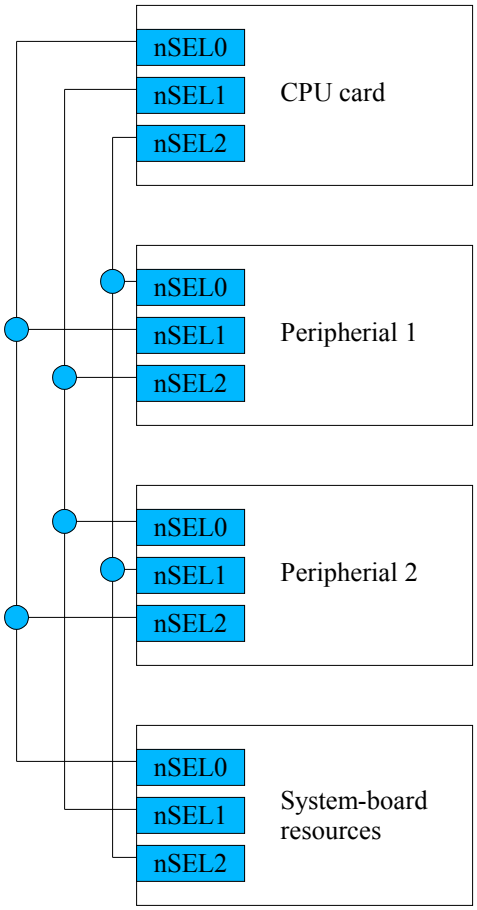

Figure 6: Signal rotation

- Another function of the system-board is to provide distribution of the two-wire PnP-bus signals towards the peripherial cards. The primary purpose of this bus is to provide access to the plug-and-play information record, stored in a small PROM on each peripherial board (and on the system-board as well). It also provides a way for the CPU card to isolate peripherial cards from the system-bus and might serve as a generic low-bandwidth communication interface to the peripherial cards and the system-board. All Peripherial cards are required to implement this interface and access to a PnP configuration ROM. Most system boards except for the simplest passive ones would also contain a PnP configuration ROM.
- Another main functionality of the system-board is to provide power to all the other subsystems of the design. In the generic case the system-board would generate three power sources: 3.3V, 2.5V and 1.8V. However if at the design of the system it is known that only single or dual-supply components will be chosen one or two of the supplies can be left out. Many CPU and peripherial cards are available in two versions: one with an integrated power supply that would generate the required lower voltages from a single 3.3V supply and another that would depend on the system-board to provide the required voltages. When only a single module requires a voltage other than 3.3V probably the first option is the favorable, however if multiple components of the system require a lower voltage as well, a central power source can be more reasonable.
- The last main group of functionality that the system-board serves is the connection to the external world. This can mean as little as level-shifters and connectors (though if the peripherial boards provide the right connectors even that might be omitted) and as complex as analog-to-digital converters, isolators, motor drivers, displays, buttons or any arbitrary complex functionality that the design requires.

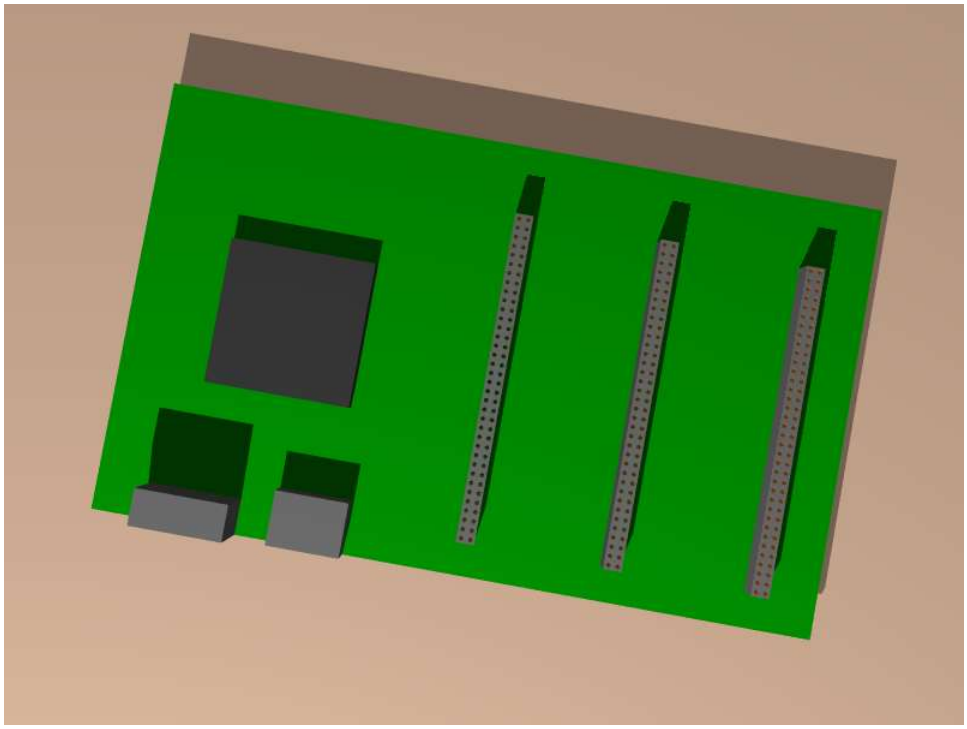

Figure 7: 3D rendering of a typical system-board

Electrical interfaces

The components of an H-Storm system are connected together through a 72-pin connector. This connector contains various signals comprising different groups. These groups are the system-bus, the PnP-bus, the power rails, the user-defined signals and the 'nPROG/nBRD_EN' pin.

The H-Storm system-bus

The H-Storm system-bus is a simple 16-bit asynchronous bus with individual module-select, interrupt - and in the case of extended connectors, DMA - signals for up to three modules. It allows peripherials to generate external wait-states, but does not allow more than one bus-master on the bus. The extended version of the bus adds more address.
The standard bus contains the following signals:
3 module-select signals: **nSEL0**, **nSEL1** and **nSEL2**. An active, low level on these lines signals an active cycle towards that particular peripherial. At most one select signal is active at any given time. These lines are driven by the CPU card, the only bus-master in the system and monitored by the peripherials. A hard-wired rotation of these signals along the peripherial module sockets ensures that no resource conflicts arise as long as all modules use only a single select signal. The CPU card can discover resource-conflicts by consulting the PnP configuration PROM on the peripherial cards and could warn the user of an incorrect configuration or make preventive steps if possible.
The type of the access cycle is determined by three signals: **nRD**, **nLWE** and **nHWE**. These are also active low signals and encode the type of operation in the following way:

nRD	***nLWE***	***nHWE***	***Access type***
L	H	H	Read access
H	L	H	8-bit write access on the lower half of the bus
H	H	L	8-bit write access on the upper half of the bus
H	L	L	16-bit write access
H	H	H	Idle
L	H	L	Reserved and invalid
L	L	H	
L	L	L	

These signals are driven by the CPU card and shared among all members of the bus. The state of these lines is undefined when no active cycle is present on the bus (none of the nSELx are active).
The length of the access cycle depends on the CPU card and its configuration. The peripherial card can however make the access longer by pulling the **nWait** signal low. The peripherial card however should only assert nWait if it is selected by the appropriate nSELx signal. Some CPU cards might ignore the state of the nWait.
Peripherial cards should report their usage of the nWait signal in their PnP configuration block so that such incompatibilities can be automatically discovered. The nWait signal is a wired AND function of all the nWait sources of the peripherials and the system-board of the H-Storm system by utilizing open-drain drivers. The pull-up resistor that is required for such an operation is provided on the CPU card. This technique also makes possible that the nWait input of the CPU card is at a valid logic level even if no other party drives this signal.
The address of the transfer is presented on the address lines **A0..A10** (and **A11..A21** in case of an extended bus). These lines are never driven by other than the CPU card and listened to by all the peripherials. The state of these lines is undefined when no active cycle is present on the bus. It is not required to drive the address lines when no valid cycle is present on the bus.
The data is transferred between the communicating parties using the **D0..D15** lines. These can be driven by either the CPU card or the selected peripherial, depending on the direction of the transfer. The state of these lines is undefined when no active cycle is present on the bus. No peripherial card should drive any of these signals if it's not selected by an nSELx line.
The **nRESET** signal, when active (low), should initialize all devices in an H-Storm system to a valid, starting position. It can be driven low by any device by open-drain drivers, and must be pulled up on the CPU card. Pull-up on the Peripherial cards or the system-board can be provided as well. This way any element of the system can make sure that the required minimum length of the reset pulse is met. The CPU card is required to drive this line low for at least 1us upon power-up, while the system-board and the peripherials can chose not to if they can work with that short reset pulses. It is required that the state of the nPROG/nBRD_EN, the nRD, nLWE, nHWE, nSEL0...2 as well as the PnP-bus signals remain stable while the nRESET signal is active.
The signals **nIRQ0**, **nIRQ1** and **nIRQ2** are active-low interrupt signals that a peripherial or the system-board can use to draw the programs executed on the CPU attention to itself. The lines are driven by open-drain drivers on the peripherial drives, and pulled up on the CPU card. The same rotation of these lines are performed on the system-board as on the nSELx signals. Peripherial cards can only drive an nIRQx line if they also drive all lower numbered nIRQx lines. In practice all peripherial cards that use a single IRQ line are required to use nIRQ0. All cards that use two lines are required to use nIRQ0 and nIRQ1. However the wired AND logic on these lines makes possible to share an nIRQx line between multiple peripherials, provided the CPU card can handle such cases. The PnP configuration ROM is again a valuable resource to find possible conflicts in a system.
On the extended bus connector three DMA request signals **nDRQ0**, **nDRQ1** and **nDRQ2** are provided along with an acknowledge signal, **nDACK**. All are active low signals. Request lines are driven by the peripherial cards and the system-board in the very same fashion as nIRQx lines. The same wire-rotation, wired-AND line sharing and resource-conflict discovery through PnP configuration data applies here as well. nDACK is an additional access cycle qualifier that distinguishes normal access from DMA acknowledge cycles.

The nPROG/nBRD_EN signal

This pin has separate functions for CPU and peripherial cards.
On CPU cards this signal is named nPROG and when active (low), it puts the card into a special programming mode. This pin is used to download the initial software to the CPU cards FLASH memory. The exact way of doing that is card-dependent and not part of the H-Storm specification. This pin has a pull-up resistor on the CPU card and should not be driven under normal circumstances.
On Peripherial cards this pin is called nBRD_EN and is used for two purposes. During manufacturing its used to download the code to the PnP-bus system-controller. During normal operation this signal is high when the card is isolated from the system-bus and low when the card is connected to the system-bus. This signal has an internal pull-up on peripherial cards. The system-board can drive this signal low to inhibit the normal PnP-bus isolation logic and force the card to respond to operations on the system-bus. Inhibiting the PnP-bus isolation logic simplifies the operating software, however makes the PnP functions less capable. In particular it makes it harder to automatically detect which card is plugged into which socket.
It is very important the state of this signal to remain stable during reset.

The power rails

There are six power lines, divided into four classes on each H-Storm connector: two GND signals, two 3.3V, one 2.5V and one 1.8V nominal power supply lines. Extended connectors have additional GND and 3.3V power lines.

The PnP bus

Two signals, PnPD and PnPC are used for the side-band configuration bus and used to access the PnP configuration memories as well as any additional devices the peripherial cards and the system-board might attach to this bus. The physical signaling on this bus conforms to the two-wire serial interface specification (also known as I^2C™ bus). Each peripherial card and system-board is required to have a system-controller chip connected to this bus. This device implements the PnP discovery protocol and usually contains the PnP configuration information record as well. Additional two-wire interfaced devices can be connected to the PnP bus through the system-controller. More detailed description on how these devices need to be connected and how they operate can be found in chapter PnP bus operation

The user-defined signals

The rest of the pins are application-defined signals and can be used by either the CPU or the peripherial cards as they like to. Recommendation for common use of these signals exist however to ensure at least limited compatibility between peripherials with similar functionality.

Power considerations

In a usual H-Storm system the system-board provides the power for the CPU card as well as all peripherials in the system. To ensure compatibility, power budgeting must be done so that the system-board designer can put the right supply for the application in place.
H-Storm requires that power-requirements for each peripherial or system-board are listed in the configuration EPROM. Separate entries are used for devices that can supply power to the system (usually the system-board) and for devices that consume power. Separate entries are used to describe power consumption under various conditions.
On the top of this fine-grain discoverability of the system, hard limits also exist on the amount of power a single CPU or Peripherial card can consume: None of these cards can draw more than 1A of current on any of their

power lines (VCC_3, VCC_2_5, VCC_1_8). Of course there's no limit on the amount of power a power supply can produce.

H-Storm connector pin-outs

All standard CPU and Peripherial boards have a 72-pin dual-line 100mil standard connector. This connector is divided into two areas. Approximately half of the pins are reserved for the system-bus, the PnP bus and other purposes, and their functionality is fixed. The other half of the pins are application-specific I/O pins. Each card is free to use these latter pins in any which way they desire, however there are some guidelines on how to layout common functions to those pins. Extended connector layouts are also defined to add more pins if a card design requires it. One of the extended connectors add fixed function pins while the other extends the number of user-defined pins. Both extension pins define additional power lines to increase the coupling between system-components. A card is free to implement either or both extension connectors.

Standard connector pin-out

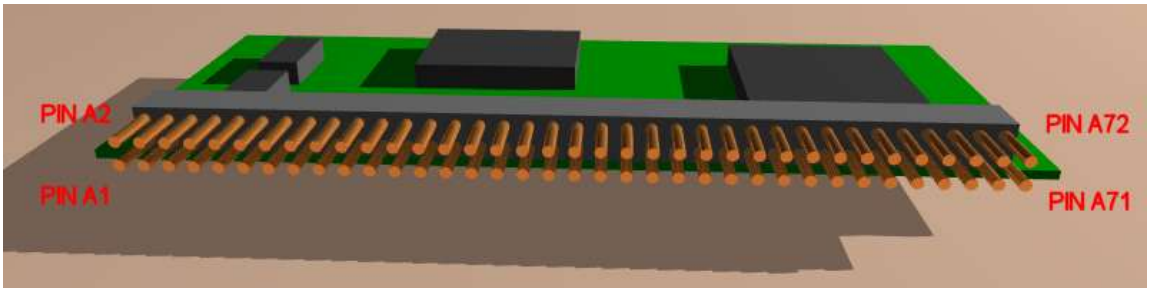

Figure 8: Standard connector

Pin No.	*CPU card Mode*	*Perihperial card Mode*	*Name*	*Description*
A1	O	I	nRD	Active low read select signal.
A2	O	I	nUWE	Active low write select signal. Upper byte.
A3	O	I	nLWE	Active low write select signal. Lower byte.
A4	IOCPU	IOCPU	nRESET	Active low reset signal.
A5	O	I	nSEL0	Active low peripherial select signal.
A6	O	I	nSEL1	Active low peripherial select signal.
A7	O	I	nSEL2	Active low peripherial select signal.
A8	IPU	OC	nIRQ0	Active low interrupt signal. Can be edge or level sensitive.
A9	IPU	OC	nIRQ1	Active low interrupt signal. Can be edge or level sensitive.
A10	IPU	OC	nIRQ2	Active low interrupt signal. Can be edge or level sensitive.
A11	IPU	OC	nWAIT	Active low signal to introduce wait-states in access cycles. Some CPU cards may not support this feature.

Pin No.	***CPU card Mode***	***Perihperial card Mode***	***Name***	***Description***
A12	I	OCPU	nPROG/nBRD_EN	This pin used for programming CPU and Peripherial cards and to inhibit PnP card-isolation. Do not connect for normal operation.
A13	O	I	A0	Address lines
A14	O	I	A1	
A15	O	I	A2	
A16	O	I	A3	
A17	O	I	A4	
A18	O	I	A5	
A19	O	I	A6	
A20	O	I	A7	
A21	O	I	A8	
A22	O	I	A9	
A23	IO	IO	D0	Data lines
A24	IO	IO	D1	
A25	IO	IO	D2	
A26	IO	IO	D3	
A27	IO	IO	D4	
A28	IO	IO	D5	
A29	IO	IO	D6	
A30	IO	IO	D7	
A31	IO	IO	D8	
A32	IO	IO	D9	
A33	IO	IO	D10	
A34	IO	IO	D11	
A35	IO	IO	D12	
A36	IO	IO	D13	
A37	IO	IO	D14	
A38	IO	IO	D15	
A39	PWR	PWR	GND	Ground
A40	PWR	PWR	GND	
A41	PWR	PWR	VCC_3	3.3V nominal power supply.
A42	PWR	PWR	VCC_3	
A43	PWR	PWR	VCC_1_8	1.8V nominal power for core supply.
A44	PWR	PWR	VCC_2_5	2.5V nominal power for core supply.

Pin No.	*CPU card Mode*	*Perihperial card Mode*	*Name*	*Description*
A45	IO	IO	IOA0	Card specific I/O group A
A46	IO	IO	IOA1	
A47	IO	IO	IOA2	
A48	IO	IO	IOA3	
A49	IO	IO	IOA4	
A50	IO	IO	IOA5	
A51	IO	IO	IOB0	Card specific I/O group B
A52	IO	IO	IOB1	
A53	IO	IO	IOB2	
A54	IO	IO	IOB3	
A55	IO	IO	IOB4	
A56	IO	IO	IOB5	
A57	IO	IO	IOC0	Card specific I/O group C
A58	IO	IO	IOC1	
A59	IO	IO	IOC2	
A60	IO	IO	IOC3	
A61	IO	IO	IOC4	
A62	IO	IO	IOC5	
A63	IO	IO	IOD0	Card specific I/O group D
A64	IO	IO	IOD1	
A65	IO	IO	IOD2	
A66	IO	IO	IOD3	
A67	IO	IO	IOD4	
A68	IO	IO	IOD5	
A69	IO	IO	IOD6	
A70	IO	IO	IOD7	
A71	IOPU	IOPU/IO	PNPC	PnP bus clock
A72	IOPU	IOPU/IO	PNPD	PnP bus data

Recommended alternative pin-outs for common peripherial functions

The following alternative pin-outs are shown the group A, but they can be applied to either group A, B or C.

Sync serial interface

Pin No.	*CPU card Mode*	*Perihperial card Mode*	*Name*	*Description*
A45	IO	IO	RCLK	Receive clock signal
A46	IO	IO	RFS	Receive frame signal
A47	I	I	RD	Receive data
A48	IO	IO	TCLK	Transmit clock signal
A49	IO	IO	TFS	Transmit frame signal
A50	O	O	TD	Transmit data

RS-232 interface

Pin No.	*CPU card Mode*	*Perihperial card Mode*	*Name*	*Description*
A45	O	O	DTR	Data Terminal Ready
A46	O	O	RTS	Request To Send
A47	I	I	RXD	Receive Data
A48	I	I	DSR	Data Set Ready
A49	I	I	CTS	Clear To Send
A50	O	O	TXD	Transmit Data

RS-422/RS-425 interface

Pin No.	*CPU card Mode*	*Perihperial card Mode*	*Name*	*Description*
A45				
A46	I	I	RXD+	Receive port
A47	I	I	RXD-	
A48				
A49	O	O	TXD+	Transmit port
A50	O	O	TXD-	

SPI interface

Pin No.	*CPU card Mode*	*Perihperial card Mode*	*Name*	*Description*
A45	IO	IO	CLK	Clock signal
A46	IO	IO	nSS0	First peripherial select signal
A47	I	I	MOSI	Receive data
A48	IO	IO	nSS1	Second peripherial select signal
A49	IO	IO	nSS2	Third peripherial select signal
A50	O	O	MISO	Transmit data

CAN interface

Pin No.	*CPU card Mode*	*Perihperial card Mode*	*Name*	*Description*
A45				
A46	I	I	RXD-/GND	Receive port
A47	I	I	RXD+	
A48				
A49	O	O	TXD-/GND	Transmit port
A50	O	O	TXD+	

USB interface

Pin No.	*CPU card Mode*	*Perihperial card Mode*	*Name*	*Description*
A45	O	O	Host	high: Host; low: Target device
A46	IO	IO	DATA+	USB data lines
A47	IO	IO	DATA-	
A48				
A49				
A50				

Timers

Pin No.	*CPU card Mode*	*Perihperial card Mode*	*Name*	*Description*
A45	IO	I	TC0	Clock input for timer 0
A46	IO	I	TG0	Gate input for timer 0
A47	IO	O	TO0	Output for timer 0
A48	IO	I	TC1	Clock input for timer 1
A49	IO	I	TG1	Gate input for timer 1
A50	IO	O	TO1	Output for timer 1

Extended connector pin-out

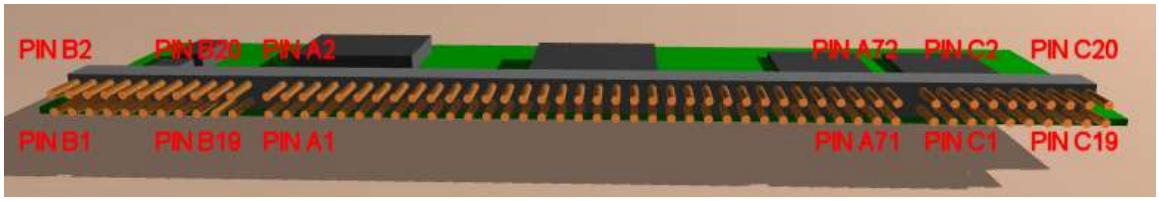

Figure 9: Extended connectors

There are two extension connectors defined. Connector B that goes to the left of the basic connector (by pin 1) is used to add additional address space and DMA capability to the H-Storm bus. The extension connector C that goes to the right of the basic connector (by pin 72) adds more user-defined pins. See mechanical layouts for the exact locations of the extended connectors.

Bus-signal extension connector

Pin No.	*CPU card Mode*	*Perihperial card Mode*	*Name*	*Description*
B1	IPU	OC	nDRQ0	Active low DMA request signal. Can be edge or level sensitive.
B2	IPU	OC	nDRQ1	Active low DMA request signal. Can be edge or level sensitive.
B3	IPU	OC	nDRQ2	Active low DMA request signal. Can be edge or level sensitive.
B4	O	I	nDACK	Active low DMA acknowledge signal. A low on this pin while the corresponding nSELx pin is active is to signal a DMA acknowledge to the selected device
B5	O	IPD	A10	Address lines
B6	O	IPD	A11	
B7	O	IPD	A12	
B8	O	IPD	A13	
B9	O	IPD	A14	
B10	O	IPD	A15	
B11	O	IPD	A16	
B12	O	IPD	A17	
B13	O	IPD	A18	
B14	O	IPD	A19	
B15	O	IPD	A20	
B16	O	IPD	A21	
B17	PWR	PWR	VCC_3	3.3V nominal power supply.
B18	PWR	PWR	VCC_3	3.3V nominal power supply.
B19	PWR	PWR	GND	Ground
B20	PWR	PWR	GND	Ground

Card-specific signal extension connector

Pin No.	*CPU card Mode*	*Perihperial card Mode*	*Name*	*Description*
C1	IO	IO	IOE0	Card specific I/O group E
C2	IO	IO	IOE1	
C3	IO	IO	IOE2	
C4	IO	IO	IOE3	
C5	IO	IO	IOE4	
C6	IO	IO	IOE5	
C7	IO	IO	IOE6	
C8	IO	IO	IOE7	
C9	IO	IO	IOE8	
C10	IO	IO	IOE9	
C11	IO	IO	IOE10	
C12	IO	IO	IOE11	
C13	IO	IO	IOE12	
C14	IO	IO	IOE13	
C15	IO	IO	IOE14	
C16	IO	IO	IOE15	
C17	PWR	PWR	VCC_3	3.3V nominal power supply.
C18	PWR	PWR	VCC_3	3.3V nominal power supply.
C19	PWR	PWR	GND	Ground
C20	PWR	PWR	GND	Ground

Symbols:

I: input
O: output
OC: open-collector output
IPU: input with internal pull-up
IPU: input with internal pull-down
IO: input-output
IOPU: input-output with internal pull-up
IOCPU: input-output with open-collector drive and pull-up
PWR: power pin

- nSELx, nDMAx and nIRQx signals are rotated for each extension card
- Appropriate reset signal must be generated by the CPU-board
- A PnP system-controller and an integrated or external PnP configuration ROM must exist an all extension cards and system-boards. They never exist on CPU cards.
- A CPU card can implement DMA and IRQ lines as edge or level sensitive or both. Peripherial cards can use either type of signaling as well. Since this can cause incompatibilities between CPU and peripherial cards, the type of behavior must be clearly stated for each design. Level-sensitive signals are preferred over edge-sensitive ones.
- Additional power and GND lines are added for each expansion board to tighten the coupling between the system components. A CPU or Peripherial card however can choose not to use those additional signals. If a

system-board supports the extended version of the bus signals, it must provide power over the additional pins.
- Any component of the system can freely implement either or none of the extension connectors (B or C).
- Pull-down resistors of a value of 1M should be added to the additional address lines on any peripherial board that implements extension connector B. This is to make sure that all address lines are valid even if the card is inserted into a system with a standard CPU card.

Bus-cycles

There are sixteen different bus cycles, defined for the H-Storm bus. A cycle can be read or write, with or without wait-states, normal or burst and DMA or CPU bus-cycles. The nDACK line which differentiates CPU and DMA cycles has the same timing as the address lines, so no additional timing data is required. For this reason only eight timing diagram is provided. Extended connector address lines have the same timing as standard-connector address lines. On the following diagrams only standard address lines are shown.
On the following diagrams many of the signals are referenced to the signal, named nCYC. This is not a signal of the H-Storm bus. It is a logical signal, which is defined as follows: nCYC := nSEL or (nRD and nLWE and nUWE). Simply put it designates the active part of the cycle, i.e. when nSEL is active as well as either of the cycle qualifiers (nRD, nLWE or nUWE).

Normal read cycle without external wait-states

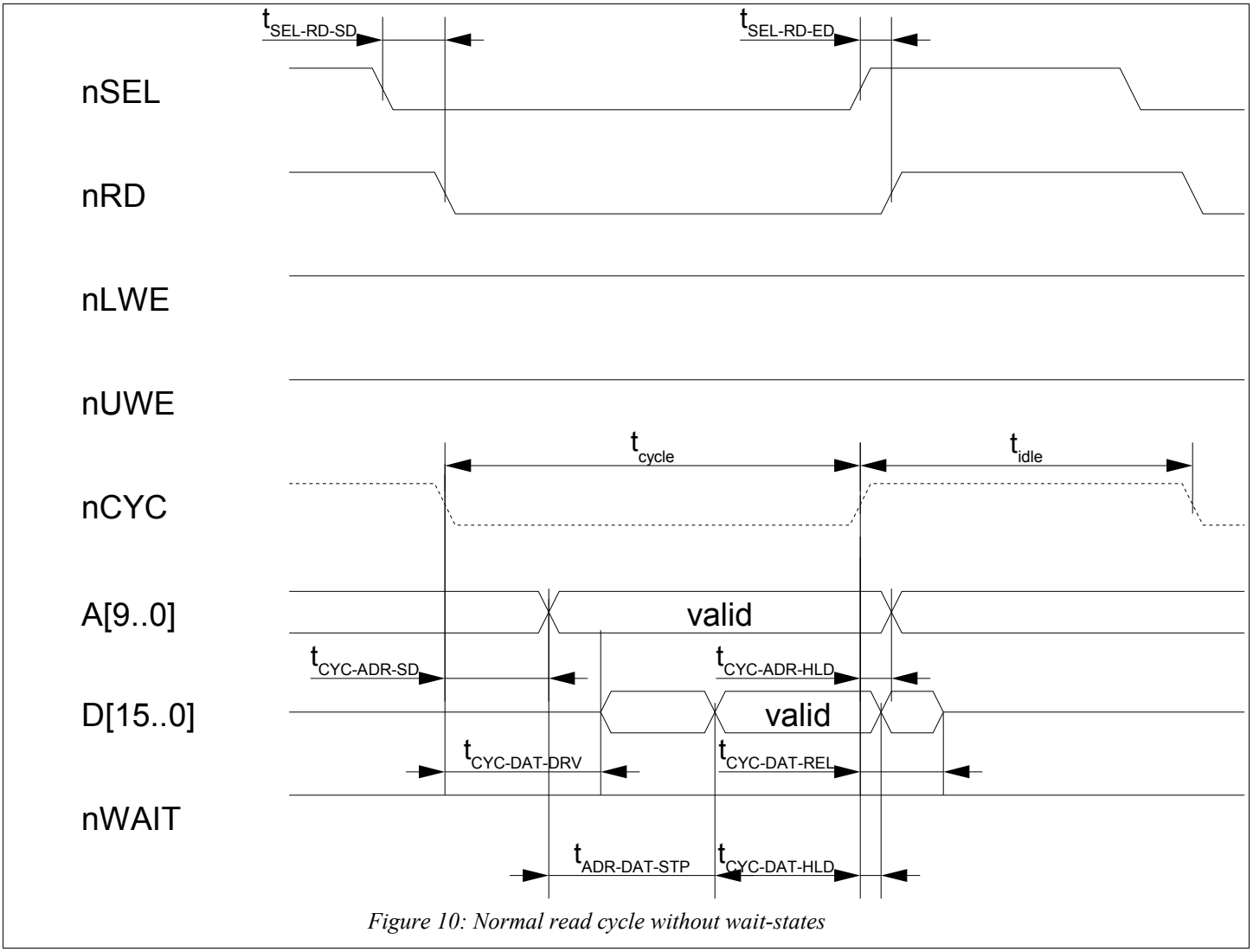

Figure 10: Normal read cycle without wait-states

t_{cycle}: width of the read access cycle
t_{idle}: time between two consecutive accesses
$t_{SEL-RD-STP}$: setup delay between the assert of nSELx and nRD (generally shouldn't matter)
$t_{SEL-RD-ED}$: end delay between the de-assert of nSELx and nRD (generally shouldn't matter)
$t_{CYC-ADR-SD}$: setup delay between an active cycle-start and the valid address is presented on the address-bus
$t_{CYC-ADR-HLD}$: hold time between an active cycle-end and the valid address is removed from the address-bus
$t_{CYC-DAT-DRV}$: time between the start of an active cycle and the bus-drivers are enabled on the peripherial card
$t_{ADR-DAT-STP}$: time between a valid address is presented on the bus and the valid data is presented on the bus. If the address is valid, before nCYC goes low ($t_{CYC-ADR-SD}$ is negative) this time is measured from the falling edge of nCYC till the appearance of the valid address.
$t_{CYC-DAT-HLD}$: time between the end of an active cycle and the valid data is removed from the bus
$t_{CYC-DAT-REL}$: time between the end of an active cycle and the bus-drivers are disabled on the peripherial card

Normal read cycle with external wait-states

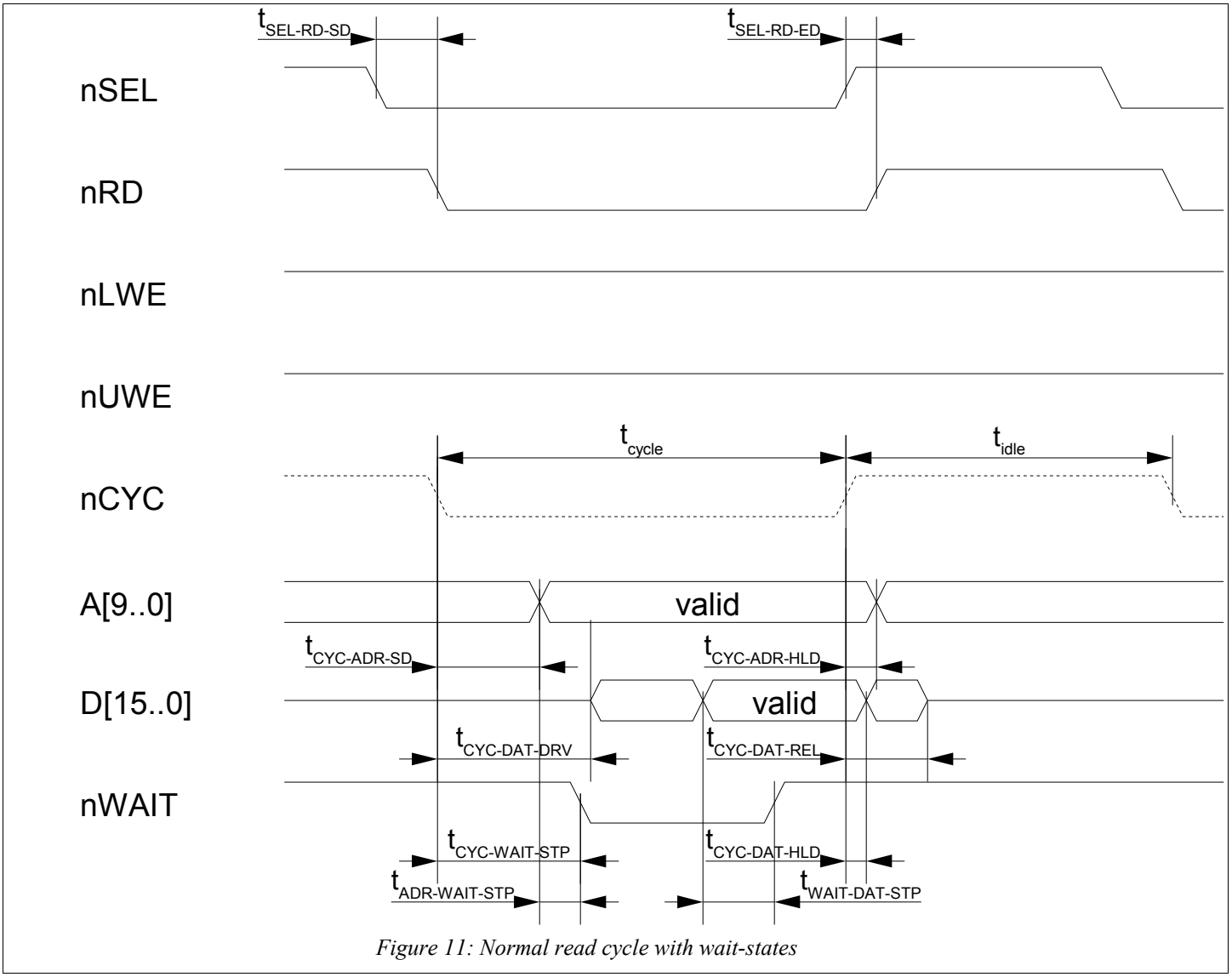

Figure 11: Normal read cycle with wait-states

t_{cycle}: width of the read access cycle
t_{idle}: time between two consecutive accesses
$t_{SEL-RD-STP}$: setup delay between the assert of nSELx and nRD (generally shouldn't matter)
$t_{SEL-RD-ED}$: end delay between the de-assert of nSELx and nRD (generally shouldn't matter)
$t_{CYC-ADR-SD}$: setup delay between an active cycle-start and the valid address is presented on the address-bus
$t_{CYC-ADR-HLD}$: hold time between an active cycle-end and the valid address is removed from the address-bus
$t_{CYC-DAT-DRV}$: time between the start of an active cycle and the bus-drivers are enabled on the peripherial card
$t_{WAIT-DAT-STP}$: time between a valid address is presented on the bus and the valid data is presented on the bus
$t_{CYC-DAT-HLD}$: time between the end of an active cycle and the valid data is removed from the bus
$t_{CYC-DAT-REL}$: time between the end of an active cycle and the bus-drivers are disabled on the peripherial card
$t_{CYC-WAIT-STP}$: time between the start of an active cycle and the assertion of the nWAIT signal.
$t_{ADR-WAIT-STP}$: time between a valid address is presented on the address lines and the assertion of the nWAIT signal.

Normal write cycle without external wait-states

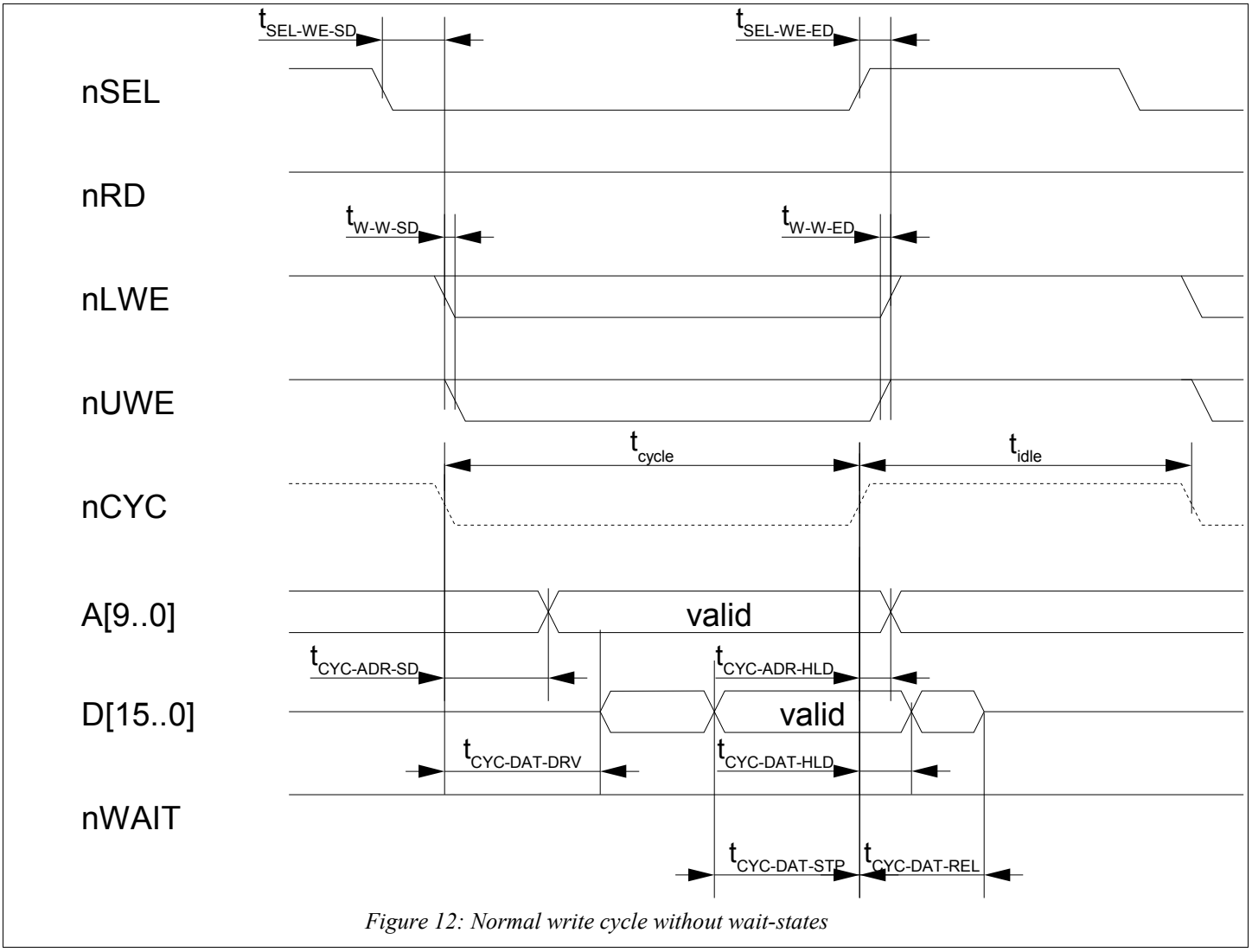

Figure 12: Normal write cycle without wait-states

t_{cycle}: width of the read access cycle
t_{idle}: time between two consecutive accesses
$t_{SEL-WE-STP}$: setup delay between the assert of nSELx and nxWE (generally shouldn't matter)
$t_{SEL-WE-ED}$: end delay between the de-assert of nSELx and nxWE (generally shouldn't matter)
t_{W-W-SD}: setup delay between the two write-enable signals (nLWE and nUWE) in a 16-bit access
t_{W-W-ED}: end delay between the two write-enable signals (nLWE and nUWE) in a 16-bit access
$t_{CYC-ADR-SD}$: setup delay between an active cycle-start and the valid address is presented on the address-bus
$t_{CYC-ADR-HLD}$: hold time between an active cycle-end and the valid address is removed from the address-bus
$t_{CYC-DAT-DRV}$: time between the start of an active cycle and the bus-drivers are enabled on the peripherial card
$t_{CYC-DAT-STP}$: setup time between the end of an active cycle and a valid data is presented on the address-bus
$t_{CYC-DAT-HLD}$: time between the end of an active cycle and the valid data is removed from the bus
$t_{CYC-DAT-REL}$: time between the end of an active cycle and the bus-drivers are disabled on the peripherial card

Normal write cycle with external wait-states

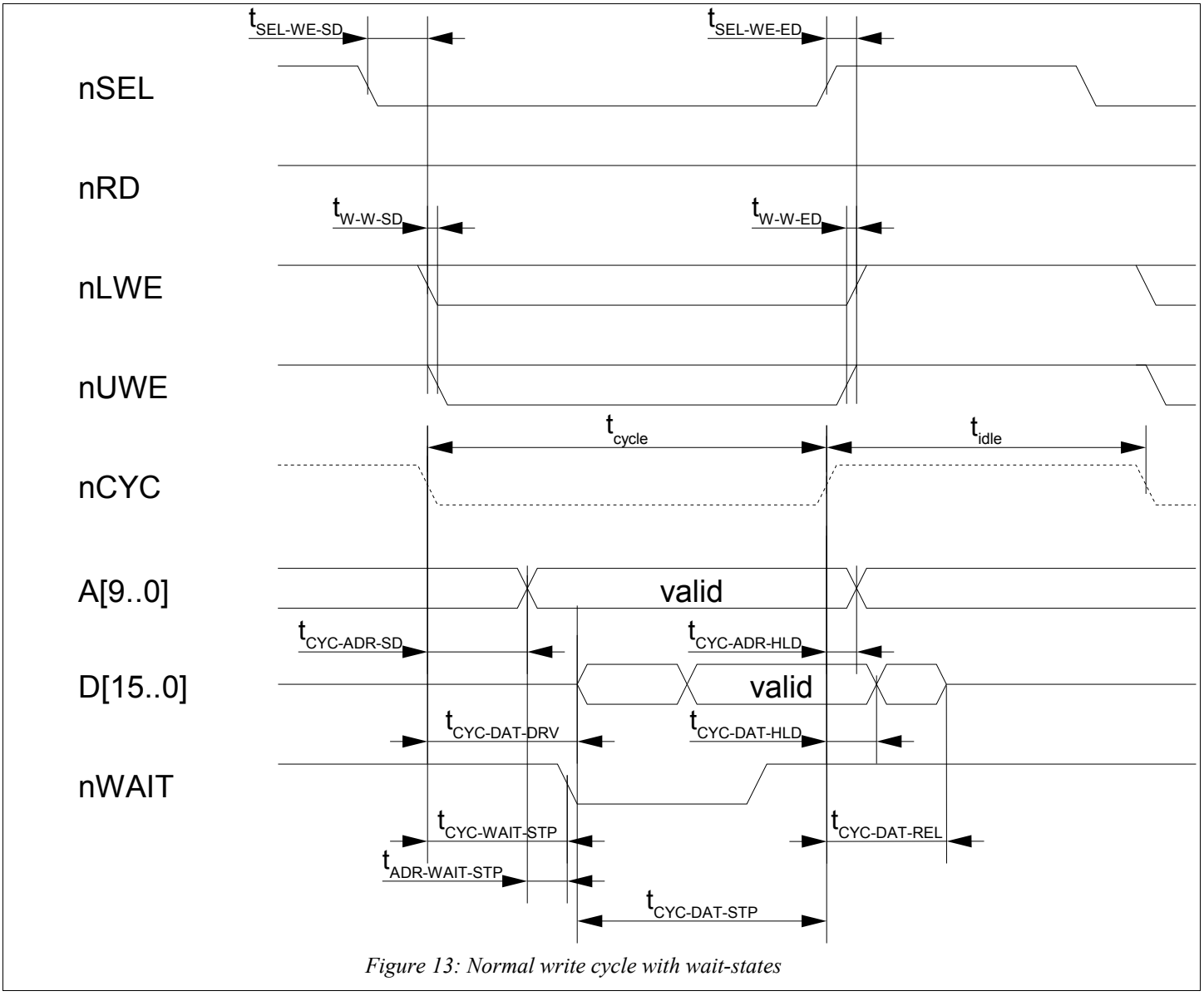

Figure 13: Normal write cycle with wait-states

t_{cycle}: width of the read access cycle
t_{idle}: time between two consecutive accesses
$t_{SEL-WE-STP}$: setup delay between the assert of nSELx and nxWE (generally shouldn't matter)
$t_{SEL-WE-ED}$: end delay between the de-assert of nSELx and nxWE (generally shouldn't matter)
t_{W-W-SD}: setup delay between the two write-enable signals (nLWE and nUWE) in a 16-bit access
t_{W-W-ED}: end delay between the two write-enable signals (nLWE and nUWE) in a 16-bit access
$t_{CYC-ADR-SD}$: setup delay between an active cycle-start and the valid address is presented on the address-bus
$t_{CYC-ADR-HLD}$: hold time between an active cycle-end and the valid address is removed from the address-bus
$t_{CYC-DAT-DRV}$: time between the start of an active cycle and the bus-drivers are enabled on the peripherial card
$t_{CYC-DAT-STP}$: setup time between the end of an active cycle and a valid data is presented on the address-bus
$t_{CYC-DAT-HLD}$: time between the end of an active cycle and the valid data is removed from the bus
$t_{CYC-DAT-REL}$: time between the end of an active cycle and the bus-drivers are disabled on the peripherial card
$t_{CYC-WAIT-STP}$: time between the start of an active cycle and the assertion of the nWAIT signal.

$t_{ADR-WAIT-STP}$: time between a valid address is presented on the address lines and the assertion of the nWAIT signal.

Burst read cycle without external wait-states

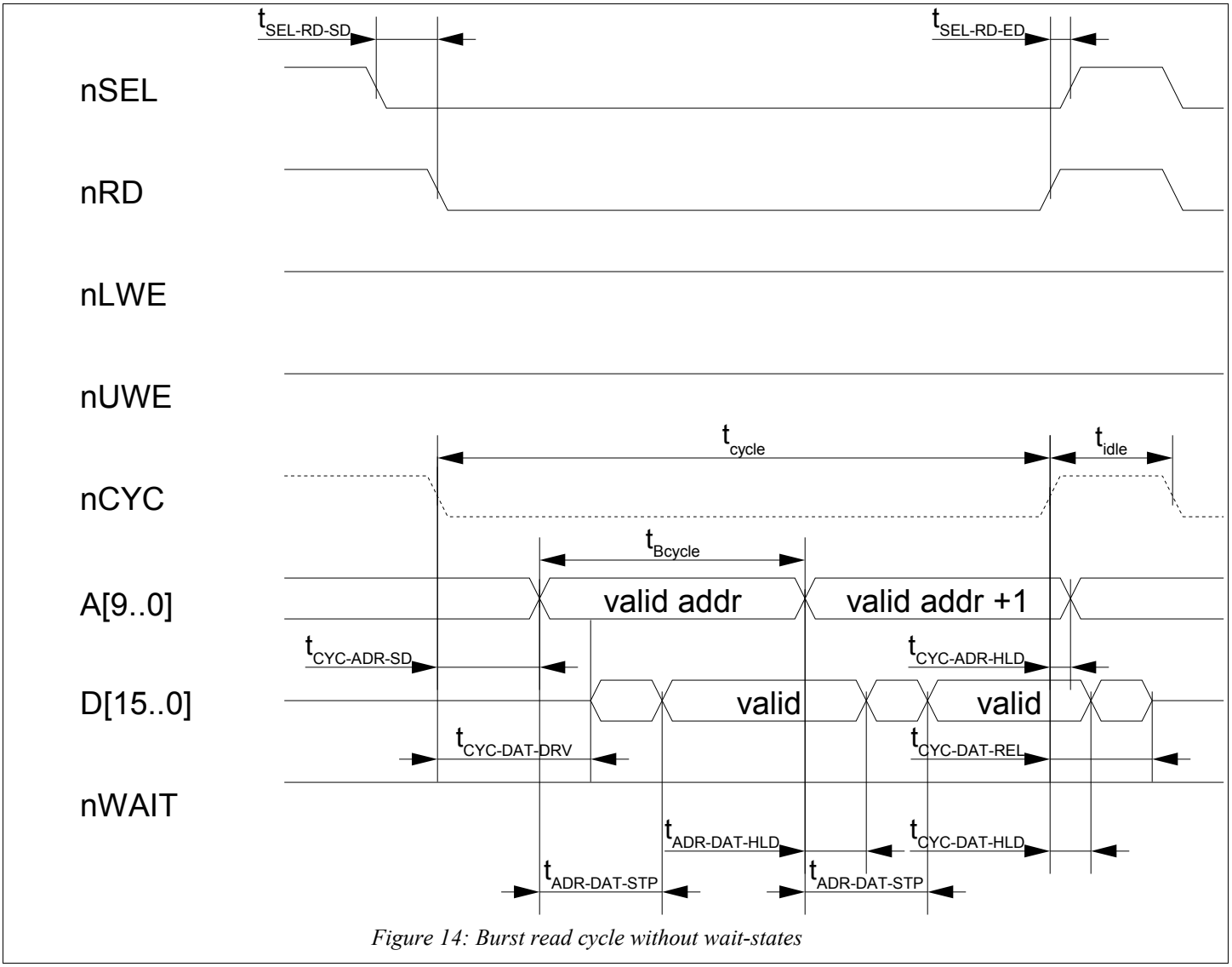

Figure 14: Burst read cycle without wait-states

t_{cycle}: width of the read access cycle
t_{idle}: time between two consecutive accesses
$t_{SEL-RD-STP}$: setup delay between the assert of nSELx and nRD (generally shouldn't matter)
$t_{SEL-RD-ED}$: end delay between the de-assert of nSELx and nRD (generally shouldn't matter)
$t_{CYC-ADR-SD}$: setup delay between an active cycle-start and the valid address is presented on the address-bus
$t_{CYC-ADR-HLD}$: hold time between an active cycle-end and the valid address is removed from the address-bus
$t_{CYC-DAT-DRV}$: time between the start of an active cycle and the bus-drivers are enabled on the peripherial card
$t_{ADR-DAT-STP}$: time between a valid address is presented on the bus and the valid data is presented on the bus If the address is valid, before nCYC goes low ($t_{CYC-ADR-SD}$ is negative) this time is measured from the falling edge of nCYC till the appearance of the valid address.
$t_{CYC-DAT-HLD}$: time between the end of an active cycle and the valid data is removed from the bus
$t_{CYC-DAT-REL}$: time between the end of an active cycle and the bus-drivers are disabled on the peripherial card
t_{Bcycle}: cycle time of each read in the burst

Burst read cycle with external wait-states

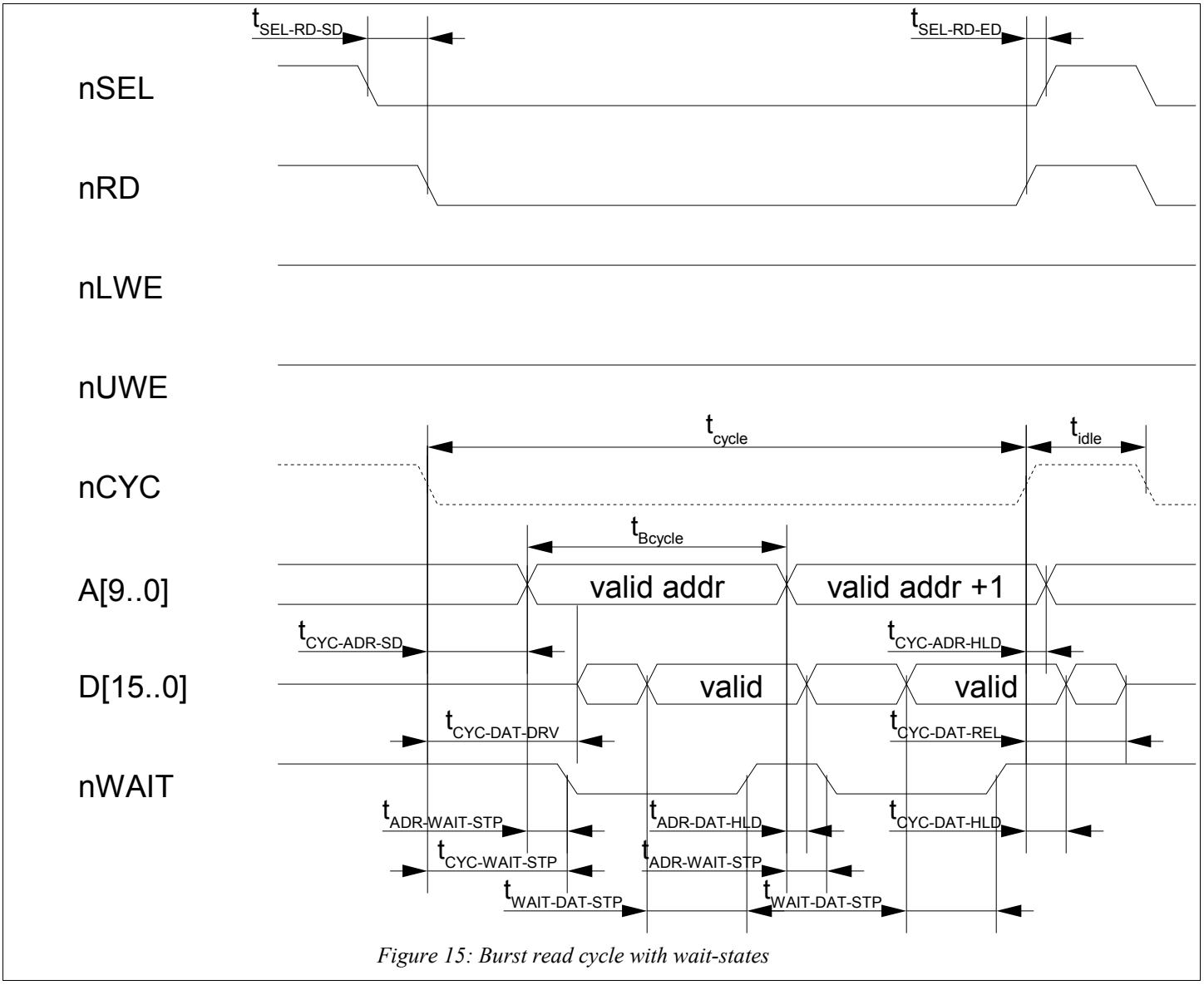

Figure 15: Burst read cycle with wait-states

t_{cycle}: width of the read access cycle
t_{idle}: time between two consecutive accesses
$t_{SEL-RD-STP}$: setup delay between the assert of nSELx and nRD (generally shouldn't matter)
$t_{SEL-RD-ED}$: end delay between the de-assert of nSELx and nRD (generally shouldn't matter)
$t_{CYC-ADR-SD}$: setup delay between an active cycle-start and the valid address is presented on the address-bus
$t_{CYC-ADR-HLD}$: hold time between an active cycle-end and the valid address is removed from the address-bus
$t_{CYC-DAT-DRV}$: time between the start of an active cycle and the bus-drivers are enabled on the peripherial card
$t_{WAIT-DAT-STP}$: time between a valid address is presented on the bus and the valid data is presented on the bus
$t_{CYC-DAT-HLD}$: time between the end of an active cycle and the valid data is removed from the bus
$t_{CYC-DAT-REL}$: time between the end of an active cycle and the bus-drivers are disabled on the peripherial card
$t_{CYC-WAIT-STP}$: time between the start of an active cycle and the assertion of the nWAIT signal.
$t_{ADR-WAIT-STP}$: time between a valid address is presented on the address lines and the assertion of the nWAIT signal.

t_{Bcycle}: cycle time of each read in the burst
$t_{ADR\text{-}DAT\text{-}HLD}$: time between the change of the address on the address lines and the removal of the valid data from the data lines.

Burst write cycle without external wait-states

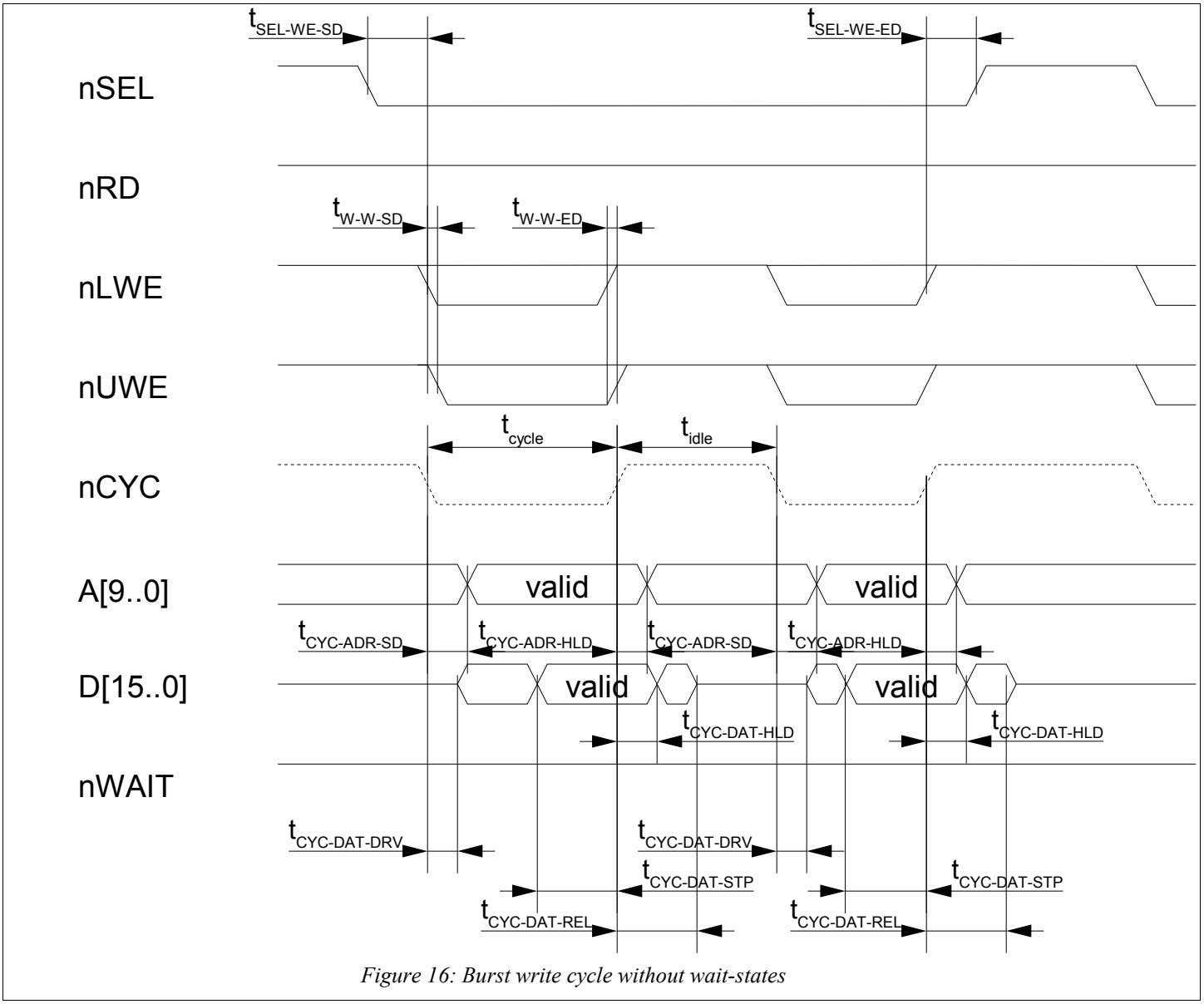

Figure 16: Burst write cycle without wait-states

t_{cycle}: width of the read access cycle
t_{idle}: time between two consecutive accesses
$t_{SEL-WE-STP}$: setup delay between the assert of nSELx and nxWE (generally shouldn't matter)
$t_{SEL-WE-ED}$: end delay between the de-assert of nSELx and nxWE (generally shouldn't matter)
t_{W-W-SD}: setup delay between the two write-enable signals (nLWE and nUWE) in a 16-bit access
t_{W-W-ED}: end delay between the two write-enable signals (nLWE and nUWE) in a 16-bit access
$t_{CYC-ADR-SD}$: setup delay between an active cycle-start and the valid address is presented on the address-bus
$t_{CYC-ADR-HLD}$: hold time between an active cycle-end and the valid address is removed from the address-bus
$t_{CYC-DAT-DRV}$: time between the start of an active cycle and the bus-drivers are enabled on the peripherial card
$t_{CYC-DAT-STP}$: setup time between the end of an active cycle and a valid data is presented on the address-bus
$t_{CYC-DAT-HLD}$: time between the end of an active cycle and the valid data is removed from the bus
$t_{CYC-DAT-REL}$: time between the end of an active cycle and the bus-drivers are disabled on the peripherial card

Burst write cycle with external wait-states

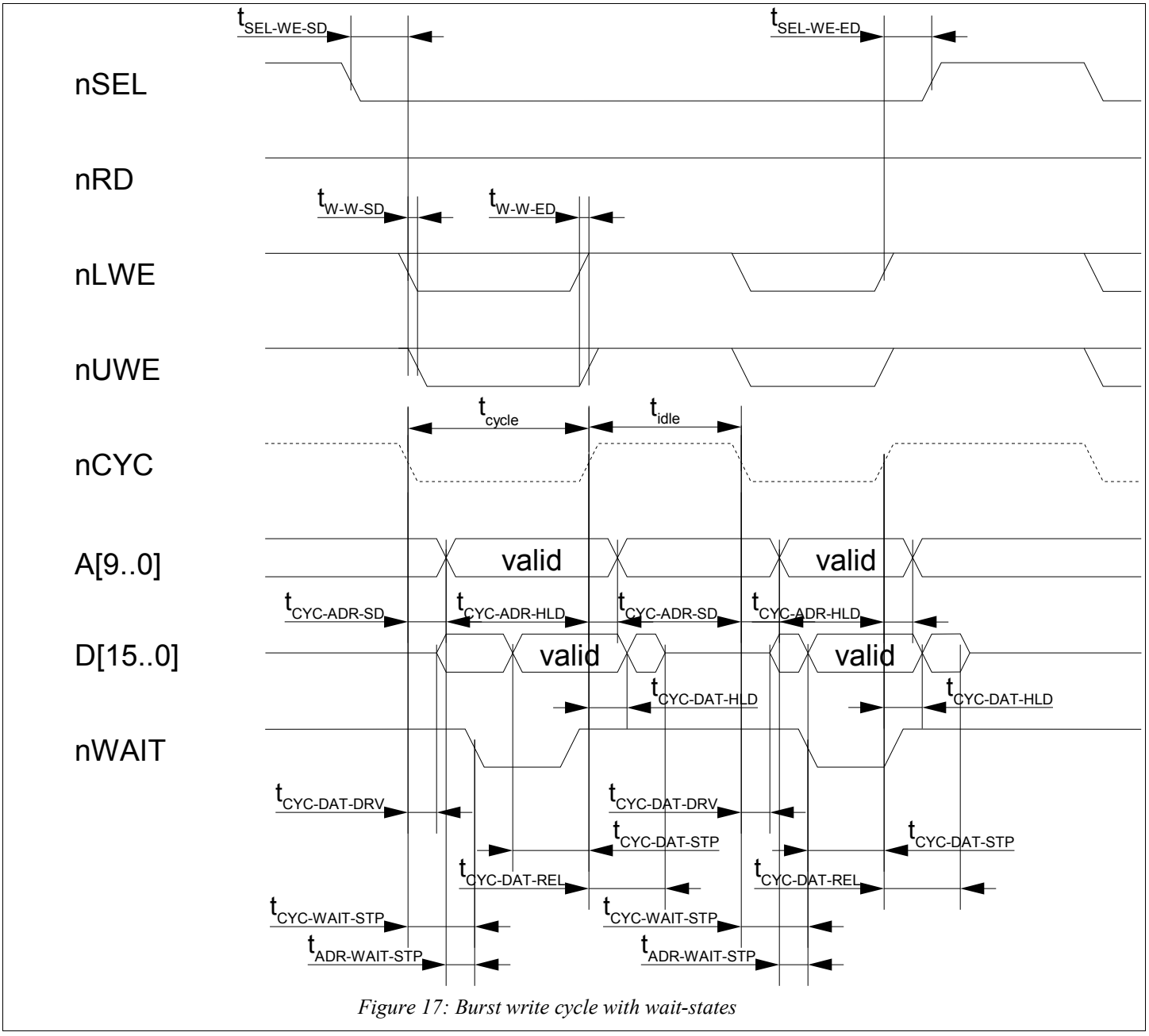

Figure 17: Burst write cycle with wait-states

t_{cycle}: width of the read access cycle
t_{idle}: time between two consecutive accesses
$t_{SEL-WE-STP}$: setup delay between the assert of nSELx and nxWE (generally shouldn't matter)
$t_{SEL-WE-ED}$: end delay between the de-assert of nSELx and nxWE (generally shouldn't matter)
t_{W-W-SD}: setup delay between the two write-enable signals (nLWE and nUWE) in a 16-bit access
t_{W-W-ED}: end delay between the two write-enable signals (nLWE and nUWE) in a 16-bit access
$t_{CYC-ADR-SD}$: setup delay between an active cycle-start and the valid address is presented on the address-bus
$t_{CYC-ADR-HLD}$: hold time between an active cycle-end and the valid address is removed from the address-bus
$t_{CYC-DAT-DRV}$: time between the start of an active cycle and the bus-drivers are enabled on the peripherial card
$t_{CYC-DAT-STP}$: setup time between the end of an active cycle and a valid data is presented on the address-bus

$t_{CYC-DAT-HLD}$: time between the end of an active cycle and the valid data is removed from the bus
$t_{CYC-DAT-REL}$: time between the end of an active cycle and the bus-drivers are disabled on the peripherial card
$t_{CYC-WAIT-STP}$: time between the start of an active cycle and the assertion of the nWAIT signal.
$t_{ADR-WAIT-STP}$: time between a valid address is presented on the address lines and the assertion of the nWAIT signal.

Mechanical Design

Standard CPU/Peripherial card layout

Standard CPU and peripherial cards are 100mm wide circuits boards. Their height can vary from card to card, but an around 40mm heights is recommended. A 2mm clearance from each edge of the board is also recommended. In that area no electrical signal should be routed. This makes manufacturing of the PCB easier. Components can overhang to that 2mm clearance zone as long as all of their electrical contacts are inside that area. This clearance however is dependent on the manufacturing process and must be checked with the manufacturer. The connector is a standard dual-row 100mil 72-pin, right-angle, header with its Pin1 positioned at 5mm from the left- and 4.5 mm from the bottom side of the circuit board. The board can contain contain components of 15mm high on its top side. On the bottom side there could be no component that's higher than 2.5mm. The circuit board can be no thicker than 1.8mm, but a standard 1.5mm (0.062”) thickness is recommended.

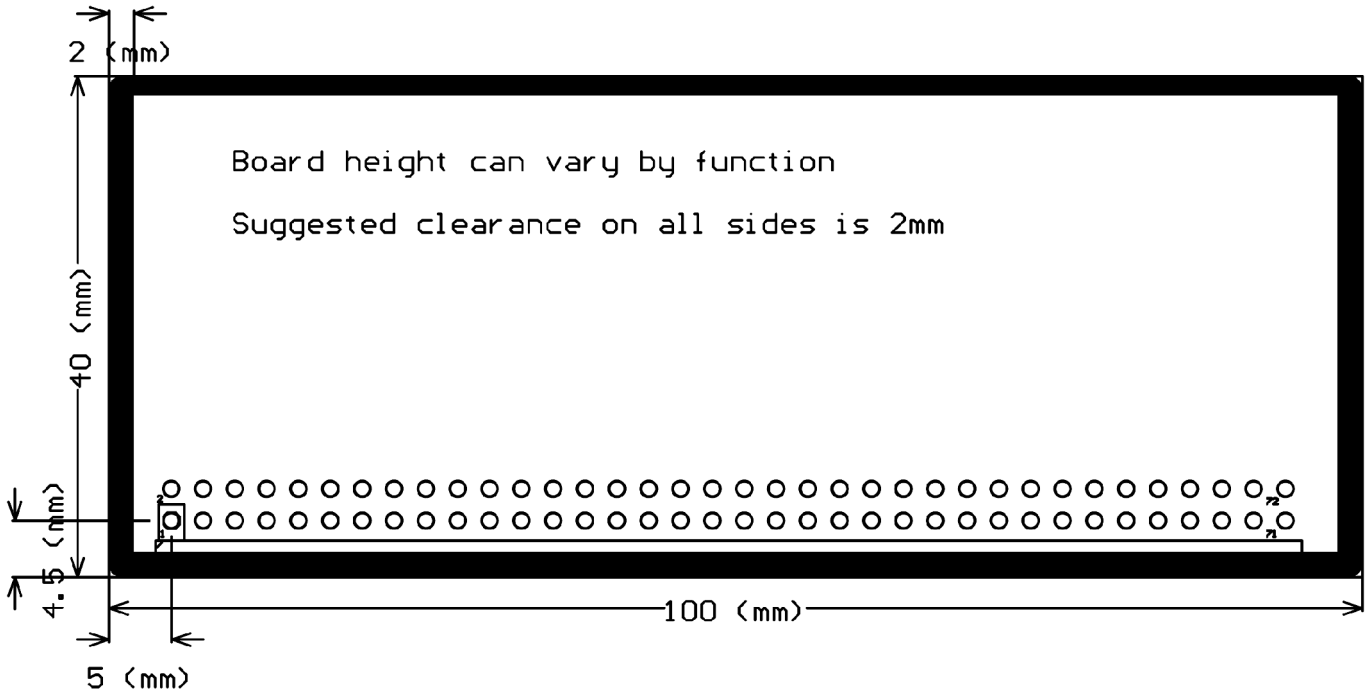

Figure 18: Standard CPU/Peripherial card layout

Extended CPU/Peripherial card layout

Extended CPU and peripherial cards are 160mm wide. Their height can vary from card to card just as the standard version, with a recommended heights of 50mm. A 2mm clearance from each edge of the board is also recommended. In that area no electrical signal should be routed. This makes manufacturing of the PCB easier.

Components can overhang to that 2mm clearance zone as long as all of their electrical contacts are inside that area. This clearance however is dependent on the manufacturing process and must be checked with the manufacturer. There are three connectors on each extended board. Apart from the standard 72 pin A connector, two 20-pin extended connectors are also used. (B and C connectors). All connectors are standard dual-row 100mil, right-angle, headers. The standard (A) connectors Pin1 positioned at 35mm from the left- and 4.5 mm from the bottom side of the circuit board. The extended connectors are placed to the left and right of the standard connector with 5.08 (200mil) hole center-to-center distance. This aligns all connector pins on a single 100mil to make the modules easy to use on a prototyping board. The board can contain contain components of 15mm high on its top side. On the bottom side there could be no component that's higher than 2.5mm. The circuit board can be no thicker than 1.8mm, but a standard 1.5mm (0.062”) thickness is recommended.

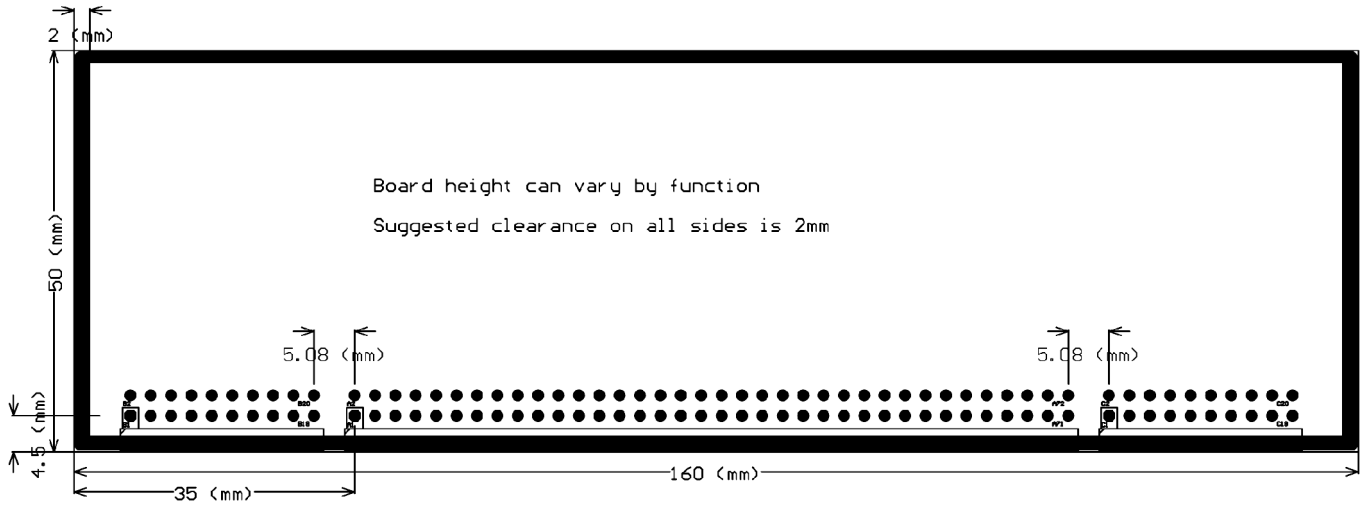

Figure 19: Extended CPU/Peripherial card layout

Standard System-board layout

The dimensions of system-boards are not specified in an H-Storm system. Every implementation can choose the most desirable layout. However CPU and Peripherial connectors should be laid out in a way that those cards can actually be inserted into the slots. Enough clearance should be kept on all sides of the connectors. Inside this low-clearance zone no component higher than the H-Storm connector can be placed. The inserted card can occupy a 100mm x 19mm area around the socket. Between the sockets a 20mm clearance should therefore be left to prevent adjacent cards from touching each other. The low-clearance zone should be 1mm wider in all direction than this minimum requirement to allow for manufacturing imperfections. The connectors should be standard 100mil dual-line 72-pin, straight female header connectors.

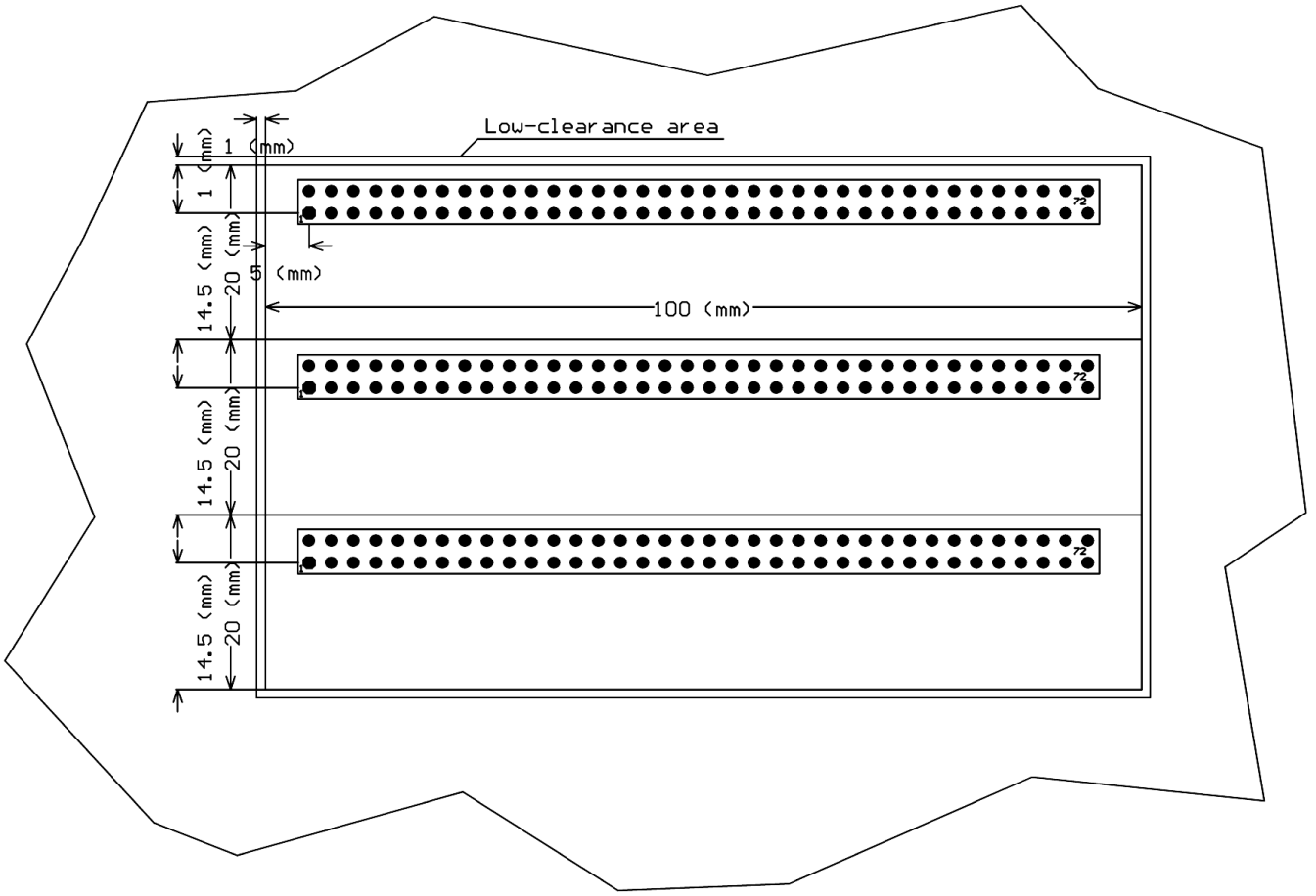

Figure 20: Standard System-board layout

Extended System-board layout

The dimensions of system-boards are not specified in an H-Storm system. Every implementation can choose the most desirable layout. However CPU and Peripherial connectors should be laid out in a way that those cards can actually be inserted into the slots. Enough clearance should be kept on all sides of the connectors. Inside this low-clearance zone no component higher than the H-Storm connector can be placed. The inserted card can occupy a 160mm x 19mm area around the socket. Between the sockets a 20mm clearance should therefore be left to prevent adjacent cards from touching each other. The low-clearance zone should be 1mm wider in all direction than this minimum requirement to allow for manufacturing imperfections. The connectors should be standard 100mil dual-line, straight female header connectors. Apart from the standard 72-pin connector (A) there are two 20-pin connectors as well (B and C). These connectors are placed 5.08mm (200mil) hole center-to-center distance from the standard connector (A) just as on the extended CPU and Peripherial cards.

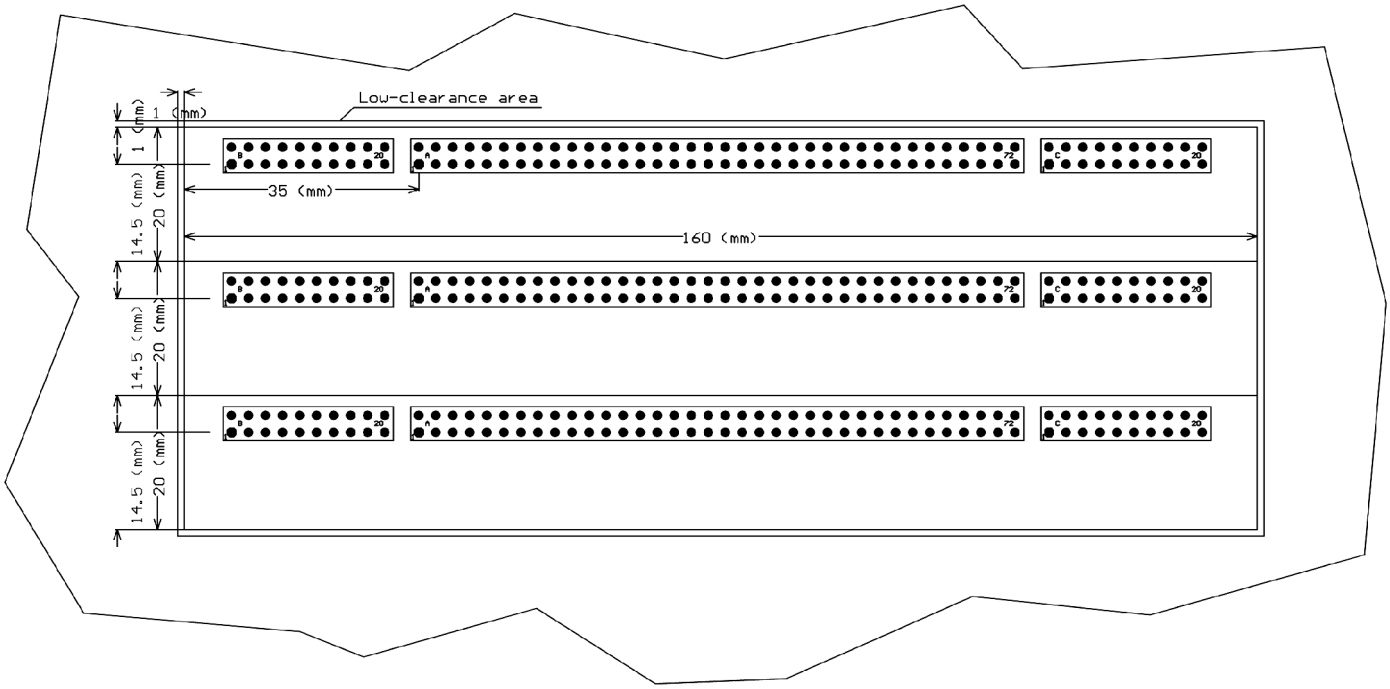

Figure 21: Extended System-board layout

PnP bus operation

UPDATE: This information is obsolete and will be revised in the next revision of the documentation!

The PnP-bus on H-Storm uses the standard two-wire interface (also known as I²C™ bus).

Low-level signaling over the PnP-bus

This bus contains two signals, a clock (PnPC) and a data (PnPD). These signals are driven by open-drain drivers and are pulled up on the CPU card.
In the idle state of the bus both the clock and the data lines are at high level. A transaction over the bus is started by pulling the data line low, while keeping the clock line high. This is called the start condition or start-bit. The device that generated the start-condition on the bus will be the master for the duration of the transaction. Any other device that takes a part in the communication is called a slave device. After the start-bit the transaction continues in 8-bit chunks with one acknowledge bit between each 8-bit data. The direction of the communication can change at each byte-boundary. The clock is always supplied by the master, the data-bits are provided by either the master or the slave(s). The acknowledge bit is always sent by the other party, i.e. If the slave provided the data bits, the acknowledge comes from the master, and if the data bits are provided by the master, the acknowledge is generated by the salve. During the transaction the data line is allowed to change when the clock line is low. The data-line has to be stable while the clock-line is high. The end of the transaction is signaled by pulling the data line high while the clock line is high.
Note that the start and stop conditions are signaled by an edge on the data-line, while the clock signal is high: a falling edge signifies a start condition, while a rising edge encodes the stop-condition. Data-bits never change the state of the data-line while the clock signal is high.
Acknowledge bits are low level for if the device acknowledges the transfer, and high if its not.

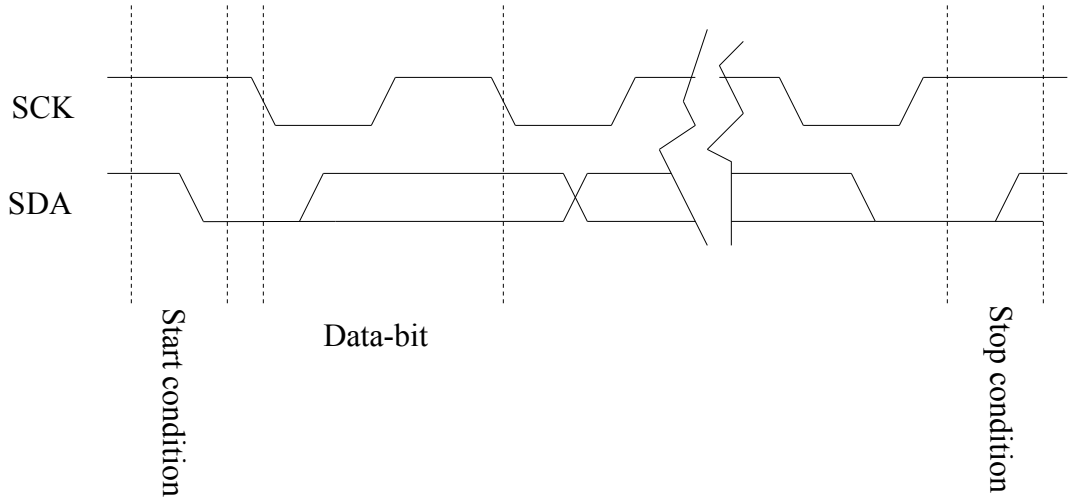

Figure 22: Two-wire interface bit-level signaling

If a stop condition is detected on the line, the current transaction has to be terminated, no matter how many data or acknowledge bits have been transmitted and at what point the communication is. All devices (both masters and slaves) should return to their idle state, however some internal state-changes (like data already written into some memory) does not required to be discarded.
If a start condition is detected without a stop condition preceding it (called are-start condition), a stop-condition is implied. All devices should treat the re-start conditions as if the previous transaction has been completed and a new one has been started.
The maximum speed of the clock signal is 100kHz.
Bytes on the wire transmitted with MSB first bit-ordering.
Each slave device on the PnP bus has at least one physical address. For each transaction the first transmitted byte contains the slave-address. If a slave device has multiple slave addresses, the type of operation can be encoded in the first byte of the transaction as well. (It's typical to use two physical addresses, which differ only in the LSB (bit-0). The two physical addresses are then used to distinguish read and write operations.) A slave device answers to a transaction start with an acknowledge only if it sees one if it's physical addresses on the bus as the first byte of the transaction.
The rest of the transaction both length-wise and direction-wise depend on the particular slave device that's being addressed. This means that the physical addresses has to be preassigned to slave functions and known by the system software.

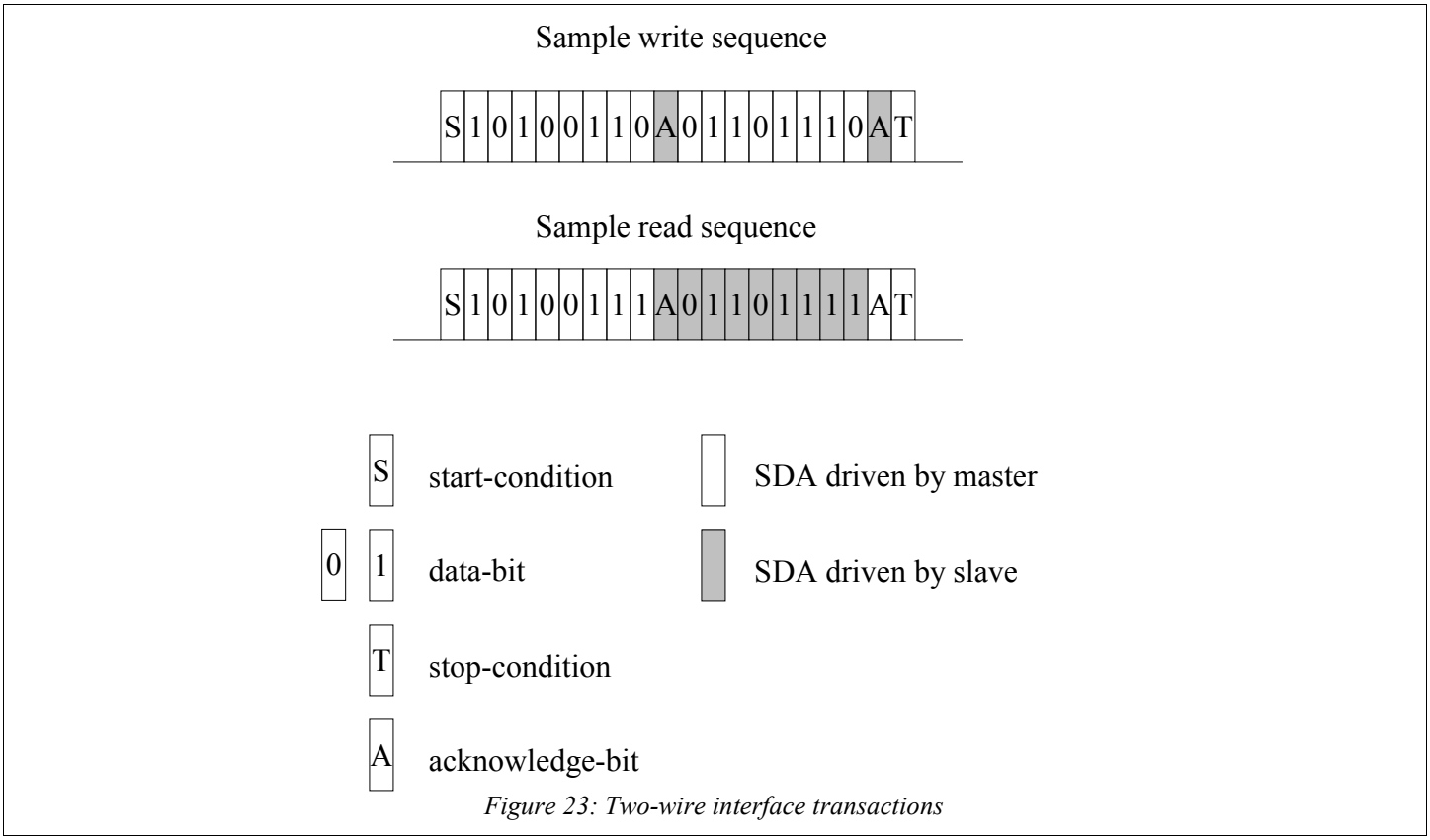

Figure 23: Two-wire interface transactions

The two-wire interface has a neat bus-arbitration mechanism that allows multiple devices contending for communication to work out the ownership of the bus. This mechanism originally was developed to allow multi-master operation over the bus, but it can also be used for soft-addressing as well. This technique is similar to the one used in the 21LCS61/62 devices from Microchip. The process exploits the fact that the data-lines in the two-wire interface are driven by open-drain drivers. This means that if multiple sources are enabled, the signal level will be the logical AND of all the enabled drivers. I.e. any driver can pull the line low. So, each driver tries to transmit it's information but it monitors the line as well constantly. As long as the wire contains the same level as the driver tried to transmit, no collision occurred and the transfer can continue. Once the signal-level is different, it shows that another device also tries to drive the wire. At that point the device that detected the mismatch decide that it has lost the battle for the ownership of the bus, disconnects and stops sending information. Note that this condition can only happen if the device tries to send a logical '1' while another device tries to send a logical '0'. The one that sends the '0' will win and the one that tries to send the '1' will disconnect. As long as all devices send the same data, the fact that multiple devices access the bus cannot be detected but it's not important: if they try to transmit the same information, it doesn't matter who's driver put that information on the wire.

Differences between the I²C™ bus and the H-Storm PnP bus

While the signaling and the implementation of the two buses are the same, there are some particularities of the H-Storm implementation.

1. The H-Storm bus doesn't support multi-master operation: the only master on the bus can be the CPU card
2. The H-Storm bus doesn't generally support clock-stretching. Clock-stretching might be used by the system-controllers on the peripherial boards but not by other devices. The bus-master (the CPU card) however must support clock-stretching.

3. The H-Storm bus does not require the LSB bit of the first byte to identify the direction of the transfer: A device that uses that convention would have two physical addresses in the H-Storm PnP bus notation. One for reads and one for writes.

The System-controller and the PnP operation

The PnP bus of the H-Storm system provides rich information for the operation software about the system it operates in. On the top of that this interface gives the option to the software to isolate any of the peripherial cards or the system-board from the system-bus. This functionality can be used for many reasons but the main purpose is to help the PnP discovery of the system during startup.
The PnP discovery has two main goals to achieve. To find out what peripherials and system-board the CPU card is connected to, and how to access those devices.
On the highest level, the discovery of the components of the system is done by scanning the PnP bus for devices. After a component is identified, its configuration can be retrieved from the PnP configuration ROM. Using this configuration the CPU card can determine if it's possible for it to communicate with the component, and by comparing the configuration of each component of the system it can detect resource conflicts.
The design of the PnP system must support similar or identical devices in the system. It also have to make it possible to completely discover the system using only the two wires of the PnP bus. There is a problem though: how can we individually address each device on the PnP bus, even if they are identical?
There are two possible solutions to the problem: one would be to make every device different, by assigning a unique ID to them at manufacturing or bring-up, similar to the way each Ethernet card on the world have a unique MAC address. The other solution is to provide some means to individually enable the PnP bus on each peripherial board and on the system-board. This functionality has to be implemented on the system-board though and has to use special and hard-to-get I2C bus multiplexers. In the end both solutions would switch bus connections on and off and isolate parts of the system from one another, but the first solution uses distributed resources to do this, while the second relies on a central component on the system-board to operate.
One of the design goals of H-Storm was to keep the system-board as simple as possible. For this reason I voted for the first option. This means, that each peripherial card has to contain a unique ID that can be used to access that particular board, using only the resources of the PnP bus. This functionality and some other as well is implemented by the H-Storm system-controller.

The system-controller

The system-controller is physically a small 8-pin device with following features:

- It is the only device directly connected to the PnP bus. This two-wire interface is called the host-side interface.
- It's physical address is 0x65.
- It has a secondary (board-side) two-wire interface. All devices on the peripherial board that communicate over the PnP bus are connected to this two-wire interface.
- It has an active-low open-drain signal that is used to enable system-bus communication with the peripherial card. If this signal is high, the peripherial card is isolated from the system-bus. This signal is connected to the nBRD_EN pin of the H-Storm connector.
- It contains an ID that is guarantied to uniquely identify the card
- It implements the H-Storm PnP system configuration protocol.
- It can be assigned an 5-bit soft-address and be addressed with that afterwords. It is possible to have up to 30 system-controllers on the same PnP bus.

The block-diagram of the system-controller is the following:

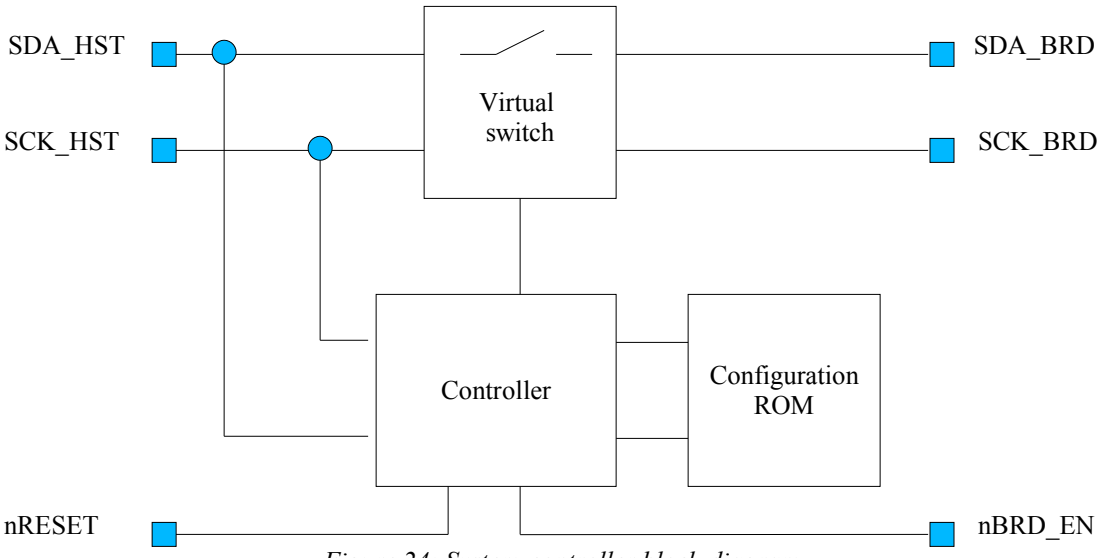

Figure 24: System-controller block-diagram

Usually the system controllers host-side interface is connected to the two-wire interface on the connector of the peripherial card, the board-side interface to any on-board two-wire interface devices. The nRESET signal is connected to the pin of the same name on the H-Storm bus connector,so as the nBRD_EN pin. That pin also controls the behavior of the address-decoder on the board: if the signal is high, the address-decoder is disabled, and no transactions on the system-bus are recognized. The board in isolated from the rest of the system. If the signal is low, the address decoder resumes its normal operation and the board answers transactions directed to it.

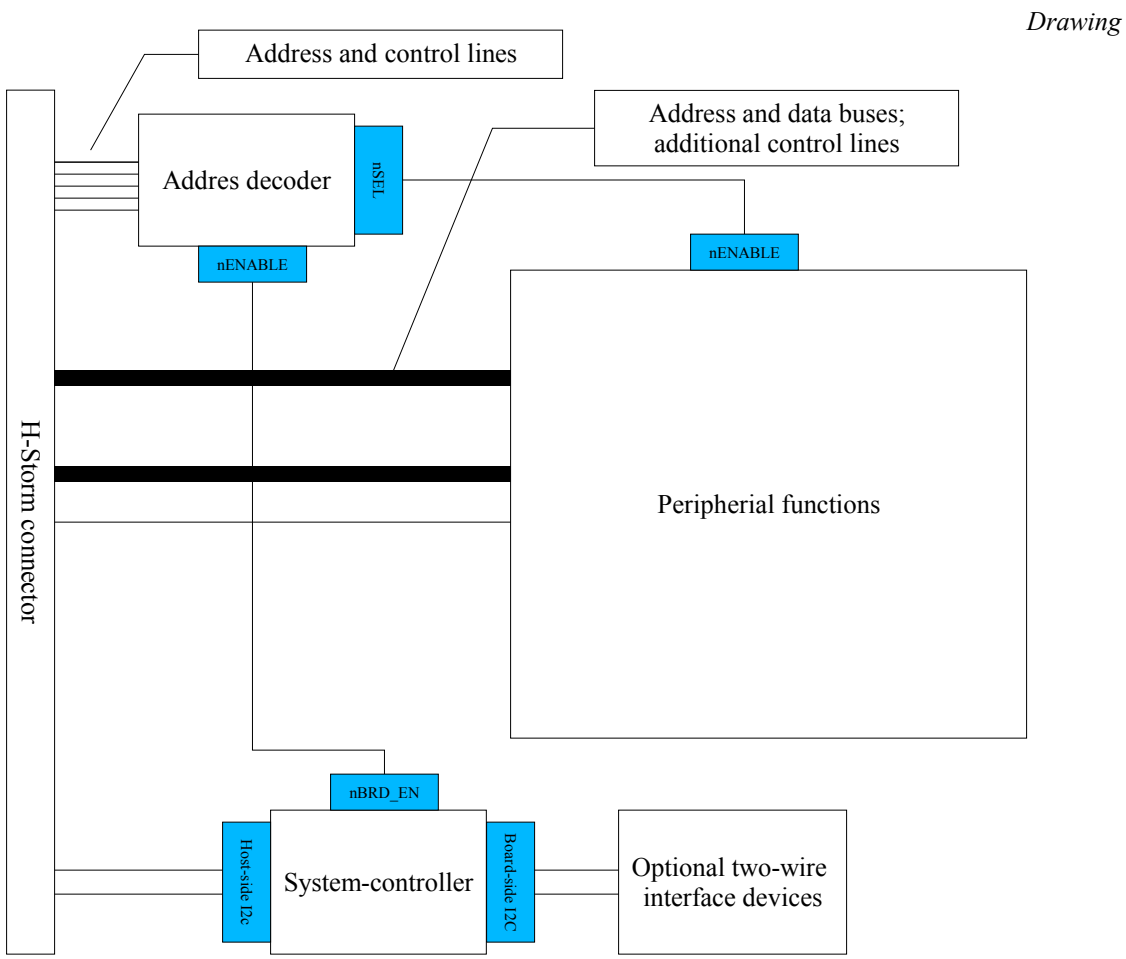

Figure 25: System-controller connection

The configuration protocol identifies five commands:

- A CMD_RESET command is used to set the system-controller to it's power-on state
- A CMD_ASSIGN_SW_ADDRESS soft-address assign command is used to assign a unique soft-address to the system-controller. This soft address is used for all subsequent commands.
- The CMD_ENABLE_BOARD command is used to enable system-bus communication between the CPU card and the peripherial
- The CMD_DISABLE_BOARD command is used to disable system-bus communication between the CPU card and the peripherial
- The CMD_I2C_PASSTHROUGH command is used to connect the board-side two-wire interface to the PnP bus.

The PnP discovery sequence

1. At the beginning of the PnP discovery, the CPU issues the CMD_RESET command. This will be interpreted by all system-controllers in the system. They will reset their configuration which means, they delete their assigned soft-address and isolate the board from the system-bus. Their board-side two-wire bus should have been isolated from the PnP bus already, so no disconnection is required there.
2. The CPU than issues the first CMD_ASSIGN_SW_ADDRESS command. This will be received by all the system-controllers which don't have an assigned soft-address (at the beginning of the process all system-controllers), and they use the two-wire interface bus-arbitration protocol to decide which one to answer the

request. This is done by each device sending their unique ID over the PnP bus and listening back to actual value transmitted over the line. If the read-back address matches the ID of the device, they remain on the line, if not, they abort the operation and stop sending any information on the PnP bus until a new command arrives. This process will leave only one device on the line by the time the full ID is transmitted. The CPU than send the soft-address to that device and records the fact that a peripherial device (or system-board) was detected. It repeats this step over and over again, until all system-controllers have an assigned soft-address. At that point no system-controller will answer to the CMD_ASSING_SW_ADDRESS command.

3. The CPU card iterates through all the boards (soft-addresses) it assigned and reads the PnP configuration data from the boards. This is done by accessing the PnP configuration ROM on the board-side two-wire interface of each board, using the CMD_READ_PNP_DATA command.
4. At this point the CPU card can figure out what cards are connected to the system, what are their requirements with regards to power, timing, system resources, and so on. The only thing that's missing is the location of each board: which of the identified board would answer for nSEL0, which for nSEL1 and which for nSEL2. This information cannot be retrieved from using the PnP bus. Each card however can individually isolated from the system-bus, so the following approach can be used:
5. Isolate all but one peripherial using the CMD_ENABLE_BOARD and CMD_DISABLE_BOARD commands. After that, program all nSELx signals to be compatible with that particular peripherial. Try to access a known location on that peripherial board to detect the presence of it on each nSEL signal. The one(s) the board answers will give the CPU the location info. Repeat this for peripherial in the system.
   A possible speed up the above process would be to try only locations where no other card has been detected before. This would also eliminate the need to isolate cards that were already identified.
6. If cards were isolated after detection, set up the memory controller for each nSELx signal to be compatible with the detected card in that slot.
7. At the end of the process the software knows the location of each component and have configured itself to use compatible cycles to access them. It can also associate the system-controllers with the nSELx signals (peripherial slots) so that it can access PnP-bus devices on a peripherial board if that's necessary using the CMD_I2C_PASSTHROUGH command.

Note, that this process requires a board-specific detection scheme on the system-bus in step 5. However there are a couple of things to help the process:
The algorithm does not have to take other cards into account. It can assume that if anything answers to a particular nSELx signal, it is the card that the algorithm needs to detect (due to the fact that all other cards are isolated). It also knows exactly what type of card it looks for, since that information was retrieved from the PnP ROM. If an unknown card is detected, for which there's no detection algorithm in the system, that card would be useless anyway (no driver routines are present), so the process can simply ignore them and keep them isolated.

System-controller transactions

The first byte in each transaction is the physical address of the system-controllers. Each system controller has the same physical address of 0x65. This

First byte: Device physical address

MSB							LSB
Physical address (0x65)							

Second byte: Command code and logical address

MSB							LSB
Command code			Soft address				

Command code values:

0x00: CMD_RESET
0x01: CMD_ASSIGN_SW_ADDRESS
0x02: CMD_I2C_PASSTHROUGH
0x03: CMD_READ_PNP_DATA
0x04: CMD_ENABLE_BOARD
0x05: CMD_DISABLE_BOARD
Following bytes are command-specific

Initialization and the CMD_RESET command (command code 0x00)

This command is interpreted by each system-controller independent of if they have assigned a soft-address or not. When receiving this command, they isolate the board, disconnect their board-side interface (though it should be already disconnected) and delete their soft-address. The soft address for this command should be 0. All other values are reserved and should not be used.

Bus-arbitration and the CMD_ASSIGN_SW_ADDRESS command (command code 0x01)

In the H-Storm identification case, each device has a unique 128-bit identifier. This identifier is read by the CMD_ASSIGN_SW_ADDRESS command. This command is interpreted by each system-controller who does not have an assigned soft-address. After receiving this command, each (addressed) system-controller starts sending their unique identifier and monitoring the data-line at the same time for collisions. Using the two-wire interface arbitration process at the end of the transmission of the ID only one device remains on the bus. All the others detected a conflict and disconnected. When the CPU received the ID, it assigns and sends a soft-address to the device. The device ID is required to be less than 32 and non-zero. The device stores this address, and no longer answers to the CMD_ASSING_SW_ADDRESS command. The soft address for this command should be 0. All other values are reserved and should not be used.

The details of CMD_I2C_PASSTHROUGH command (command code 0x02)

This command is used to communicate with devices connected to the board-side port of the system controller. This command is recognized by devices with a valid assigned soft-address only. Devices match the soft-address in the second byte with their assigned address and answer to the command only if it matches.
After the second byte is received, the system-controller generates a start-condition on the board-side interface and after that starts mirroring the board-side and the bus-side interfaces, logically connecting the two interfaces together. The system-controller will not interpret any information sent on either of its interfaces and continues mirroring the information until a stop or re-start condition is detected. At that point it generates a stop condition on the board-side interface (in case of a re-start condition only) and disconnects the two interfaces from each other.
This command in practice can be prepended to any transaction the master would like to send to a device on the particular peripherial. The command will open a channel towards that device, and will close the channel when the transaction completes.

The CMD_READ_PNP_DATA command (command code 0x03)

This command is used to read the PnP configuration data form the board by the CPU. This command in practice is really similar to CMD_I2C_PASSTHROUGH. It uses the soft-address to select a single system-controller and – in some versions of the system-controller – can eventually open up the board-side interface and mirror the communication until a stop or restart condition is detected. Its format mimics the access cycles to a 24c32 or bigger two-wire interfaced EEPROM device.
There are two reasons for separating this type of access out of the CMD_I2C_PASSTHROUGH command. One is that smaller EEPROM devices have a different access method and would require additional logic form the

PnP software to identify the correct size of the chip, which can't be done without modifying the content of the chip. The other reason is that in some cases the PnP configuration record can be stored in the system-controller itself. The controller than can decide to answer the CMD_READ_PNP_DATA command completely and not mirror the payload part over to the board-side interface.
The command has multiple formats and supports many different operations. They are defined as follows:

Set address sub-command

First byte: Device physical address

MSB							LSB
Physical address (0x65)							

Second byte: Command code and logical address

MSB							LSB
Command code (0x03)			Soft address				

Third byte: EEPROM device address and access direction flag

MSB							LSB
1	0	1	0	0	0	0	0

Fourth byte: Address high byte

MSB							LSB
Address high byte							

Fifth byte: Address low byte

MSB							LSB
Address low byte							

This command sets the internal address-counter. Every memory-operation uses this address and post-increments it after each memory-location is accessed. The current implementation supports a 768 byte long configuration ROM area. The content of this memory can be specified at configuration time. This memory size is sufficient to store the PnP configuration record for all peripherial cards that use only a single or two nSELx lines. With an alternative configuration it also supports external EEPROM memories of 24c32 or bigger.

Read sub-command

First byte: Device physical address

MSB							LSB
Physical address (0x65)							

Second byte: Command code and logical address

MSB							LSB
Command code (0x03)			Soft address				

Third byte: EEPROM device address and access direction flag

MSB							LSB
1	0	1	0	0	0	0	1

Fourth byte: First data byte

MSB							LSB
Data at address							

Subsequent bytes: Next data byte

MSB							LSB
Data at address + 1							

This command reads sequential data from the address counter. Every memory-operation post-increments the address counter.

The nBRD_EN signal, the CMD_ENABLE_BOARD (0x04) and CMD_DISABLE_BOARD 0x05) commands

An external pin, called nBRD_EN can be enabled (pulled low) or disabled (released) by these commands. The external pin is an open-drain output and requires an external pull-up resistor. This pin is usually used in H-Storm to enable or disable the board-selection circuit on the board. The peripherial should not respond to any system-bus cycles if this signal is inactive (high). This signal resets to high and needs to be enabled by the CMD_ENABLE_BOARD command. However this signal is brought out to the H-Storm connector and can be pulled low externally. In that case the peripherial will be connected to the system-bus independent of the CMD_ENABLE_BOARD or CMD_DISABLE_BOARD commands.

The unique identifier

Each system-controller is assigned a unique ID. This 128-bit value is generated when the device is programmed, using GUID generation routines. This technique ensures unique Ids without the need for a central database or communication with a registrar of used Ids. There are a number of algorithms and utilities available for generating GUIDs, one is described on this web-page: http://www.ics.uci.edu/~ejw/authoring/uuid-guid/draft-leach-uuids-guids-01.txt.

System controller implementation

The system controller as it is right now is implemented using an 8-pin 8-bit AVR microcontroller from Atmel, the AT90LS2343. The source code and binary images for the program are available on the H-Storm project website. The pin-out of the microcontroller when used as a system-controller in an H-Storm system is as follows:

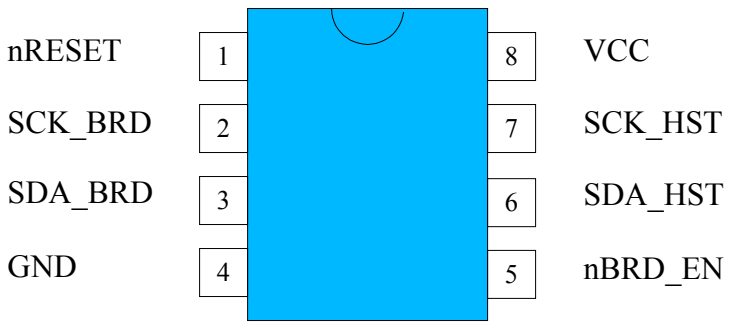

Figure 26: System-controller pin-out

This pin-out and connection of the chip, described previously allows it to be in-system-programmed using the pins that are connected to the H-Storm connector. This has two benefits: it can be in-system programmed without any special connector occupying additional PCB real estate and the device can be soldered in place

blanked and programmed later on, when the whole board is assembled. The drawback is that special care must be taken not to accidentally re-program the device on the board. Reprogramming however requires that the PnP bus signals as well as the nBRD_EN signal change their state while nRESET is pulled low. This almost never happen accidentally and is prohibited by the H-Storm specification.
Of course a peripherial board can use a bigger controller or a different one as long as the it implements the described protocol.

PnP ROM content

Every Peripherial card contains a small two-wire interfaced ROM. This device can be read from any CPU card using only the two-wire PnP bus. The ROM contains information on how to access the card, what functionality is it providing, what is the name and the manufacturer of the card as well as a unique identifier for the card. It also might contain-user-defined fields after the predefined part of the memory.
The ROM also contains extensive timing information fully describing the behavior and the requirements of the card. This information can be used by the software on the CPU card to set up the peripherial controller found on many CPUs. At the very least the software is able to determine if it can safely communicate with the peripherial or not. This ROM is usually embedded into the system-controller chip, however its also possible to use a separate two-wire interface EEPROM device, like the 24c32. If an external device is used, it must be connected to the board-side interface of the system-controller and be configured to physical address 0xA0.

Data types

IDs

IDs in H-Storm are used to identify the manufacturer of the device, the type of the device, and to uniquely identify the device for PnP enumeration. These IDs are 128-bit GUIDs, stored as binary values in 8-bites, LSB first. There are various algorithms available that can generate guarantied to be unique GUIDs without consulting a central database. It is the responsibility of each board-manufacturer to fill in these IDs with valid values.

Timing data

For each timing data two 8-bit values are presented, a minimum and a maximum value. The values are encoded in a special floating-point scheme with a 3-bit unsigned exponent, and a 5-bit signed mantissa. The value 0xE0 is reserved for don't care/not specified values.

MSB							LSB
Exponent			Mantissa				

To convert the value to a 16-bit integer, use the following formula (with remembering to sign-extend the mantissa first to 16-bits):
`Timing_value=(Mantissa<<Exponent)`
The decoded timing value (`Timing_value`) is measured in nanoseconds. Note that all values where the Mantissa part is 0 encodes the timing_value 0. However only the encoding 0x00 is valid. All other possible encodings are reserved for future use. The value 0xE0 is already assigned to don't care/not specified values. The available range of timing information is around ±3.8µs. In the following table the type `Timing` refers to a 16-bit location, containing two 8-bit timing values, encoded in the above mentioned fashion. The minimum value is stored in the lower 8-bits and the maximum value in the higher 8-bits of the 16-bit word.

Power requirement data

For each power requirement data, 4 8-bit values are specified, representing the idle, power-down, normal and peak power consumption of a particular device over a particular power rail. The four values are packed into a 32-bit value, where the most significant byte contains the idle value, the next one the power down, after that the normal and the least significant byte finally the peak value:

MSB			LSB
IDLE value	PWR_DN value	NORMAL value	PEAK value

Each 8-bit requirement data is represented with the same encoding as timing values:

MSB							LSB
Exponent			Mantissa				

The value 0 is reserved and denotes a situation when the power line is not used. 0xE0 is also reserved for don't care/not specified values. The values are measured in mA. The available range is therefore around ±3.8A. Negative values indicate that the device can source current to the power line while positive values indicate that the device consumes power.

Strings

String in the configuration record are fixed-length, 0-padded strings. This means, that the size of the array, storing a string value is predefined. Strings shorter than the maximum available length are padded with 0x00 at their end. Note, that if the string is exactly as long as the array, the string will not be zero-terminated. Strings are encoded as ASCII or UTF-8, when it's appropriate.

Structures

System-bus timing structure

This structure contains timing values for each of the eight defined access cycles. As mentioned previously, each value is a 16-bit timing data, encoding a minimum and a maximum values. Offsets are relative to the start of the structure.

Timing information for normal read cycle without external wait-states

Offset 0: NR_T_CYCLE

NR_T_CYCLE: 16-bit `Timing` information. Width of the read access cycle

Offset 2: NR_T_IDLE

NR_T_IDLE: 16-bit `Timing` information. Time between two consecutive accesses

Offset 4: NR_T_SEL_RD_STP

NR_T_SEL_RD_STP: 16-bit `Timing` information. Setup delay between the assert of nSELx and nRD (generally shouldn't matter)

Offset 6: NR_T_SEL_RD_ED

NR_T_SEL_RD_ED: 16-bit `Timing` information. End delay between the de-assert of nSELx and nRD (generally shouldn't matter)

Offset 8: NR_T_CYC_ARD_SD

NR_T_CYC_ADR_SD: 16-bit `Timing` information. Setup delay between an active cycle-start and the valid address is presented on the address-bus

Offset 10: NR_T_CYC_ADR_HLD

T_CYC_ADR_HLD: 16-bit `Timing` information. Hold time between an active cycle-end and the valid address is removed from the address-bus

Offset 12: NR_T_CYC_DAT_DRV

NR_T_CYC_DAT_DRV: 16-bit `Timing` information. Time between the start of an active cycle and the bus-drivers are enabled on the peripherial card

Offset 14: NR_T_ADR_DAT_STP

NR_T_ADR_DAT_STP: 16-bit `Timing` information. Time between a valid address is presented on the bus and the valid data is presented on the bus If the address is valid, before nCYC goes low (NR_T_CYC_ADR_SD is negative) this time is measured from the falling edge of nCYC till the appearance of the valid address.

Offset 16: NR_T_CYC_DAT_HLD

NR_T_CYC_DAT_HLD: 16-bit `Timing` information. Time between the end of an active cycle and the valid data is removed from the bus

Offset 18: NR_T_CYC_DAT_REL

NR_T_CYC_DAT_REL: 16-bit `Timing` information. Time between the end of an active cycle and the bus-drivers are disabled on the peripherial card

Timing information for normal read cycle with external wait-states

Offset 20: NRW_T_CYCLE

NRW_T_CYCLE: 16-bit `Timing` information. Width of the read access cycle

Offset 22: NRW_T_IDLE

NRW_T_IDLE: 16-bit `Timing` information. Time between two consecutive accesses

Offset 24: NRW_T_SEL_RD_STP

NRW_T_SEL_RD_STP: 16-bit `Timing` information. Setup delay between the assert of nSELx and nRD (generally shouldn't matter)

Offset 26: NRW_T_SEL_RD_ED

NRW_T_SEL_RD_ED: 16-bit `Timing` information. End delay between the de-assert of nSELx and nRD (generally shouldn't matter)

Offset 28: NRW_T_CYC_ADR_SD

NRW_T_CYC_ADR_SD: 16-bit `Timing` information. Setup delay between an active cycle-start and the valid address is presented on the address-bus

Offset 30: NRW_T_CYC_ADR_HLD

NRW_T_CYC_ADR_HLD: 16-bit `Timing` information. Hold time between an active cycle-end and the valid address is removed from the address-bus

Offset 32: NRW_T_CYC_DAT_DRV

NRW_T_CYC_DAT_DRV: 16-bit `Timing` information. Time between the start of an active cycle and the bus-drivers are enabled on the peripherial card

Offset 34: NRW_T_WAIT_DAT_STP

NRW_T_WAIT_DAT_STP: 16-bit `Timing` information. Time between a valid address is presented on the bus and the valid data is presented on the bus

Offset 36: NRW_T_CYC_DAT_HLD

NRW_T_CYC_DAT_HLD: 16-bit `Timing` information. Time between the end of an active cycle and the valid data is removed from the bus

Offset 38: NRW_T_CYC_DAT_REL

NRW_T_CYC_DAT_REL: 16-bit `Timing` information. Time between the end of an active cycle and the bus-drivers are disabled on the peripherial card

Offset 40: NRW_T_CYC_WAIT_STP

NRW_T_CYC_WAIT_STP: 16-bit `Timing` information. Time between the start of an active cycle and the assertion of the nWAIT signal.

Offset 42: NRW_T_ADR_WAIT_STP

NRW_T_ADR_WAIT_STP: 16-bit `Timing` information. Time between a valid address is presented on the address lines and the assertion of the nWAIT signal.

Timing information for normal write cycle without external wait-states

Offset 44: NW_T_CYCLE

NW_T_CYCLE: width of the read access cycle

Offset 46: NW_T_IDLE

NW_T _IDLE: time between two consecutive accesses

Offset 48: NW_T_SEL_WE_STP

NW_T_SEL_WE_STP: setup delay between the assert of nSELx and nxWE (generally shouldn't matter)

Offset 50: NW_T_SEL-WE-ED

NW_T_SEL-WE-ED: end delay between the de-assert of nSELx and nxWE (generally shouldn't matter)

Offset 52: NW_T_W-W-SD

NW_T_W-W-SD: setup delay between the two write-enable signals (nLWE and nUWE) in a 16-bit access

Offset 54: NW_T_W-W-ED

NW_T_W-W-ED: end delay between the two write-enable signals (nLWE and nUWE) in a 16-bit access

Offset 56: NW_T_CYC-ADR-SD

NW_T_CYC-ADR-SD: setup delay between an active cycle-start and the valid address is presented on the address-bus

Offset 58: NW_T_CYC-ADR-HLD

NW_T_CYC-ADR-HLD: hold time between an active cycle-end and the valid address is removed from the address-bus

Offset 60: NW_T_CYC-DAT-DRV

NW_T_CYC-DAT-DRV: time between the start of an active cycle and the bus-drivers are enabled on the peripherial card

Offset 62: NW_T_CYC-DAT-STP

NW_T_CYC-DAT-STP: setup time between the end of an active cycle and a valid data is presented on the address-bus

Offset 64: NW_T_CYC-DAT-HLD

NW_T_CYC-DAT-HLD: time between the end of an active cycle and the valid data is removed from the bus

Offset 66: NW_T_CYC-DAT-REL

NW_T_CYC-DAT-REL: time between the end of an active cycle and the bus-drivers are disabled on the peripherial card

Normal Write cycle with external wait-states

Offset 68: NWW_T_CYCLE

NWW_T_CYCLE: width of the read access cycle

Offset 70: NWW_T_IDLE

NWW_T_IDLE: time between two consecutive accesses

Offset 72: NWW_T_SEL_WE_STP

NWW_T_SEL_WE_STP: setup delay between the assert of nSELx and nxWE (generally shouldn't matter)

Offset 74: NWW_T_SEL_WE_ED

NWW_T_SEL_WE_ED: end delay between the de-assert of nSELx and nxWE (generally shouldn't matter)

Offset 76: NWW_T_W_W_SD

NWW_T_W_W_SD: setup delay between the two write-enable signals (nLWE and nUWE) in a 16-bit access

Offset 78: NWW_T_W_W_ED

NWW_T_W_W_ED: end delay between the two write-enable signals (nLWE and nUWE) in a 16-bit access

Offset 80: NWW_T_CYC_ADR_SD

NWW_T_CYC_ADR_SD: setup delay between an active cycle-start and the valid address is presented on the address-bus

Offset 82: NWW_T_CYC_ADR_HLD

NWW_T_CYC_ADR_HLD: hold time between an active cycle-end and the valid address is removed from the address-bus

Offset 84: NWW_T_CYC_DAT_DRV

NWW_T_CYC_DAT_DRV: time between the start of an active cycle and the bus-drivers are enabled on the peripherial card

Offset 86: NWW_T_CYC_DAT_STP

NWW_T_CYC_DAT_STP: setup time between the end of an active cycle and a valid data is presented on the address-bus

Offset 88: NWW_T_CYC_DAT_HLD

NWW_T_CYC_DAT_HLD: time between the end of an active cycle and the valid data is removed from the bus

Offset 90: NWW_T_CYC_DAT_REL

NWW_T_CYC_DAT_REL: time between the end of an active cycle and the bus-drivers are disabled on the peripherial card

Offset 92: NWW_T_CYC_WAIT_STP

NWW_T_CYC_WAIT_STP: time between the start of an active cycle and the assertion of the nWAIT signal.

Offset 94: NWW_T_ADR_WAIT_STP

NWW_T_ADR_WAIT_STP: time between a valid address is presented on the address lines and the assertion of the nWAIT signal.

Burst Read cycle without external wait-states

Offset 96: BR_T_CYCLE

BR_T_CYCLE: width of the read access cycle

Offset 98: BR_T_B_CYCLE

BR_T_B_CYCLE: cycle time of each read in the burst

Offset 100: BR_T_IDLE

BR_T_IDLE: time between two consecutive accesses

Offset 102: BR_T_SEL_RD_STP

BR_T_SEL_RD_STP: setup delay between the assert of nSELx and nRD (generally shouldn't matter)

Offset 104: BR_T_SEL_RD_ED

BR_T_SEL_RD_ED: end delay between the de-assert of nSELx and nRD (generally shouldn't matter)

Offset 106: BR_T_CYC_ADR_SD

BR_T_CYC_ADR_SD: setup delay between an active cycle-start and the valid address is presented on the address-bus

Offset 108: BR_T_CYC_ADR_HLD

BR_T_CYC_ADR_HLD: hold time between an active cycle-end and the valid address is removed from the address-bus

Offset 110: BR_T_CYC_DAT_DRV

BR_T_CYC_DAT_DRV: time between the start of an active cycle and the bus-drivers are enabled on the peripherial card

Offset 112: BR_T_ADR_DAT_STP

BR_T_ADR_DAT_STP: time between a valid address is presented on the bus and the valid data is presented on the bus If the address is valid, before nCYC goes low (BR_T_CYC_ADR_SD is negative) this time is measured from the falling edge of nCYC till the appearance of the valid address.

Offset 114: BR_T_CYC_DAT_HLD

BR_T_CYC_DAT_HLD: time between the end of an active cycle and the valid data is removed from the bus

Offset 116: BR_T_CYC_DAT_REL

BR_T_CYC_DAT_REL: time between the end of an active cycle and the bus-drivers are disabled on the peripherial card

Burst Read cycle with external wait-states

Offset 118: BRW_T_CYCLE

BRW_T_CYCLE: width of the read access cycle

Offset 120: BRW_T_B_CYCLE

BRW_T_B_CYCLE: cycle time of each read in the burst

Offset 122: BRW_T_IDLE

BRW_T_IDLE: time between two consecutive accesses

Offset 124: BRW_T_SEL_RD_STP

BRW_T_SEL_RD_STP: setup delay between the assert of nSELx and nRD (generally shouldn't matter)

Offset 126: BRW_T_SEL_RD_ED

BRW_T_SEL_RD_ED: end delay between the de-assert of nSELx and nRD (generally shouldn't matter)

Offset 128: BRW_T_CYC_ADR_SD

BRW_T_CYC_ADR_SD: setup delay between an active cycle-start and the valid address is presented on the address-bus

Offset 130: BRW_T_CYC_ADR_HLD

BRW_T_CYC_ADR_HLD: hold time between an active cycle-end and the valid address is removed from the address-bus

Offset 132: BRW_T_CYC_DAT_DRV

BRW_T_CYC_DAT_DRV: time between the start of an active cycle and the bus-drivers are enabled on the peripherial card

Offset 134: BRW_T_WAIT_DAT_STP

BRW_T_WAIT_DAT_STP: time between a valid address is presented on the bus and the valid data is presented on the bus

Offset 136: BRW_T_CYC_DAT_HLD

BRW_T_CYC_DAT_HLD: time between the end of an active cycle and the valid data is removed from the bus

Offset 138: BRW_T_CYC_DAT_REL

BRW_T_CYC_DAT_REL: time between the end of an active cycle and the bus-drivers are disabled on the peripherial card

Offset 140: BRW_T_CYC_WAIT_STP

BRW_T_CYC_WAIT_STP: time between the start of an active cycle and the assertion of the nWAIT signal.

Offset 142: BRW_T_ADR_WAIT_STP

BRW_T_ADR_WAIT_STP: time between a valid address is presented on the address lines and the assertion of the nWAIT signal.

Offset 144: BRW_T_ADR_DAT_HLD

BRW_T_ADR_DAT_HLD: time between the change of the address on the address lines and the removal of the valid data from the data lines.

Burst Write cycle without external wait-states

Offset 146: BW_T_CYCLE

BW_T_CYCLE: width of the read access cycle

Offset 148: BW_T_B_ CYCLE

BW_T_B_ CYCLE: cycle time of each read in the burst

Offset 150: BW_T_IDLE

BW_T_IDLE: time between two consecutive accesses

Offset 152: BW_T_SEL_WE_STP

BW_T_SEL_WE_STP: setup delay between the assert of nSELx and nxWE (generally shouldn't matter)

Offset 154: BW_T_SEL_WE_ED

BW_T_SEL_WE_ED: end delay between the de-assert of nSELx and nxWE (generally shouldn't matter)

Offset 156: BW_T_W_W-SD

BW_T_W_W-SD: setup delay between the two write-enable signals (nLWE and nUWE) in a 16-bit access

Offset 158: BW_T_W_W-ED

BW_T_W_W-ED: end delay between the two write-enable signals (nLWE and nUWE) in a 16-bit access

Offset 160: BW_T_CYC_ADR_SD

BW_T_CYC_ADR_SD: setup delay between an active cycle-start and the valid address is presented on the address-bus

Offset 162: BW_T_CYC_ADR_HLD

BW_T_CYC_ADR_HLD: hold time between an active cycle-end and the valid address is removed from the address-bus

Offset 164: BW_T_CYC_DAT_DRV

BW_T_CYC_DAT_DRV: time between the start of an active cycle and the bus-drivers are enabled on the peripherial card

Offset 166: BW_T_CYC_DAT_STP

BW_T_CYC_DAT_STP: setup time between the end of an active cycle and a valid data is presented on the address-bus

Offset 168: BW_T_CYC_DAT_HLD

BW_T_CYC_DAT_HLD: time between the end of an active cycle and the valid data is removed from the bus

Offset 170: BW_T_CYC_DAT_REL

BW_T_CYC_DAT_REL: time between the end of an active cycle and the bus-drivers are disabled on the peripherial card

Burst Write cycle with external wait-states

Offset 172: BWW_T_CYCLE

BWW_T_CYCLE: width of the read access cycle

Offset 174: BWW_T_B_ CYCLE

BRW_T_B_ CYCLE: cycle time of each read in the burst

Offset 176: BWW_T_IDLE

BWW_T_IDLE: time between two consecutive accesses

Offset 178: BWW_T_SEL_WE_STP

BWW_T_SEL_WE_STP: setup delay between the assert of nSELx and nxWE (generally shouldn't matter)

Offset 180: BWW_T_SEL_WE_ED

BWW_T_SEL_WE_ED: end delay between the de-assert of nSELx and nxWE (generally shouldn't matter)

Offset 182: BWW_T_W_W-SD

BWW_T_W_W-SD: setup delay between the two write-enable signals (nLWE and nUWE) in a 16-bit access

Offset 184: BWW_T_W_W-ED

BWW_T_W_W-ED: end delay between the two write-enable signals (nLWE and nUWE) in a 16-bit access

Offset 186: BWW_T_CYC_ADR_SD

BWW_T_CYC_ADR_SD: setup delay between an active cycle-start and the valid address is presented on the address-bus

Offset 188: BWW_T_CYC_ADR_HLD

BWW_T_CYC_ADR_HLD: hold time between an active cycle-end and the valid address is removed from the address-bus

Offset 190: BWW_T_CYC_DAT_DRV

BWW_T_CYC_DAT_DRV: time between the start of an active cycle and the bus-drivers are enabled on the peripherial card

Offset 192: BWW_T_CYC_DAT_STP

BWW_T_CYC_DAT_STP: setup time between the end of an active cycle and a valid data is presented on the address-bus

Offset 192: BWW_T_CYC_DAT_HLD

BWW_T_CYC_DAT_HLD: time between the end of an active cycle and the valid data is removed from the bus

Offset 196: BWW_T_CYC_DAT_REL

BWW_T_CYC_DAT_REL: time between the end of an active cycle and the bus-drivers are disabled on the peripherial card

Offset 198: BWW_T_CYC_WAIT_STP

BWW_T_CYC_WAIT_STP: time between the start of an active cycle and the assertion of the nWAIT signal.

Offset 200: BWW_T_ADR_WAIT_STP

BWW_T_ADR_WAIT_STP: time between a valid address is presented on the address lines and the assertion of the nWAIT signal.

Offset 202: RESERVED

RESERVED: This is a 16-bit reserved location to DWORD align the size of the structure.

Configuration record layout

The configuration record starts at location 0 of the PnP EEPROM. The structure has a variable size, but size-info is encoded in the structure itself. The layout presented here is for the configuration record version 2. Offsets are relative to the start of the record (0).

Offset 0: BLK_SIZE

BLK_SIZE 16-bit size of the predefined part of the configuration block, measured in bytes. The values 0x0000 and 0xffff are reserved and invalid. The reserved values are to distinguish an unprogrammed EEPROM from one with valid information in it. Any user-defined information starts right after the offset encoded into this location. Currently the maximum possible size for the record is 726 bytes. In most cases the size would be only 318.

Offset 2: BLK_VER

BLK_VER 16-bit version number, defining the version of the configuration block. This is a running number, incremented each time a major change to the configuration block is made. This document describes configuration block, version 0x0002. The values 0x0000 and 0xffff are reserved and invalid. These are to distinguish an unprogrammed EEPROM from one with valid information in it.

Offset 4: MNF_CODE

MNF_CODE: 128-bit unique manufacturer ID code.

Offset 12: PROD_CODE

PROD_CODE: 128-bit unique product ID code.

Offset 20: MNF_NAME

MNF_NAME: 32 character \0-padded company name string.

Offset 52: PROD_NAME

PROD_NAME: 32 character \0-padded product name string.

Offset 84: CONFIG_FLAGS1

MSB							LSB
RST_GEN	8_BIT_SUP	8_BIT_ONLY	CONN_C	CONN_B	WAIT_USED	NEEDS_WAIT	PERI

PERI 0: this is a peripherial card
1: this is a system-board

NEEDS_WAIT 0: the card doesn't need wait-state generation (still might generated wait-states, but it has a pre-defined maximum access time over which all accesses will complete)
1: the card can't function without wait-state generation. If the CPU card can't handle external wait-states the card and the CPU cards are incompatible.

WAIT_USED 0: the card won't generate wait-states
1: the card generates wait-states

CONN_B 0: the extension connector B is not used/present on the card
1: the extension connector B is used/present on the card

CONN_C 0: the extension connector C is not used/present on the card
1: the extension connector C is used/present on the card

8_BIT_ONLY 0: the card can handle 16-bit transactions
1: the card can handle 8-bit transactions only (this means that the upper 8-bit of a 16-bit transfer is ignored, so as 8-bit transfers to odd locations. Data is only accessible on even 8-bit locations.)

8_BIT_SUP 0: the card can't handle 8-bit transactions, only 16-bit ones. 8-bit reads are still supported but 8-bit write accesses lead to undefined results.
1: the card can handle 8-bit transactions as well as 16-bit ones.
This bit is ignored if 8_BIT_ONLY is set.

RST_GEN 0: the card doesn't have it's own reset generator
1: the card does generate it's own reset.

Offset 85: CONFIG_FLAGS2

MSB							LSB
			BRD	BWR	SEL2_USED	SEL1_USED	SEL0_USED

SEL0_USED 0: this card doesn't use the nSEL0 line
1: this card uses the nSEL0 line
If this bit is set, a timing configuration structure for the card select line is available

SEL1_USED 0: this card doesn't use the nSEL1 line
1: this card uses the nSEL1 line
If this bit is set, a timing configuration structure for the card select line is available
SEL2_USED 0: this card doesn't use the nSEL2 line
1: this card uses the nSEL2 line
If this bit is set, a timing configuration structure for the card select line is available
BWR: 0: this card doesn't support burst writes
1: this card supports burst writes
It's very uncommon for a card to doesn't support burst writes.
BRD: 0: this card doesn't support burst reads
1: this card supports burst reads
All other bits are reserved for future use and should be set to 0

Offset 86: IRQ_CONFIG_FLAGS

MSB							LSB
		IRQ2_MODE		IRQ1_MODE		IRQ0_MODE	

IRQx_MODE 00: this nIRQ line is not used
01: this nIRQ line generates edge-triggered interrupts
10: this nIRQ line generates level-triggered interrupts
11: this nIRQ line can generate both edge-triggered and level-triggered interrupts. The interrupt-generation mode is SW selectable in a card-specific way.
All other bits are reserved for future use and should be set to 0

Offset 87: DMA_CONFIG_FLAGS

MSB							LSB
		DMA2_MODE		DMA1_MODE		DMA0_MODE	

DMAx_MODE 00: this nDMA line is not used
01: this nDMA line generates edge-triggered DMA requests
10: this nDMA line generates level-triggered DMA requests
11: this nDMA line can generate both edge-triggered and level-triggered DMA requests. The DMA requests-generation mode is SW selectable in a card-specific way.
All other bits are reserved for future use and should be set to 0

Offset 88: RESERVED

RESERVED: 16-bit reserved data for future use. Should be 0x0000.

Offset 90: RESET_TIMING

RST_TIMING: 16-bit `Timing` information for the nRESET line. If on-board reset generator is provided, that generator should provide a pulse within the limits of this specification. Maximum value is not specified for most of the cards.

Offset 92: IRQ0_TIMING

IRQ0_TIMING: 16-bit `Timing` information for the nIRQ0 line. This value applies only to edge-triggered modes.

Offset 94: IRQ1_TIMING

IRQ1_TIMING: 16-bit `Timing` information for the nIRQ1 line. This value applies only to edge-triggered modes.

Offset 96: IRQ2_TIMING

IRQ2_TIMING: 16-bit `Timing` information for the nIRQ2 line. This value applies only to edge-triggered modes.

Offset 98: DMA0_TIMING

DMA0_TIMING: 16-bit `Timing` information for the nDRQ0 lines. This value applies only to edge-triggered modes.

Offset 100: DMA1_TIMING

DMA1_TIMING: 16-bit `Timing` information for the nDRQ1 lines. This value applies only to edge-triggered modes.

Offset 102: DMA2_TIMING

DMA2_TIMING: 16-bit `Timing` information for the nDRQ2 lines. This value applies only to edge-triggered modes.

Offset 104: PWR_33

PWR_33_IDLE: 32-bit power requirement information for the 3.3V rail

Offset 108: PWR_33

PWR_25_IDLE: 32-bit power requirement information for the 2.5V rail

Offset 112: PWR_18

PWR_18_IDLE: 32-bit power requirement information for the 1.8V rail

Offset 114: PRIMARY_TIMING

PRIMARY_TIMING: A 204 bytes long timing record for the first nSELx line that the card uses (nSEL0 in most cases). If the card doesn't use any nSEL lines, this field is optional.

Offset 318: SECONDARY_TIMING

SECONDARY_TIMING: A 204 bytes long timing record for the second nSELx line that the card uses (nSEL1 most likely, if any). If the card uses a single nSELx line, this field is optional.

Offset 522: TERTIARY_TIMING

TERTIARY_TIMING: A 204 bytes long timing record for the third nSELx line that the card uses (nSEL2 most likely, if any). If the card uses only two nSELx lines, this field is optional.

Software considerations

Though the H-Storm project doesn’t directly define any kind of software behavior, I had some typical software activities and solutions in mind when I designed the system. Some SW approaches even have HW or system

integration consequences. I'll list some the most important software activities and the recommended approach to attack them in an H-Storm environment

Boot-loader

Whenever an H-Storm CPU card comes out of its reset state, a boot procedure must follow. For most situations the following boot sequence is recommended:
Upon power-up, the CPU starts executing a small boot-loader from the on-board FLASH memory. This boot-program first programs one of the communication (serial) ports on the CPU cards and starts listening for a sync character from a host computer.
If a sync character is received, it enters into a boot-monitor mode and downloads a program from the host computer that later can be executed.
If it doesn't receive the sync character in a reasonable time period, it looks through the FLASH memory on the card for a loadable and executable program image.
If it finds one, it loads it to the memory and transfers the execution to it.
If no executable image is found, the program restarts.
Note, that this boot-loader has some special characteristics:
It never talks to any off-board peripherial. This means that its operation is not affected by any device connected to the H-Storm bus.
It initially doesn't configure any pin of the CPU card to output, so it can't produce an electric conflict no matter how the pins of the CPU card is connected
It starts sending data on the serial communication port only after receiving a valid sync signal. This means that output drivers on the CPU card are enabled only after the program made sure its safe to do so.
This initial listening phase can be used not only to make sure there's a host computer trying to connect, but also to detect the communication speed.
The whole procedure makes sure that a host-computer can always download a program and start execution of it independent of what program is already in CPU card (provided the boot-loader isn't overwritten, of course)
The process however delays the startup of the main application by the timeout of the initial communication (a couple of seconds) and it might not be acceptable for all purposes. There's also a slight chance that the boot-loader gets confused by an external device and it thinks it is a host-computer so care must be taken when using the on-board serial ports of the CPU cards with devices that can transmit data at any arbitrary time (without any handshake or initial setup).

H-Storm and SIMM-Sys: a comparison

The two designs have many commonalities though they also differ in some more-or-less important ways.

- First of all, while SIMM-Sys targets the professional audience, low volume and prototype-development, the H-Storm targets the home and hobby user. Though they share some market segment around prototyping they generally satisfy different needs. Most of the differences stem from this targeting difference.
- One consequence of the different target audience is a change in the priorities. Fast assembly, robust connections, small footprint, versatile mounting options gave way to other features, like easy assembly with low-tech tools, simple interfaces or cheap production.
- One of the most important changes was to change the connector from the 72-pin SIMM memory connector to a 72-pin dual-line 100mil header. This change though sacrificed some robustness, makes the modules much easier to mount even into no-solder bread-boards, or those pre-punched PCBs. It's basically no more complicated to use than a DIP socket. You can even attach a ribbon-cable to a module though that's not recommended do to EMI issues.
- The pin-out was also changed slightly so that the power pins line up better in the new dual-line layout
- The nPROG pin changed function and on Peripherial cards.

- The changed connector also allowed us to use a cheaper production process for the boards. This helps not only to keep the prices low but enables the use of less sophisticated production procedures. As an extreme case in theory at least it is possible to design a single-sided Peripherial board.
- The SIMM-Sys project discouraged Peripherial cards of having sockets on their sides. The proper way of interfacing with the outside world was through the general-purpose portion of the SIMM-Sys socket. This way the designer of the system-board had control over the type, location, side, mounting etc. of the connectors. In the H-Storm this flexibility was removed and now Peripherial boards usually have the connectors installed on either or both of their sides. This again, is less robust than the original SIMM-Sys concept, but makes system integration much simpler in a low-tech home environment. It also helps mitigate against EMI issues.
- While H-Storm kept all the different powering options (3.3V I/O, 3.3V, 2.5V and 1.8V core) most of the cards so far use only a single 3.3V power supply. They create whatever voltage they need internally. This gives the benefit of depending on various power supply voltages when the design requires, but makes system integration significantly easier when only boards with a single power supply requirement are used.
- CPU cards are now required to have proper pull-up resistors on all of their input signals on the bus-interface part of the connector. This includes nIRQ0...2, nWAIT, nRESET and nPROG. The PnP bus also has pull-up resistors on the CPU cards. This makes system-board design simpler and one of the components of the ‘just apply power’ concept.
- The nRESET signal is changed from input to I/O, with open-collector drive. Any component of an H-Storm design can drive the line low, keeping the system in the reset state. Also, the CPU cards are now required to generate proper power-on reset signals for themselves as well as for other parts of the system. While they can’t make sure that the reset signal is held low for all components in the system they have to make sure that the signal is low for long enough at least for the components on the CPU card. In most cases that’s satisfactory for all system components. If not, than that particular system component can drive the nRESET pin low for an extended period. This design change, again simplifies system integration and another part of the ‘just apply power’ concept.
- The PnP protocol has been changed dramatically. A system-controller was introduced and is required on every peripherial board and every active motherboard. This change while made PnP discovery more complicated, removed the need for an I2C bus-multiplexer on the system-board. The reason for the change was to simplify system-board design, the component of the system thats most likely to be designed and built for each application. This change has another interesting side-effect: the CPU and Peripherial cards are now interchangeable. The software complexity can is compensated with detailed documentation and sample (application and library) code
- Another change, related to the revised PnP operation is the introduction of the nBRD_EN pin on the peripherial cards. This pin, when pulled low, disables the PnP card-inhibit mechanism, and connects the peripherial card to the system-bus. For simple or well-controlled environment, where PnP operation is not required, simple connecting these pins of the peripherial boards to GND allows operation without completing the PnP discovery algorithm.
- A boot-loader code was developed and added as standard to the CPU cards. This of course applies only to cards that we supply. If you build your own cards, you are responsible for that. Anyway, in theory at least, this boot-loader first checks the FLASH memory for a valid boot image. If it finds one, it starts executing it. If it does not find one, it tries to contact a host computer through some means of communication (serial port for example) and download the program from there. The communication protocol is designed in such a way that it has almost no impact on how the general-purpose pins are used in a design. The boot-loader configures pins as output only after it made sure that it’s safe to do so, i.e. when it detected a valid connection request packet from the host.
- ‘Just apply power’ concept: the H-Storm CPU cards are designed in such a way that if you apply power to them, something valid will happen. They will properly power up (thanks to the on-board power-on reset

generation and the valid signal levels on all bus signals) and the boot-loader will start operating. It will (out of the box at least) not find a valid boot-image so will go ahead and try to connect to a host computer. Current CPU cards have some LEDs integrated on them so that the user gets some feedback of this operation. This concept gives early positive feedback to the not so experienced users and encourages them to continue exploring the capabilities of the system. The techniques allow the simplest system-board to contain only a 3.3V regulator as a power supply and nothing else.

Design Tips and Practices

The nPROG signal on CPU cards

The nPROG signal is used to palce the CPU cards into a special programming mode. This pin should not be connected under normal circumstances on the System Board and is required to be pulled up on the CPU cards. The exact operation of the CPU cards when the nPROG signal is low is dependent on the CPU card. In most cases it would however behave in one of the following ways:

- Alter the behavior of the address-decoder on the CPU card so that the CPU will start executing a program from an external memory connected to nSEL0 instead of the integrated FLASH device upon reset. In this mode a small boot-loader program can be placed in a 1kWord external memory that would initialize the CPU card and download a bigger application.
- Active a boot behavior of the CPU. Many modern SOC implementations have a boot functionality in which they execute a small application from an internal ROM. This application would download a bigger program and execute it through a serial port or USB or some other means. The option of booting the CPU into this special mode is usually controlled by an external pin during system-reset. This pin can be connected to the nPROG signal.

The nBRD_EN signal on Peripherial cards; How to inhibit the PnP board-isolation

The nBRD_EN signal on peripherial boards is an I/O signal with internal pull-up and open-drain drivers. This signal is not connected on system boards under normal circumstances. The signal is controlled by the on-board system-controller and is connected to the address-decoder logic of the peripherial card. When the signal is low, the card accept transactions over the system-bus. When the signal is high, the peripherial board is isolated from the system bus and it cannot be addressed. It is however possible that the card generates IRQ and DMA request signals if the card was previously programmed to do so. No peripherial cards are allowed to drive any of the nIRQx and nDRQx lines however without a SW enabling this functionality after reset. In other words a card should not drive any of the system-bus signals after reset as long as it is enabled by pulling the nBRD_EN signal low.

This signal, as mentioned earlier is controlled by the system-controller on board on the peripherial cards under normal circumstances.

However it is possible to drive this signal low externally, thus inhibiting the card-isolation functionality. This prevents the PnP discovery function from operate properly so this practice is discouraged unless the HW configuration of the whole system is static, well known and under close control. Note, that due to the internal pull-up resistor on this line, to reliably tie this signal to low level an active driver or a direct connection to GND is required. Don't use a pull-down resistor on this pin.

A secondary function of this pin is to provide in-system programming capability for the system-controller on the peripherial board. This functionality can be activated while the nRESET signal is active (low). It is therefore paramount that the state of this pin remain static while nRESET is active to prevent accidental reprogramming of the system-controller.

Simulation of DMA operations with a CPU that doesn't support it

Since DMA and normal bus-cycles are very similar, indeed almost identical, it's very easy for a CPU card to pretend to have a DMA engine. The only difference between a DMA acknowledge cycle and a normal access cycle is the state of the nDACK line. The timing of the nDACK line is identical to the address lines of the H-Storm bus. It is therefore possible to connect one of the unused address lines of the CPU to the nDACK line. In case of memory mapped peripherials, which almost all modern CPUs have, this would mean that from the software's point of view the peripherial would appear in two separate location of the memory. One with the additional address line being low, and one with it being high. Depending on which address range the software accesses the peripherial card would see either a normal or a DMA operation. If there's no additional address line you can use, it's also possible to connect a programmable output line to the nDACK signal. Prior access to **any** of the peripherials this signal would have to be set to the correct state though.
This covers the acknowledge part; however there's the question of how to detect requests. If the processor has enough free IRQ lines, you can simply connect the nDRQ lines to those and write an interrupt handler to process the requests.
If there isn't enough free interrupt lines, you somehow have to start combining multiple interrupt and DMA sources together. If the CPU supports level-sensitive interrupts, this can be as simple as 'and'-ing together the various signals ('and' operation is required due to the negated logic on the interrupt and DMA lines). If only edge-sensitive interrupts are supported, more software involvement might be necessary and a generic solution might not be possible (i.e. re-enabling the various sources might require peripherial-specific code).
This technique gives the fastest solution if there's no direct DMA support since this would simply convert DMA operations into interrupt-driver transfers.

When to use multiple nSELx lines for a single peripherial- or system-board

In general H-Storm discourages the use of multiple nSELx, nIRQx or nDRQx lines on a single peripherial card or system-board since it reduces the number of attachable components. However there are some cases when it might be better to use multiple resources. The reason for that is that most CPU card can control access cycle timing for each nSELx signal independently. They however can't control the timing for various address ranges inside a single nSELx range. So, if you incorporate two or more functions into a single peripherial- or system-board with vastly different timing requirements its worth considering using multiple nSELx signals to let the CPU optimally set up its timing. If the two incorporated components have incompatible timing requirements, you don't even really have another choice than to use multiple nSELx lines. For slight differences however, do not use multiple nSELx lines. Instead work out a timing requirement that works with all of the components and put that into the PnP configuration ROM. This might not lead to optimal timing but keep in mind that the CPU card probably doesn't have that low-granularity control over the timing parameters as it is specified in the configuration block anyway. It is quite possible that the resulting timing would have been identical even if you had used two different nSELx lines and configuration records.
Also note that its practical to tie the nSELx, nIRQx and nDRQx lines together. i.e. If you use nSEL1 to select a component on the peripherial card, use nDRQ1 to request DMA operations toward that component and nIRQ1 to signal interrupts generated by that component. Do that even if the component that is connected to nSEL0 doesn't use nDRQ0 or nIRQ0.

FAQ

Q: How to make sure that the boot-loader doesn't get confused by an external device?

A: The boot loader start by listening on one of the serial communication ports of the CPU card (provided there's one) for a sync character. If it receives a valid sync character it enters into a boot-loader mode where it communicates with a host computer. There's a slight chance that the boot-loader gets confused by an external device and it thinks it is a host-computer. So care must be taken when using the on-board serial ports of the CPU cards with devices that can transmit data at any arbitrary time (without any handshake or initial setup).
A simple solution to the problem would be to artificially cut the communication channel (the RxD line) upon power-on between this device and the CPU card. Once the main application starts executing and is ready to receive data from the external device, it can reconnect the device.

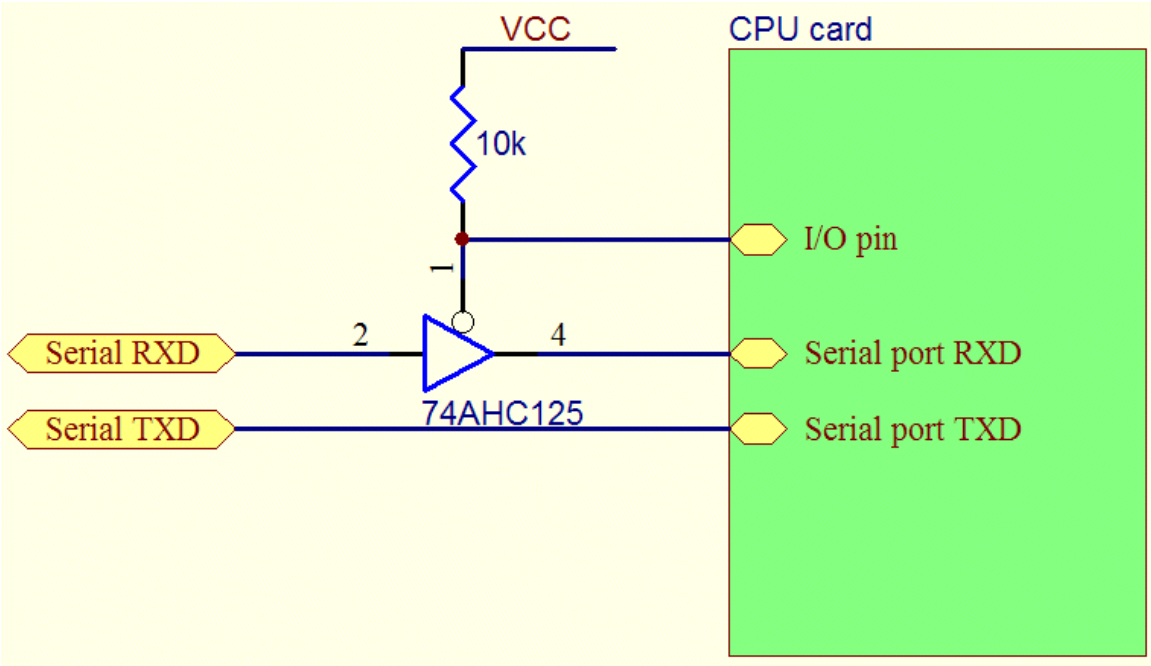

Figure 27: Serial port protection circuit

The above circuit is a simple implementation of such an idea. The tri-state buffer that connects the device to the CPU card is disabled, when its enable input is high. Upon reset all I/O lines of the CPU card is programmed as inputs, meaning they don't force any particular voltage level to their pins. The resistor on the enable-pin will pull up the potential to a high level, which disables the buffer. In this state, there's no driver on the RxD pin of the CPU card, so the boot-loader will not mistake any signals coming from the external device to the host-computers sync character. If on the other hand, a host computer is connected to the same port, the boot-loader will be able to communicate with it.
Once the main application is started, it can program the I/O pin to output and to a low level, at which point it will drive enough current to the pin to pull its potential to low level against the external resistor. The tri-state buffer is enabled and the communication between the CPU card and the external device is restored.

Q: What is the signaling level of the H-Storm bus signals?

A: All H-Storm bus signals are defined as LVCMOS33. This means that the low-level signals should be around 0V, while high-level signals should be around 3.3V.

Q: Are H-Storm bus signals 5V tolerant?

A: No, at least not by default. You have to check with the specific cards on 5V-compatibility.

Q: Can I connect non-LVCMOS33 signals to the application-defined parts of the H-Storm connector?

A: Yes, you can however please use special care. If your card has external signals with lower levels than 3.3V, your card can't damage any device on the system-board that expects such inputs. Your input lines however might get damaged by a system-board device trying to force a 3.3V level to one of the pins. Make sure your device can tolerate that (a typical example would be LVDS receivers).
If your card (like in the case of RS-232 transceivers) operates on wider voltage-swings than the LVCMOS33 signals, the situation is quite to opposite: you can damage system-board resources but you're safe from external LVCMOS33 sources. In that case it's a good practice to make those high-voltage signals disabled by default. The application can enable them if they are needed. Using the plug-and-play information from the peripherial cards and the system-board the application can make sure that it's safe to enable high-level signals on the peripherial card.
Obviously it's an error to plug such peripherial cards into connectors that they were not designed to be plugged into. They can't work in such environment and they are not expected to. The above techniques would only prevent permanent damage to the devices and in most cases, automatic detection of such an error.
Of course it is also possible to connect such signals unprotected to the I/O lines of the H-Storm connector. You only run the risk of damaging your card or the system-board if it isn't plugged into the wrong place.

Q: Why can't I use clock-stretching on the PnP bus? Is there anything I can do about it?

A: Clock-stretching would require the system-controller to drive the host-side SCL line in the same fashion it does with the SDA line. However it's not possible to detect with pure digital logic (or in this case SW, controlling digital lines) when the clock-stretching has been removed from the line without momentarily let the clock line to go high. This is the same for data lines, however while it is acceptable on data-lines to have some additional transitions while the clock line is low, it's prohibited on the clock line. The extra edges would trigger additional bits to be shifted in the receiver and the whole transfer would stop working. For this reason the system controller never reverses the direction of the SCL line: it always drives board-side interface from the host-side signal. Clock-stretching is not possible in that case.

Q: Why can't I use multiple masters on the PnP bus?

A: There are multiple reasons for this. One is that the system controller does not accept commands from the board-side interface and as such it could not be opened up from the board-side in the same way as it can be from the host-side. Another reason is that once its opened up, the direction of the clock signal is fixed and never reversed (see previous answer) and thus a board-side master would not be able to drive the clock line. A third reason is that it's not possible to negotiate bus ownership between masters on different sides of the system-controller, mostly since the controller does not mirror the two sides all the time (and that's the whole point of having it there). It would be possible to have multiple masters on the host-side interface however, the support of multi-master operations would complicate the two-wire interface communication routines in the CPU card SW.

The added benefit of having multiple masters in that limited fashion doesn't compensate for the incurred complications so the H-Storm specification simply disallow multi-master operations.

Q: A separate microcontroller for each board?! Isn't that overkill?

A: There are a couple of design-decisions that lead to this solution. One was that I wanted a side-band bus that worked completely independent from the main bus. This is to allow the complete discovery of the system by the CPU card without actually touching anything. This is a very important principle in seamless system integration. The software can be completely aware of its environment before starting the initialization of the system. It can also make sure that all the components are able to work together and there's no resource or other type of conflicts. If there were conflicts any operation on the bus might result in undefined situations, dead-locks or all kinds of weird things.
The other constrain was the limited resources I had to implement this side-band interface, especially regarding the number of pins to use. There were simply not enough pins to provide a per-peripherial select signal for the side-band bus as well. (And combining the side-band bus and the main bus select signals together is a bad idea since apart from the HW implications it would break the independency of the two buses.) So, the problem was: find a way to independently address two-wire interface devices with the same physical address without using any other resources than the ones connected to the two-wire interface.
There are two solutions, for the problem:
One is that there's a bus-multiplexer on the system-board. This multiplexer would be a Philips PCA9544 or similar device. It has one input interface and four output channels. It is possible connect and disconnect each of its output channels to the input interface using commands form the input interface. Since it has dedicated channels for each port, it solves the slot-detection issue and since all channels can be individually enabled or disabled it solves the address-conflict issues as well. Nice. There are a couple of problems though. This chip is next to impossible to get in small quantities I didn't like the idea anyway to incorporate a specific IC which has a single supplier in the core definition of the H-Storm system, let alone a chip that most of the potential users of the H-Storm system would not be able to purchase. The bigger problem however is that this chip would have to go to the system-board. This would significantly complicate system-board design, the component that I've tried so hard to keep as simple as possible.
The other solution, the one that I've choose to implement, to use some kind of SW-addressable device, like the 24LCS61/62 from Microchip and use the two-wire interfaces arbitration mechanism to sort out which one of the identical devices do we talk to. The above mentioned device though is not usable for the role for many reasons: it doesn't allow separation of other two-wire interface devices, like the system-controller does on its board-side interface, its memory size is smaller than the PnP configuration record, and its equally hard to get as the Philips component. This pretty much lead to a home-grown solution. An 8-pin microcontroller, thats in-system-programmable, cheap, easy to get seemed perfect for the task. I could define the protocol so that it fits the purpose well, and works with small overhead, and the single-chip implementation, with the integrated PnP memory emulation results in a true single-chip implementation. And if you think about it, this solution both price-wise, component-wise comparable to the others, but it has additional flexibility and the benefit of fitting the role perfectly. This solves the addressing issue over the two-wire interface, however there's one more problematic issue with this: even after being able to individually select each device on the two-wire interface it's still a non-trivial task to figure out which peripherial is connected to which socket. This issue is solved by the addition of the board-isolation feature: each board can be individually connected or isolated from the system-bus using the two-wire interface. If only a single peripherial is connected to the system bus, it's detection is almost straightforward and no resource-conflicts can arise during the process. Once all the cards are detected and the system-bus signals are configured correctly, all peripherials can be enabled again, and the system is functional. It's a bit more complicated SW process traded in for simplified system-board design. Also: with the use of the nBRD_EN signal on peripherial cards, this board-isolation feature can be inhibited, so if a system doesn't need such a high level of PnP functionality it is not required to be used.

An interesting side-effect of this design decision is that CPU and peripherial cards are interchangeable in a system-board. The is more of an aesthetic issue than one of a real importance.

Glossary

Pull-up

A (usually) resistor connecting a signal line to the VCC power line, pulling the voltage level on the wire 'up' unless a high-current driver is driving the line. The value of the resistor is in the range of 1-100kOhms. Since the driver has to work against the resistor, a lower pull-up resistor value results in higher static current and power consumption. When no driver is forcing a voltage level on the wire, the resistor sets its 'default' value to high. When a low-level drive is removed from the wire, the pull-up resistor has to charge the input and other capacitances on the wire to high level. This takes time and the higher the resistor value is, the longer it takes. These two contradicting factors dictate the value of a pull-up resistor. In most cases in H-Storm a 10k resistor value is used.

Pull-down

A (usually) resistor connecting a signal line to the GND power line, pulling the voltage level on the wire 'down' unless a high-current driver is driving the line. The value of the resistor is in the range of 1-100kOhms. Since the driver has to work against the resistor, a lower pull-down resistor value results in higher static current and power consumption. When no driver is forcing a voltage level on the wire, the resistor sets its 'default' value to low. When a high-level drive is removed from the wire, the pull-down resistor has to dis-charge the input and other capacitances on the wire to low level. This takes time and the higher the resistor value is, the longer it takes. These two contradicting factors dictate the value of a pull-down resistor. In most cases in H-Storm a 10k resistor value is used.

Totem-pole

The output of a digital circuit is called totem-pole if it can drive the wire to both high and low levels, i.e. it can both sink and source current to the line.

Open-collector

The output of a digital circuit is called open-collector or open-drain (depending on weather it is manufactured using a CMOS or bipolar process) if it can drive the wire to low levels only. To put the wire into high level, a pull-up resistor must be used. This resistor can be external or internal to the chip.

Open-drain

Same as open-collector, only for devices manufactured using a CMOS process.

Three-stated output

This type of output has three possible states: low and high level, just like the totem-pole outputs, and a high-impedance state. In this third state, the output doesn't force any particular voltage level to the wire. The output 'floats' unless a pull-up or pull-down resistor is connected to the line or another output drives a voltage level to the same wire.

Wired-OR

A technique that allows the implementation of logical OR combination of signals using passive components only. The idea is that the logical OR of signals is '1' if any of the components are '1'. The technique requires the drivers of the signals to actively drive the wire 'high' for logical '1's and release the wire for '0'. A pull-down resistor is used to set the state of the wire to'0' of no driver is enabled.

Wired-AND

A technique that allows the implementation of logical AND combination of signals using passive components only. The idea is that the logical AND of signals is '0' if any of the components are '0'. The technique requires the drivers of the signals to actively drive the wire 'low' for logical '0's and release the wire for '1'. A pull-up resistor is used to set the state of the wire to'1' of no driver is enabled. Open-drain or open-collector drivers can easily be combined into a wired-AND circuit.

Side-band bus

A side-band bus is almost anything that works along (on the side) of the primary bus, transmitting information independently from that. In the case of H-Storm this bus is a simple two-wire bus, which provides an alternative configuration channel and is used for plug-and-play information retrieval.

Two-wire interface

A two-wire interface is what Philips calls I²C™. It's also known as SMB (system-management bus) in the PC world. It's a wide-spread simple to use bus, with many cheap devices available for it. The version in H-Storm uses LVCMOS33 signaling levels, and signaling rates of no higher than 100kHz. You can find more information on this bus for example here on the Philips I2C bus site (http://www.semiconductors.philips.com/buses/i2c/).

Revision History

1.0 Initial release

1.1 Changed power and GND pin-out on extended connectors to match signal order of standard connectors. Corrected typo in DRQ channel numbering on extended connector