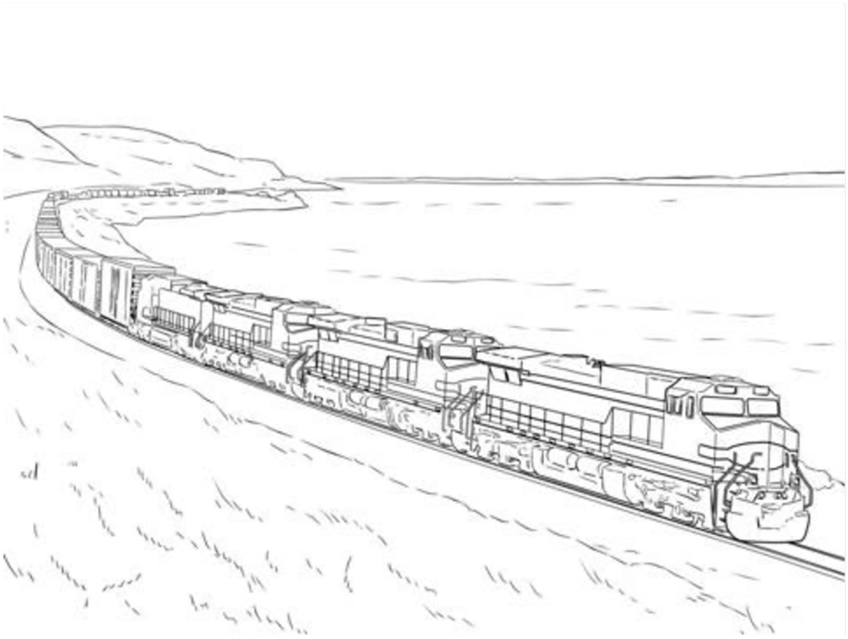


A Layout Control System for Model Railroads

Helmut Fieres

December 24, 2024

Contents

1 Introduction

Model railroading. A fascinating hobby with many different facets. While some hobbyist would just like to watch trains running, others dive deeper into parts of their hobby. Some build a realistic scenery and model a certain time era with realistic operations. Others build locos and rolling equipment from scratch. Yet others enjoy the basic benchwork building, electrical aspects of wiring and control. They all have in common that they truly enjoy their hobby.

This little book is about the hardware and software of a layout control system for managing a model railroad layout. Controlling a layout is as old as the hobby itself. I remember my first model railroad. A small circle with one turnout, a little steam engine and three cars. Everything was reachable by hand, a single transformer supplied the current to the locomotive. As more turnouts were added, the arm was not long enough any more, simple switches, electrical turnouts and some control wires came to the rescue. Over time one locomotive did not stay alone, others joined. Unfortunately, being analog engines, they could only be controlled by electric current to the track. The layout was thus divided into electrical sections. And so on and so on. Before you know it, quite some cabling and simple electrical gear was necessary.

Nearly four decades ago, locomotives, turnouts, signals and other devices on the layout became digital. With growing sophistication, miniaturization and the requirement to model operations closer and closer to the real railroad, layout control became a hobby in itself. Today, locomotives are running computers on wheels far more capable than computers that used to fill entire rooms. Not to mention the pricing. Turnout control and track occupancy detection all fed into a digital control system, allowing for very realistic operations.

The demands for a layout control system can be divided into three areas. The first area is of course **running** locomotives. This is what it should be all about, right? Many locomotives need to be controlled simultaneously. Also, locomotives need to be grouped into consists for large trains, such as for example a long freight train with four diesel engines and fifty boxcars. Next are the two areas **observe** and **act**. Track occupancy detection is a key requirement for running multiple locomotives and knowing where they are. But also, knowing which way a turnout is set, the current consumption of a track section are good examples for layout observation. Following observation is to act on the information gathered. Setting turnouts and signals or enabling a track section are good examples for acting on an observation.

Running, observing an acting requires some form of **configurations** and **operations** What used to be a single transformer, some cabling and switches has turned into computer controlled layout with many devices and one or more bus systems. Sophisticated layouts need a way to configure the locomotives, devices and manage operations of layouts. Enter the world of digital control and computers.

After several decades, there is today a rich set of product offerings and standards available. There are many vendors offering hardware and software components as well as entire systems. Unfortunately they are often not compatible with each other. Furthermore, engaged open software communities took on to build do it yourself systems more or less compatible with vendors in one or the other way. There is a lively community of hardware and software designers building hardware and software layout control systems more or less from scratch or combined using existing industry products.

1.1 Elements of a Layout Control System

Before diving into concept and implementation details, let's first outline what is needed and what the resulting key requirements are. Above all, our layout control system should be capable to simultaneously run locomotives and manage all devices, such as turnouts and signals, on the layout. The system should be easy to expand as new ideas and requirements surface that need to be integrated without major incompatibilities to what was already built.

Having said that, we would need at least a **base station**. This central component is the heart of most systems. A base station needs to be able to manage the running locomotives and to produce the DCC signals for the track where the running locomotive is. There are two main DCC signals to generate. One for the main track or track sections and one for the programming track. This is the track where a locomotive decoder can be configured. A base station could also be the place to keep a dictionary of all known locomotives and their characteristics. In addition to interfaces to issues commands for the running locomotives, there also need to be a way to configure the rolling stock.

Complementing the base station is the **booster** or **block controller** component that produce the electrical current for a track section. The booster should also monitor the current consumption to detect electrical shortages. Boosters comes in several ranges from providing the current for the smaller model scales as well as the larger model scales which can draw quite a few amps. There could be many boosters, one for each track section. The base station provides the signals for all of them.

The **cab handheld** is the controlling device for a locomotive. Once a session is established, the control knobs and buttons are used to run the locomotive. Depending on the engine model, one could imagine a range of handhelds from rather simple handhelds just offering a speed dial and a few buttons up to a sophisticated handheld that mimics for example a diesel engine cab throttle stand.

With these three elements in place and a communication method between them, we are in business to run engines. Let's look at the communication method. Between the components, called nodes, there needs to be a **communication bus** that transmits the commands between them. While the bus technology itself is not necessarily fixed, the messaging model implemented on top is. The bus itself has no master, any node can communicate with any other node by broadcasting a message, observed by all other nodes. Events that are broadcasted between the nodes play a central role. Any node can produce events, any node can consume events. Base station, boosters and handhelds are just nodes on this bus.

But layouts still need more. There are **signals**, **turnouts** and **track detectors** as well as **LEDs**, **switches**, **buttons** and a whole lot more things to imagine. They all need to be connected to the common messaging bus. The layout control system needs to provide not only the hardware interfaces and core firmware for the various device types to connect, it needs to also provide a great flexibility to configure the interaction between them. Pushing for example a button on a control field should result in a turnout being set, or even a set of turnouts to guide a train through a freight-yard and so on.

Especially on larger layouts, **configuration** becomes quite an undertaking. The **configuration model** should therefore be easy and intuitive to understand. The elements to configure should all follow the same operation principles and be extensible for specific functions. A computer is required for configuration. Once configured however, the computer is not required for operations. The capacity, i.e. the number of locomotives, signals, turnouts and other devices managed should be in the thousands.

Configuration as well as operations should be possible through sending the defined messages as well as a simple ASCII commands send to the base station which in turn generates the messages to broadcast via the common bus. A computer with a graphical UI would connect via the USB serial interface using the text commands.

1.2 Standards, Components and Compatibility

The DCC family of standards is the overall guiding standard. The layout system assumes the usage of DCC locomotive decoder equipped running gear and DCC stationary decoder accessories. Beyond this set of standards, it is not a requirement to be compatible with other model railroad electronic products and communication protocols. This does however not preclude gateways to interact in one form or another with such systems. Am example is to connect to a LocoNet system via a gateway node. Right now, this is not in scope for our first layout system.

All of the project should be well documented. One part of documentation is this book, the other part is the thoroughly commented LCS core library and all software components built on top. Each lesson learned, each decision taken, each tradeoff made is noted, and should help to understand the design approach taken. Imagine a fast forward of a couple of years. Without proper documentation it will be hard to remember how the whole system works and how it can be maintained and enhanced.

With respect to the components used, it uses as much as possible off the shelf electronic parts, such as readily available microcontrollers and their software stack as well as electronic parts in SMD and non-SMD form, for building parts of the system. The concepts should not restrict the development to build it all from scratch. It should however also be possible to use more integrated elements, such as a controller board and perhaps some matching shields, to also build a hardware module.

1.3 This Book

This book will describe my version of a layout control system with hardware and software designed from the ground up. The big question is why build one yourself. Why yet another one? There is after all no shortage on such systems readily available. And there are great communities out there already underway. The key reason for doing it yourself is that it is simply fun and you learn a lot about standards, electronics and programming by building a system that you truly understanding from the ground up. To say it with the words of Richard Feynman

"What I cannot create, I do not understand. – Richard Feynman"

Although it takes certainly longer to build such a system from the ground up, you still get to play with the railroad eventually. And even after years, you will have a lay out control system properly documented and easy to support and enhance further. Not convinced? Well, at least this book should be interesting and give some ideas and references how to go after building such a system.

1.4 Parts and Chapters

The book is organized into several parts and chapters. The first chapters describe the underlying concepts of the layout control system. Hardware modules, nodes, ports and events and their interaction are outlined. Next, the set of message that are transmitted between the components and the message protocol flow illustrate how the whole system interacts. With the concepts in place, the software library available to the node firmware programmer is explained along with example code snippets. After this section, we all have a good idea how the system configuration and operation works. The section is rounded up with a set of concrete programming examples.

Perhaps the most important part of a layout control system is the management of locomotives and track power. After all, we want to run engines and play. Our system is using the DCC standard for running locomotives and consequently DCC signals need to be generated for configuring and operating an engine. A base station module will manage the locomotive sessions, generating the respective DCC packets to transmit to the track. Layouts may consist of a number of track sections for which a hardware module is needed to manage the track power and monitor the power consumption. Finally, decoders can communicate back and track power modules need to be able to detect this communication. Two chapters will describe these two parts in great detail.

The next big part of the book starts with the hardware design of modules. First the overall outline of a hardware module and our approach to module design is discussed. Building a hardware module will rest on common building blocks such as a CAN bus interface, a microcontroller core, H-Bridges for DCC track signal generation and so on. Using a modular approach the section will describe the building blocks developed so far. It is the idea to combine them for the purpose of the hardware module.

With the concepts, the messages and protocol, the software library and the hardware building blocks in place, we are ready to actually build the necessary hardware modules. The most important module is the base station. Next are boosters, block controllers, handhelds, sensor and actor modules, and so on. Finally, there are also utility components such as monitoring the DCC packets on the track, that are described in the later chapters. Each major module is devoted a chapter that describes the hardware building blocks used, additional hardware perhaps needed, and the firmware developed on top of the core library specifically for the module. Finally, there are several appendices with reference information and further links and other information.

1.5 A final note

A final note. "Truly from the ground up" does not mean to really build it all yourself. As said, there are standards to follow and not every piece of hardware needs to be built from individual parts. There are many DCC decoders available for locomotives, let's not overdo it and just use them. There are also quite powerful controller boards along with great software libraries for the micro controllers, such as the CAN bus library for the AtMega Controller family, already available. There is no need to dive into all these details.

The design allows for building your own hardware just using of the shelf electronic components or start a little more integrated by using a controller board and other breakout boards. The book will however describe modules from the ground up and not use controller boards or shields. This way the principles are easier to see. The appendix section provides further information and links on how to build a system with some of the shelf parts instead of building it all yourself. With the concepts and software explained, it should not be a big issue to build your own mix of hardware and software.

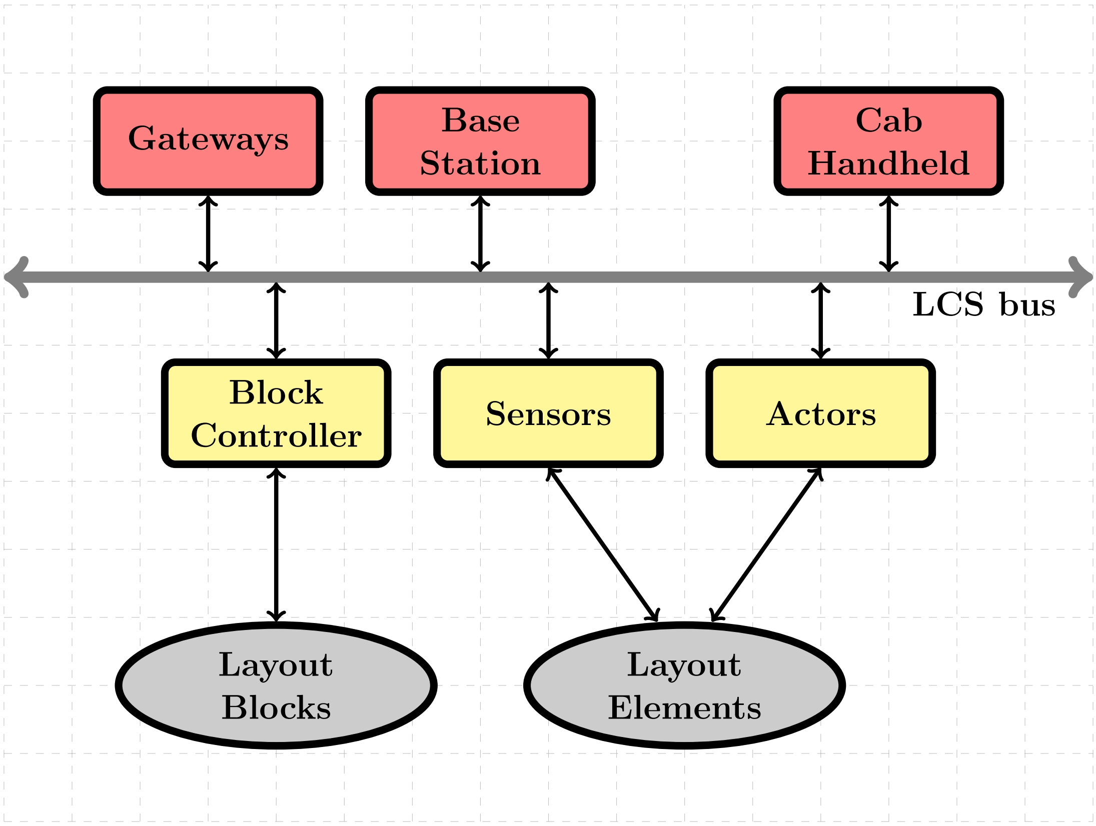
I have added most of the source files in the appendix for direct reference. They can also be found also on GitHub. (Note: still to do...) Every building block schematic shown was used and tested in one component or another. However, sometimes the book may not exactly match the material found on the web or be slightly different until the next revision is completed. Still, looking at portions of the source in the text explain quite well what it will do. As said, it is the documentation that hopefully in a couple of years from now still tells you what was done so you can adapt and build upon it. And troubleshoot.

The book hopefully also helps anybody new to the whole subject with good background and starting pointers to build such a system. I also have looked at other peoples great work, which helped a lot. What I however also found is that often there are rather few comments or explanations in the source and you have to partially reverse engineer what was actually build for understanding how things work. For those who simply want to use an end product, just fine. There is nothing wrong with this approach. For those who want to truly understand, it offers nevertheless little help. I hope to close some of these gaps with a well documented layout system and its inner workings.

In the end, as with any hobby, the journey is the goal. The reward in this undertaking is to learn about the digital control of model railroads from running a simple engine to a highly automated layout with one set of software and easy to build and use hardware components. Furthermore, it is to learn about how to build a track signaling system that manages analog and digital engines at the same time. So, enjoy.

2 General Concepts

At a higher level, the layout control system consists of components and a communication scheme. This chapter will define the key concepts of a layout system. At the heart of the layout control system is a common communication bus to which all modules connect. The others key elements are node, events, ports and attributes. Let's define these items first and then talk about how they interact. The following figure depicts the high level view of a layout control system.



2.1 Layout Control Bus

The layout control bus is the backbone of the entire system. The current implementation is using the industry standard CAN bus. All hardware modules connect to this bus and communicate via messages. All messages are broadcasted and received by all other hardware modules on the bus. The classic CAN bus standard limits the message size to 8 bytes and this is therefore the maximum message size chosen for the LCS bus. The CAN bus also has a hardware module limit of about 110 modules for bandwidth reasons. But even for a large layout this should be sufficient. And for really large layouts, another bus system or a system with CAN bus routers, could be envisioned. The software should therefore be designed to manage thousands of connected modules. While the CAN bus technology could be exchanged, the message format and size defined as well as the broadcasting paradigm are fixed in the overall design and will not change.

2.2 Hardware Module

Everything connected to the LCS bus is a **hardware module**, which is the physical entity connected to the bus. Typically it is a micro controller with the bus interface and hardware designed for the specific purpose. For example, a CAN bus interface, an AtMega Controller, and digital output drivers could form a hardware module to control railroad turnouts and signals. Base stations, handhelds and gateways are further examples of a hardware module. Hardware modules are expected to be physically located near their use and thus spread throughout the layout. Some hardware modules could be at locations that cannot be reached easily. So all interaction for configuration and operations needs to be possible through the messages on the bus. Nevertheless, putting local controls on a hardware module should not be prohibited.

A hardware module consists of a controller part and a node specific part. The controller part is the **main controller**, which consists of the controller chip, a non-volatile memory to retain any data across power down, a CAN bus interface and interfaces to the node specific hardware. The node specific hardware is called the **node extension**. Conceptually, both parts can be one monolithic implementation on one PCB board, but also two separate units connected by the extension connector. The are defined connectors between the boards. The hardware chapter will go into more detail on the board layouts and hardware design options.

2.3 Nodes

A hardware module is the physical implementation. A **node** is the software entity running in the firmware of the hardware module. Nodes are the processing elements for the layout. Conceptually, a hardware module can host more than one node. The current implementation however supports only one node on a given hardware module. A node is uniquely identified through the **node identifier**. There are two ways to set a nodeId. The first is to have central component to assign these numbers on request. The second method sets the number manually. Although a producer consumer scheme would not need a nodeId, there are many operations that are easier to configure when explicitly talking to a particular node. Both nodes and event identifiers are just numbers with no further classification scheme. A configuration system is expected to provide a classification grouping of nodes and event number ranges if needed.

A node also has a **node type**, to identify what the node is capable of. Examples of nodes types are the base station, a booster, a switch module, a signal control module, and so on. While the node number is determined at startup time and can change, the node type is set via the module firmware. As the node type describes what the hardware module can do the type cannot change unless the module changes. Once the node has an assigned node number, configuration tools can configure the node via configuration messages to set the respective node variables.

A node needs to be configured and remember its configuration. For this purpose, each node contains a **node map** that keeps all the information about the node, such as the number of ports, the node unique Id and so on. There is also a small set of user definable attributes to set data in a node map specific to the node. The data is stored in non-volatile memory space and on power up the node map is used to configure the node. If the module is a new module, or a module previously used in another layout, or the firmware version requires a new data layout of the node map, there is a mechanism to assign a new node number and initialize the node map with default values.

2.4 Ports

A node has a set of receiving targets, called ports. Ports connect the hardware world to the software world, and are the connection endpoints for events and actions. For example, a turnout digital signal output could be represented to the software as a port on a node. The node registers its interest in the event that target the signal. An event sent to the node and port combination then triggers a callback to the node firmware to handle the incoming events. Although a node can broadcast an event anytime by just sending the corresponding message, the event to send is typically associated with an outbound port for configuration purposes. In addition to the event immediate processing, the event handling can be associated with a timer delay value. On event reception the timer value will delay the event callback invocation or broadcast.

A node has a **port map** that contains one entry for each defined port. **port map entries** describe the configuration attributes and state of the port such as the port type. There is also a small set of user definable attributes to set data in a port map entry specific to the port. These attributes can be used by the firmware programmer to store port specific data items such as a hardware pin or a limit value in the port map.

2.5 Attributes

Node attributes and **port attributes** are conceptually similar to the CV resources in a DCC decoder. Many decoders, including the DCC subsystem decoders, feature a set of variables that can be queried or set. The LCS layout system implements a slightly different scheme based on items. In contrast to a purely decoder variable scheme an item can also just represent just an action such as setting an output signal. Items are passed parameter data to further qualify the item. Items are just numbers assigned. The range of item numbers is divided into a reserved section for the layout system itself, and a user defined range that allows for a great flexibility to implement the functions on a particular node and port. The meaning of user defined items is entirely up to the firmware programmer. If it is desired to have a variables, a combination of items and attributes can provide the traditional scheme as well. In addition, there are node local variables, called attributes, available to the firmware programmer for storing data items.

2.6 Events

The LCS message bus, hardware module, node and ports describe layout and are statically configured. For nodes to interact, **events** and their configuration is necessary. An event is a message that a node will broadcast via the bus. Every other node on this bus will receive the event and if interested act on the event. The sender is the producer, the receiver is the consumer. Many producers can produce the same event, many consumers can act on the same event. The **event Id**, a 16 bit number, is unique across the layout and assigned by a configuration tool during the configuration process. Other than being unique, there is no special meaning, the number is arbitrary. There are in total 65536 events available.

In addition to the event Id, an event message contains the node Id of the sender. While most events will be an ON/OFF event, events can also have additional data. For example an overload event sent by a booster node, could send the actual current consumption value in the event message. A consumer node registers its interest in an event by being configured to react to this event on a specific port. The node maintains an **event map**, which contains one entry for each event id / port id combination. For the eventing system to work, the nodeID is not required. Any port on any node can react to an event, any node can broadcast an event.

To connect producers to consumers, both parties need to be told what to do with a defined event. A producer node outbound port needs to be told what event to send for a given sensor observation. For example, a simple front panel push button needs to be told what event to send when pushed. Likewise, a consumer node inbound port needs to be told what events it is interested in and what the port should do when this event is received. Both meet through the event number used. While an inbound port can be configured to listen to many event Ids, an outbound port will exactly broadcast one eventId.

Any port on any node can react to an event, any node can broadcast an event. Still, addressing a node and port combination explicitly is required for two reasons. The first is of course the configuration of the node and port attributes. Configuration data needs to go directly to the specified node and port. The second reason is for directly accessing a resource on the layout. For example, directly setting a turnout connected to one node. While this could also be implemented with associated an event to send when operating a turnout, it has shown beneficial and easier to configure also directly access such a resource through a dedicated node/port address.

2.7 DCC Subsystem

The node, ports and events are the foundation for building a layout system based on the producer / consumer scheme. The scheme will be used heavily for implementing turnout control, signals, signal blocks and so on. In addition, there is the management of the mobile equipment, i.e. locomotives. The DCC subsystem is the other big part of our layout control system. In a sense it is another bus represented by the track sections.

LCS messages for DCC commands are broadcasted from controlling devices. For example, a handheld broadcasts a speed setting DCC command. In a layout there is one base station node which is responsible to produce the DCC signals for the track. The DCC signals are part of the physical LCS bus. While a base station design could directly supply the signal current to the track, larger layouts will typically have one or more boosters. They take the DCC signal from the LCS bus lines and generate the DCC signal current for their track section. All LCS messages for DCC operations are broadcasting messages, all nodes can send them, all nodes can receive them. Handhelds, base station and boosters are thus just nodes on the LCS bus. Only the base station will however generate the DCC signal.

The DCC standard defines mobile and stationary decoders. The DCC signal could also be used to control for example a set of turnouts via a stationary decoder. The LCS DCC message set contains messages for addressing a stationary decoder. Since the commands for stationary equipment are just DCC commands, they will be transmitted via the track as well and take away bandwidth on the track. A layout will therefore more likely use the LCS bus for implementing the management of stationary equipment. Besides, the producer / consumer model allows for a much greater flexibility when building larger and partially automated layouts.

2.8 Analog Subsystem

The layout control system is primarily a digital control system. There are however layout use cases where there are many analog locomotives that would represent a significant investment when converting to DCC or that cannot easily be equipped with a DCC decoder. In a DCC subsystem the decoder is in the locomotive and many locomotives can run therefore on the same track. In an analog system, the locomotive has no capabilities and therefore the track needs to be divided into sections that can be controlled individually. One locomotive per section is the condition. In a sense the decoder becomes part of the track section. The layout control system offers support for building such a track section subsystem. Often the sections are combined into blocks and build the foundation for a block signaling system. Note that the rest of the layout control system is of course digital. What is typically the booster to support a section of track, is the block controller for an analog layout. We will see in the later chapters that booster and block controller are very similar and design a block controller to accommodate both use cases.

2.9 Configuration Mode

Before operations the nodes, ports and events need to be configured. Once a node has an assigned valid nodeId, the node configuration is the process of configuring a node global information, the event map information and the finally the port information. The information is backed by non-volatile storage, such that there is a consistent state upon node power up. During operations, these value can of course change, but are always reset to the initial value upon startup.

The primary process of configuration is inventing events numbers and assigning them producers and consumers. The process follows the general "if this then that" principle. On the producer side the configuration process assigns a port to an event, i.e. the push of a button to an event to send. If this button is pushed then send that event. On the consumer side the configuration process is to assign the event to a port. If this event is received then execute that port action.

After the node is up and running with a valid node Id, there are event configuration messages than can be send to the node to set the event mapping table with this information. The event map table is the mapping between the event and the port associated. Events are thus configured by "teaching" the target node what port to inform about an occurring event.

2.10 Operation Mode

Besides the basic producer/consumer model with the event messages as communication mechanism, there are several LCS control and info messages used for managing the overall layout with signals turnouts and so on as well as the physical track and the running equipment. In a layout, the track typically consist of one or more sections, each managed by a booster or block controller node. Track sections are monitored for their power consumption to detect short circuits. Back communication channels such as RailCom are handled by the booster node and provide information about the running equipment. Stationary equipment such as turnouts and signals as well as detectors, such as track occupancy detectors or turnout setting detectors are monitored and controlled through LCS messages and the event system. Conceptually any node can send and receive such event, info or control messages. Some nodes, however have a special role.

For example, the key module for layout operations is the **base station**. The base station, a node itself, is primarily responsible for managing the active locomotives on the layout. When a control handheld wants to run a locomotive, a cab session for that locomotive is established by the base station. Within the session, the locomotive speed, direction and functions are controlled through the cab handheld sending the respective messages. The base station is responsible for generating the DCC packets that are sent by the booster or block controller power module to the actual track sections. Booster and block controller module are - you guessed it - node themselves.

Finally, there are LCS nodes that represent cab handhelds to control a locomotive or consists, layout panel connectors, gateways to other layout protocols, sensors and actors to implement for example turnout control, signaling, section occupancy detections and many more. All these components share the common LCS bus and use ports and events to implement the capabilities for operating a layout.

In a layout with many track sections the **block controller** is a special node that will manage a block on the layout. Like all other nodes, a block controller itself is a node that can react to events and is controller and monitored by LCS messages. There will be several chapters devoted to this topic later.

2.11 Summary

This chapter introduced the basic concepts of the layout control system described in this book. It follows very few overall guiding principles. Above all, there is the clear separation of what needs to be available for operating the mobile equipments, i.e. locomotives, and the stationary layout elements. Controlling mobile decoders are left to the DCC subsystem, all other communication takes place via the LCS bus, which is the bus to which all of the hardware modules connect. Hardware modules host the nodes. Currently, a hardware module hosts exactly one node. A node can contains one or many ports, which are the endpoints for the event system. There is a set of user allocated attributes available to node and ports. Node, port and attribute data are backed by non-volatile memory, so that a restart will use defined initial values. Nodes and their ports are also directly addressable, which is needed for configuration purposes and the directly addressable components model. Using the producer / consumer paradigm, sensors generate events and interested actors just act on them. The configuration process is simply to assign the same event to the producer node and consumer node / port id when they should work together.

The communication bus should rest on a reliable bus with a sufficient bandwidth. Although the CAN bus is used in the initial implementation, it is just one option and other technologies can be considered. In all cases however, the message format should be available for a variety of bus technologies. Our messages are therefore short, up to eight data bytes. This causes on the one hand some complexity for data items larger than a few bytes on the other hand no messages blocks the bus for a longer period. The bus technology is expected to reliably deliver a message but does not ensure its processing. This must be ensured through a request reply message scheme built on top.

3 LCS Hardware Module Design

So far we covered the general concepts, messages, protocols as well as the LCS core library and a glimpse how all of this might be used. Let's take a break from all that concepts and mostly software talk. For the software to run, hardware modules need to be built. Welcome to the next big part of this book. Here, we will talk about the lCS hardware modules. A hardware module conceptually consist of three key parts.

- communication
- controller
- function block(s)

At the center of a hardware module is the **controller**. There is a great variety of controllers and development environments available. When selecting a controller for LCS, we will talk in a minute which one was picked, its is important that there is enough CPU power and equally important a powerful development environment. A console command line interface and interfaces to load the software is also very handy for configuring, monitoring and debugging. The **communication** part implements at a minimum the LCS message bus interface for the messages to transmit between the modules. Finally, the **function block**s implement the hardware module specific capabilities.

This chapter is the first in series of chapters on hardware modules. Instead of presenting complete schematics for each major hardware module, such as the base station, we will go a slightly different route. We will first present the basic components an LCS node might need. Definitively we will need a controller and a CAN bus interface. Some LCS nodes might make use of an extended non-volatile storage, others need plenty of digital outputs. Just like Lego Blocks, all these parts should be combined easily to form the desired LCS hardware module. We will tackle each component one at a time to understand how they work. The later chapters will just combine these basic blocks with minor adaptations and perhaps some very dedicated components for their functionality.

3.1 Selecting the controller

The module designs described in this book initially used the AtMega controller platform along with the Arduino IDE to write the software. There is the Arduino IDE and by now a whole set of different processors. Since it was released, the Atmega controller family and boards such as Arduino UNO, Arduino NANO, Arduino MEGA are in widespread use. The LCS core library program and non-volatile storage requirements do place however a higher demand on the controller capabilities.

Meanwhile, the Raspberry PI Pico (`PICO`) controller joined the club. And it has a lot to offer. The PICO is a dual core controller running at up to 133 Mhz. It features a whopping 16Mbytes of flash and 264 Kbytes of main memory. There are plenty of

IO ports, and functional blocks for UARTS, SPI and I2C interfaces. What makes this controller especially interesting are the PIO state machines that allow for implementing your own I/O protocols. There is CAN bus software library built using these state machines. This way no extra CAN bus controller is needed. The PICO comes with its own software development kit and also an Arduino IDE integration is available.

As time goes by, there will be for sure other capable controller entering the market. However, when you want to complete a project versus chasing the latest controllers, you will need to pick. In our case, the PICO is the controller of choice. Its capabilities match our requirements and will be a good choice for the years to come. nevertheless, the LCS library software should be designed as independent of a particular controller as possible. More on this later.

3.2 The Controller Platform

The following table gives some guidance on the capabilities needed in our designs. This list also applies in general to other controllers.

Table 3.1: Controller Attributes

Attributes	Notes
Processor	For a typical module, the PICO offers plenty in terms of CPU power. Since we use a software implementation for the CAN bus, running the software in one core and the CAN bus state machine in the other will well match what the PICO offers.
Memory	Memory depends on the size requirements of the node, port and event maps and the node-specific firmware data demands. A simple module would perhaps get by with 2Kb, a base station could easily use 32Kb or even more.
Program Memory	The LCS library already uses round about 64Kb of code storage. A simple module would get by with 32Kb, a base station could easily use 128Kb and more.
External NVM	Additional NVM storage is allocated in a separate EEPROM or FRAM. The capacity is highly dependent on the module use case. External NVM components typically also require the SPI or I2C interface. Most external EEPROM chips have write cycles of more than a million. At a minimum, a chip size of 32Kb is recommended. The PICO does not offer an internal EEPROM, so an external NVM is always required.
Digital channels	The bulk of control lines is digital and used heavily. For some hardware modules, a subset of the digital pins should also be PWM capable.

Continued on next page

Attributes	Notes
Analog channels	Analog input is typically used for the power module for analog voltage measurements. Otherwise, it is perhaps optional. The PICO allows for only three inputs. If more are desired, an external multiplexer needs to be implemented.
I2C	The I2C interface comes in very handy to connect a large variety of chips. Communication to the external NVM and also to chips that implement functions such as a servo controller will require this bus.
Serial I/O	The serial I/O is used in some hardware modules for implementation of RailCom detectors. The PICO features two hardware UARTS and the option to implement more in software using the PIO state machines.
Console I/O	Serial I/O is used for console I/O. Rather than using dedicated I/O pins and a UART block in the controller, the PICO serial I/O will be implemented via the USB connector.
LEDs, Button and Dip Switches	A hardware module could make use of LEDs to indicate readiness and activity, as well as a set of switches to configure a hardware option. Not really required but certainly useful.
WLAN	WLAN is optional. But there is a PICO version with WLAN capability integrated.

3.3 Hardware Module Schematics

Hardware modules are described to large extent via schematics. The schematics shown in the following chapters are all drawn with the EasyEDA software. It is a great hardware development platform, and you can order PCBs for the final design in one easy step. Following a building block principle, the schematic diagrams will show functional components with many network endpoints where they connect to other building blocks. Each network endpoint is labelled with a name that is unique across all building blocks used in a hardware module schematic drawn. For example, ”VCC-3V3” will always refer to the 3.3V power supply line. If two building blocks have an endpoint with the same name, the endpoints will be connected on all building block schematics in the final hardware module design.

A general word to the building blocks. They serve as examples of how the individual parts could be implemented and help to understand how each part works. Parts of the library software assume the presence of these blocks and how they basically work. Although the library has been written with as much as possible independence of the hardware, the final adaption of timers, serial lines, I/O pins and so on is required needs to be considered. Throughout the next chapters, you will find comments on what is perhaps generic and what would require some adaption if moving to another processor family.

3.4 Controller and Extension Board

Each node in the layout control system is a node and hence there is a controller for running the node firmware. Without a question, there will be many different nodes and as time goes by perhaps even a new controller families. However, each node would need at least some form of power supply, the CAN bus interface and depending on the storage demands and controller family, an external NVM. On top there is the node specific hardware. One approach is to design a board for each dedicated purpose. This board would include all the common portion for a LCS node and the hardware module specific portion. Another approach is to design a node controller board with extension boards that can be connected to it. In the remainder of this chapter, we will describe the main controller and extension concept. However, it is also perfectly all-right to design a hardware component with all the components integrated on one board. For a complex node such as the base station, this is a very reasonable solution. The building blocks shown in this chapter thus also form the basis for a more monolithic hardware module design. But first, let's look at the physical dimension of our boards.

picture

All boards will have a form factor of 10cm wide and 8, 12, and 16cm long. In particular, the 10x16cm board should be very familiar. It has the "Euro PCB" dimensions. The main controller board has on the left side the connectors for the LCS bus and the power input. On the right side, there are two connectors toward an extension board. As described before, there are two types of extension boards. The usage of the individual connector pins are described in the upcoming chapter. To ease the hardware schematic development and ensure that all boards fit together, the PCB boards along with their connectors are available as symbols and PCB footprints in the EasyEDA library.

3.5 LCS Bus connector

Every hardware module needs the LCS bus interface to connect to the bus. Some modules may also draw power from this bus. The modules use an RJ45 connector for connecting to the bus. The bus signals can be grouped in several categories. The CAN bus differential lines represent the CAN bus. The VS line is intended for hardware modules with very little power consumptions so that they can directly be powered by the bus. The DCC signal lines are an exact copy of the DCC signal that would go to a track sent out by the DCC signal generating base station. The signal is intended to be routed from the base station to booster nodes, but also to hardware modules that analyze the DCC signal for some action. Finally there is the STOP signal line. This is a wired OR line that allows a simple button along the layout with access to this line to issue a STOP signal. The base station or any nodes interested in the signal can monitor this line. There are the following signal lines.

Table 3.2: Bus Connector Pins

Pin	Name	Purpose
1	DCC-Sig-1	The DCC signal labelled "+"
2	DCC-Sig-2	The DCC signal labelled "-"
3	GND	Common ground
4	RSV	reserved for future extensions.
5	RSV	reserved for future extensions.
6	PWR	The bus supplied 12V power line. This line is intended for devices with very little power consumption to get their power from. Any other module should connect to its own power supply line.
7	CAN-L	Line L of the differential CAN bus signal.
8	CAN-H	Line H of the differential CAN bus signal.

3.6 LCSNodes Extension Board Connector

For interchangeability of extensions, there is a standardized **extension board connector** between controller and extensions. Extension boards come in two flavors. The first will have the extension connector on both sides of the board. Main controller boards and extension board will have a female connector on the right hand side. The first flavor extension board will have a matching connector on the left side. This way main controller and extension boards can be placed next to each other, just like a train. The second type of extension boards only have the connectors on the right hand side. They are intended for a backplane style layout where main controller and extension boards are plugged next to each other. The overall concept is very similar to the the shield concept found in the Arduino or Raspberry PI universe, except that we can stack boards, as well as placing them next to each other.

The I2C interface will be the main communication method between the boards. In fact all current extension boards shown in later chapters use the I2C communication channel. Nevertheless, a rather rich set of outputs from the controller should be available to the extension board for flexibility. There should be ports for digital input and output, analog input, PWM outputs, serial outputs and so on. The raspberry pi pico offers a great flexibility on assigning function blocks such as an SPI or I2C interface to pins. The extension connector outlined below offers a set of pins which are mapped to the PICO capabilities. The following table shows the connector pin assignments for the communication between a main controller board and extension boards. All boards will have a 40-pin connector organized as 2 rows of 20 pins.

Table 3.3: Controller Attributes

Pin	Name	Pin	Name	Purpose
1	**DCC-1**	2	**DCC-2**	The DCC "+" and "-" signal as generated by the DCC Signal Generator. These pins are typically driven by the base station generating the layout DCC signal.
3	**GND**	4	**GND**	Common ground pins.
5	**ADC-0**	6	**ADC-1**	Analog input pins. The input is not protected. The analog voltage range is 0 to VCC.
7	**GND**	8	**GND**	Common ground pins.
9	**DIO-0**	10	**DIO-1**	Plain digital Pins, input or output. The pins are protected.
11	**DIO-2**	12	**DIO-3**	Plain digital Pins, input or output. The pins are protected.
13	**DIO-4**	14	**DIO-5**	Plain digital Pins, input or output. The pins are protected.
15	**DIO-6**	16	**DIO-7**	Plain digital Pins, input or output. The pins are protected.
17	**DIO-8**	18	**DIO-9**	Plain digital Pins, input or output. The pins are protected.
19	**DIO-10**	20	**DIO-11**	Plain digital Pins, input or output. The pins are protected.
21	**GND**	22	**GND**	Common ground pins.
23	**BI-0**	24	**BI-1**	Bus Address input lines. Up to four extension boards can be connected, the BI pins are used to determine the I2C address on the I2C extension bus.
25	**BO-0**	26	**BO-1**	Bus Address output lines. The BO lines are computed form the BI lines. If for example BI is 1:0 the BO lines will become 1:1. The starting output pins values are 1:1.
27	**SCL**	28	**SDA**	I2C extension bus channel. The lines are protected with a serial resistor and there is a pull-up resistor to VCC.

Continued on next page

Pin	Name	Pin	Name	Purpose
29	**RST**	30	**EXT**	RST is reset line. Active Low. EXT is the external interrupt line which be raised from an extension board. Active low.
31	**VCC**	32	**VCC**	VCC 5V supply to extension boards.
33	**GND**	34	**GND**	Common ground pins.
35	**VS**	36	**VS**	Board Input voltage forward. These connector pins are primarily used by extension boards that need the high power input. Examples are H-Bridges on such a board or boards that have their power supply circuitry.
37	**VS**	38	**VS**	Board Input voltage forward.
39	**GND**	40	**GND**	Common ground pins.

The extension board connectors on the main controller boards are female connectors placed on the right hand side of the board. Male connectors are used on an extension board to connect into the main controller or a previous extension board. There are EasyEDA symbols and PCB footprints that offer the connector pins without you going through these details. The appendix contains EasyEDA symbols for the most common board dimensions with the connectors placed in the correct location. A new projects can just start with these EasyEDA symbols.

A key question is how many controller pins are available to an extension board. As said, most of the extension boards would just need the I2C bus to drive the I2C capable ICs on an extension board. However, since there might be rather complex extension boards, the IO pins needed from the controller board to the extension are many and should allow not only for digital IO but also the function blocks inside the controller. The `DIO-x` pins on the connector map to the GPIO pins of the Raspberry Pi PICO in a way that most of the controller capabilities can be used on an extension board. We will discuss this in more detail in the main controller chapter.

For even more complex extension boards, it is perhaps the better idea to combine a main board with an extension board capabilities to one monolithic board but still keep the extension connector for other not so complex extension boards to attach. As a guideline, only the first extension board will benefit from all signals coming from the main controller board. All follow on extension boards will only get the DCC signals, the interrupt and reset line, the I2C signal and the power lines.

3.7 Track Power Connectors

In addition to the extension board connector, there is the **track power connector**. This connector is only used by the base station, block controller and associated extensions. Its purpose is to pass the track power signals from the H-bridges on the base station or block controller (or booster) board to the extension boards. This connector is described in more detail in the base station and block controller chapter.

Table 3.4: Controller Attributes

Pin	Name	Pin	Name	Purpose
1	**DCC-SIG-B0**	2	**DCC-SIG-B0**	Bridge-0 DCC Signal "+" and "-".
3	**DCC-SIG-B1**	4	**DCC-SIG-B1**	Bridge-1 DCC Signal "+" and "-".
5	**DCC-SIG-B2**	6	**DCC-SIG-B2**	Bridge-2 DCC Signal "+" and "-".
7	**DCC-SIG-B3**	8	**DCC-SIG-B2**	Bridge-3 DCC Signal "+" and "-".

When using all four bridge signal output pairs, each each output pair is rated up to 3Amps. For high power bridges with up to 6Amps, two pairs can be combined and the number of bridges signals passed on is two.

3.8 Summary

This chapter introduced the basic ideas behind a hardware module, it connectors and board layout. A key concept is the idea of a common component, the main controller, and extensions that can be connected. Nevertheless, there are good cases for combining a main controller and the extension hardware into one monolithic board. But in any case, the connectors and their purposes stay the same from board to board. While the main controller boards always have the LCS bus and power input on the left side, the extension connector and track line connector on the right, extension boards come in two flavors.

The first extension board type has male connectors for track line and extension lines on the left side of the board while the second type has not. Both types have female track line and extension line connectors on their right. The first type can just be plugged into the main controller type boards, additional extension boards are simply plugged into the previous extension board. The second extension type is intended for a backplane type design where main controller boards as well as up to four extension board types are plugged into a backplane board. Throughout the chapter to come, you will see how easy boards can be combined using the two connectors lanes and standards behind them.

Ready for the first hardware work ? All aboard, the train leaves for the next chapter.

A LCS Nodes and EasyEda

The schematics and boards shown were all developed using the EasyED software. EasyEDA is a design tool for developing the schematics and PCB layouts. A PCB can then be ordered at very reasonable prices. Even during LCS node early design stages it is therefore sometimes worthwhile to just produce a PCB and avoid searching software bugs that are actually just loose connection on a breadboard. To ease the development, there are experimental boards. However when it comes to a final design, PCB boards need to be developed and ordered in larger quantitates. The LCS Node design introduced contains a main controller board and extension boards. The sizes and location of the connectors have been standardized. This appendix contains the PCB drawings of the most common LCS boards to give you a head start in developing your own boards, ensuring that all boards fit together.

A.1 Symbols and Footprints

EasyEDA allows you to create symbols that represent components and can be placed in a schematic. To each symbol there should be a footprint that is used to put the component on to the PCB. The connection between the two is a list of assignments that associate a **pin** on the symbol with a **pad** on the footprint. For LcsNodes there is a list of symbols and footprints to ensure that the PCBs do have all their connectors at the exact place, so that they fit together.

A.1.1 Symbols

To ease the development of LCS boards, the entire board and its connectors are available as a symbol. Depending on the category, the symbol features the connection end points for the connectors found on the board. This symbol is associated with the corresponding footprint described in the next section. Note that the footprint needs to match the symbol. That is the number, position and meaning of the connectors found on the board map, only length of the PCB board varies.

A.1.2 Main Controller Board Footprints

This section contains all the footprints available so far. There are three main categories. The first is anything that represents an LCS Controller portion. There are the connections to the LCS bus and the power input connector. On the left side are two connectors. The upper connector is reserved for up to four tack pow lines. Below is the LCS extension board connector. The basic LCS Main Controller Board for example is the 16cm x 10cm board shown below.

The mounting holes may look a little odd. As shown in the text to follow, there are extension boards with a form factor of 12cm x 10cm. When are the are mounted on top of the 16cm board, the holes nicely match.

A.2 Extension Boards Footprints

Next, there are the extension boards. They are straightforward and just offer the two connector lines on the right and optional on the left. Just the length varies. Boards with a connectors on the left and right are boards that can be just connected a main controller board or another extension board.

A.3 Footprints for 12cm x 10cm boards

In addition to the basic 16cm x 10cm form factor, is a set of 12cm x 10cm boards. They have exactly the same layout, except that their tsRoundedRectangle length is 12cm.

As always, there could be many more combinations as new boards with different demands are developed. Nevertheless it is important that when connectors are used, that they have the same meaning and are placed at the same location. This is the whole idea of using footprints to ensure this exact fitting.

A.4 Links

Table A.1: ...

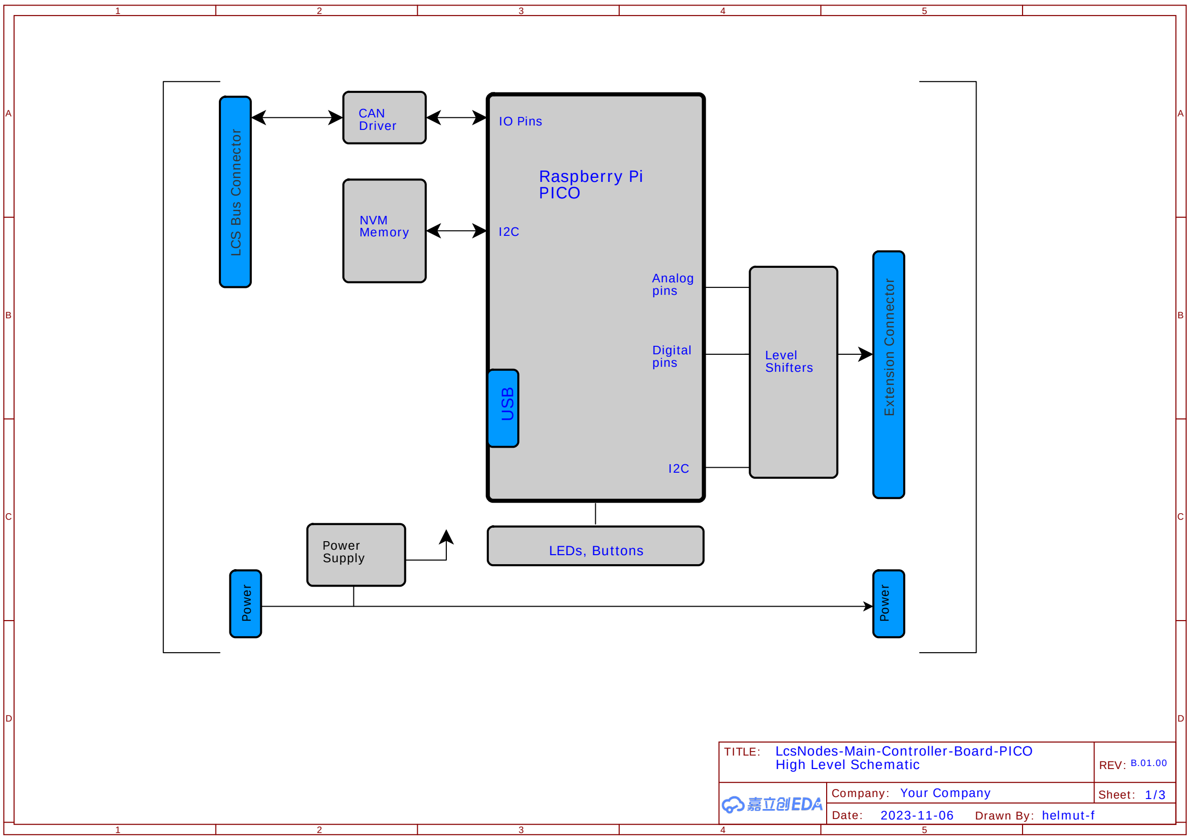
Tool	**Link**	**Comment**
EasyEDA	http://easyeda.com/de	Design tool for schematics and PCB layouts
JLCPCB	-	part of EasyEDA that manufactures PCB boards, order from within EasyEDA

B Tests

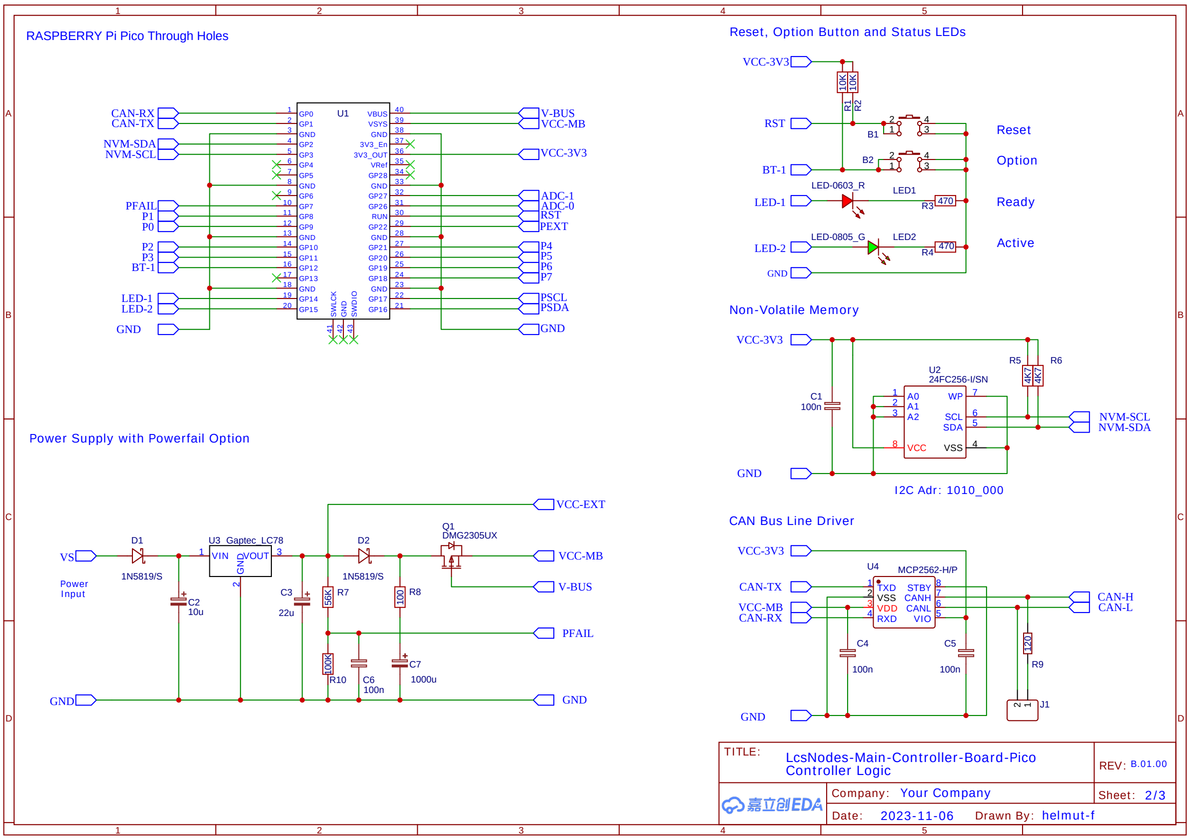
B.1 Schematics

float barrier command to ensure that text stays close to the picture but no text from after the picture.

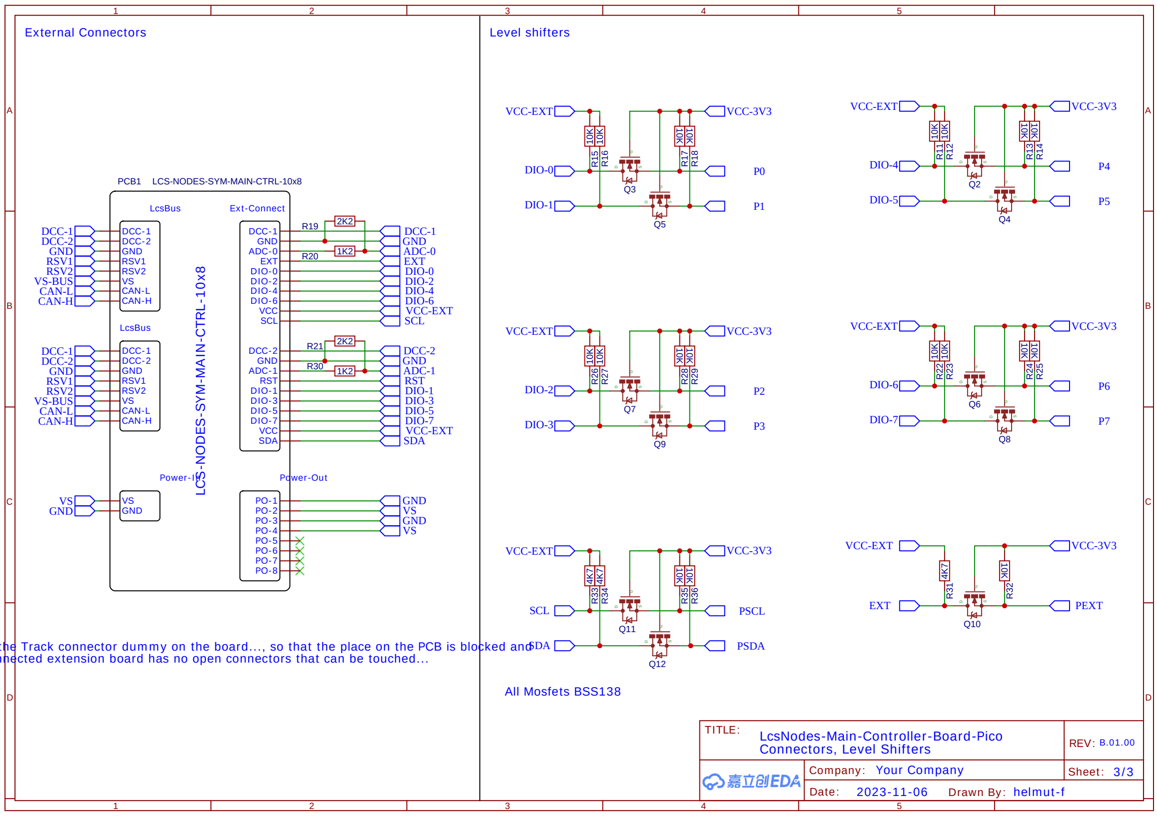
B.1.1 part 1



B.1.2 part 2


RASPBERRY Pi Pico Through Holes
CAN-RX
CAN-TX
NVM-SDA
NVM-SCL
PFAIL
P1
P0
P2
P3
BT-1
LED-1
LED-2
GND
U1
V-BUS
VCC-MB
VCC-3V3
ADC-1
ADC-0
RST
PEXT
P4
P5
P6
P7
PSCL
PSDA
GND
Reset, Option Button and Status LEDs
VCC-3V3
RST
BT-1
LED-1
LED-2
GND
Reset
Option
Ready
Active
LED-0603_R
LED-0805_G
LED1
LED2
R3
R4
470
Non-Volatile Memory
VCC-3V3
GND
U2
24FC256-I/SN
C1
100n
R5
R6
4K7
NVM-SCL
NVM-SDA
I2C Adr: 1010_000
Power Supply with Powerfail Option
VS
Power Input
D1
1N5819/S
U3_Gaptec_LC78
C2
10u
C3
22u
D2
1N5819/S
Q1
DMG2305UX
R7
56K
R8
100
R10
100K
C6
100n
C7
1000u
VCC-EXT
VCC-MB
V-BUS
PFAIL
GND
GND
CAN Bus Line Driver
VCC-3V3
CAN-TX
VCC-MB
CAN-RX
GND
U4
MCP2562-H/P
C4
100n
C5
100n
R9
120
J1
CAN-H
CAN-L
TITLE: LcsNodes-Main-Controller-Board-Pico
Controller Logic
REV: B.01.00
Company: Your Company
Sheet: 2/3
Date: 2023-11-06
Drawn By: helmut-f

B.1.3 part 3



B.2 Lists

B.2.1 A simple list

- First bullet point
- Second bullet point
- Third bullet point

B.2.2 An instruction word layout

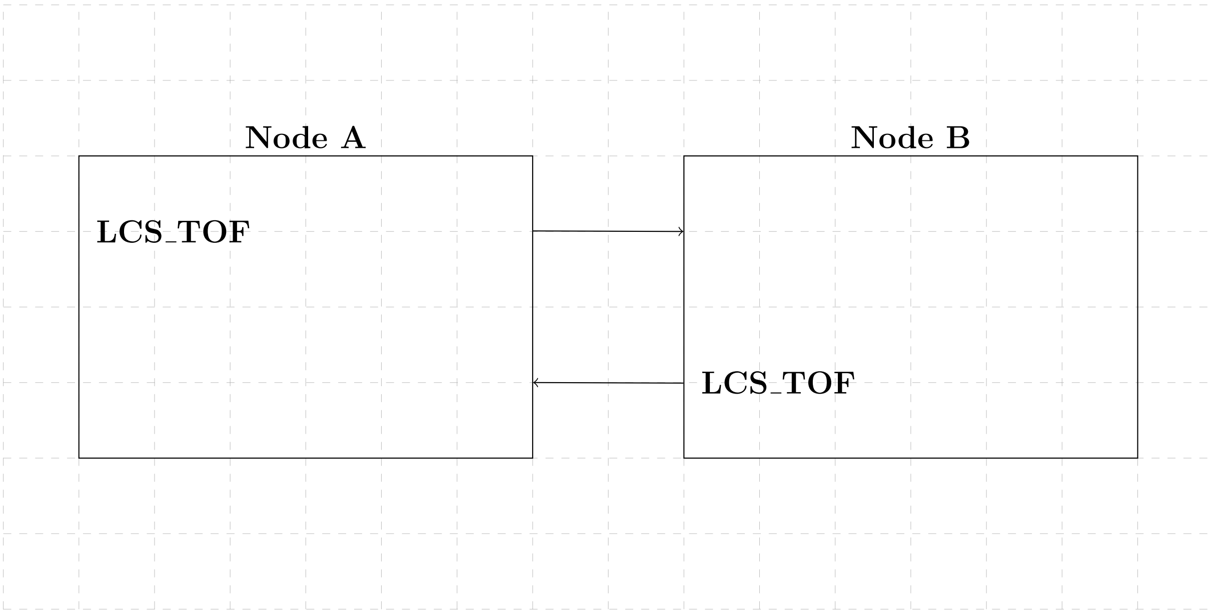
A little test for an instruction word layout ... will be a bit fiddling work ...

1 3 6

Test				

B.3 Protocol boxes

A bit cumbersome and we would need to have text at defined locations. Perhaps keep the simple table in the protocol chapter.

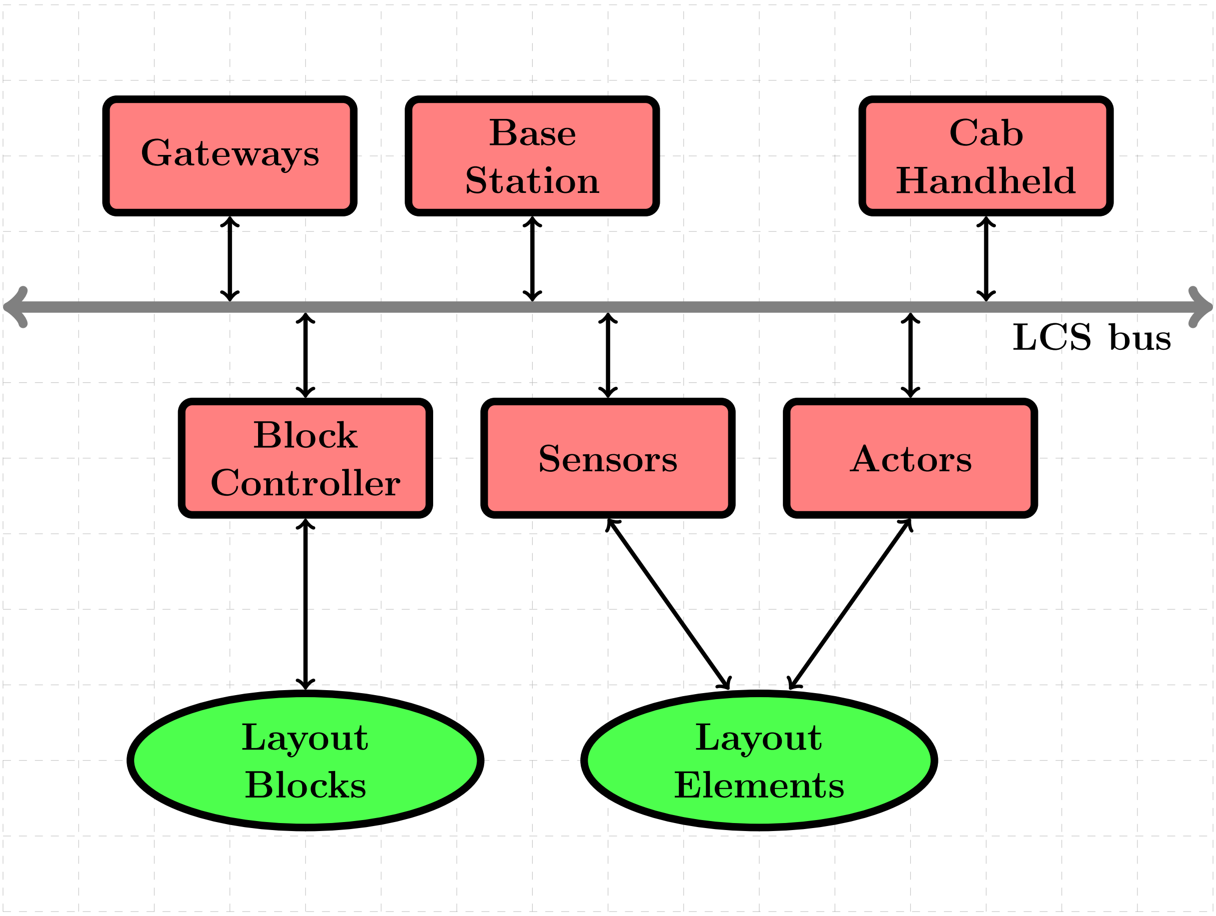


B.4 Split rectangle

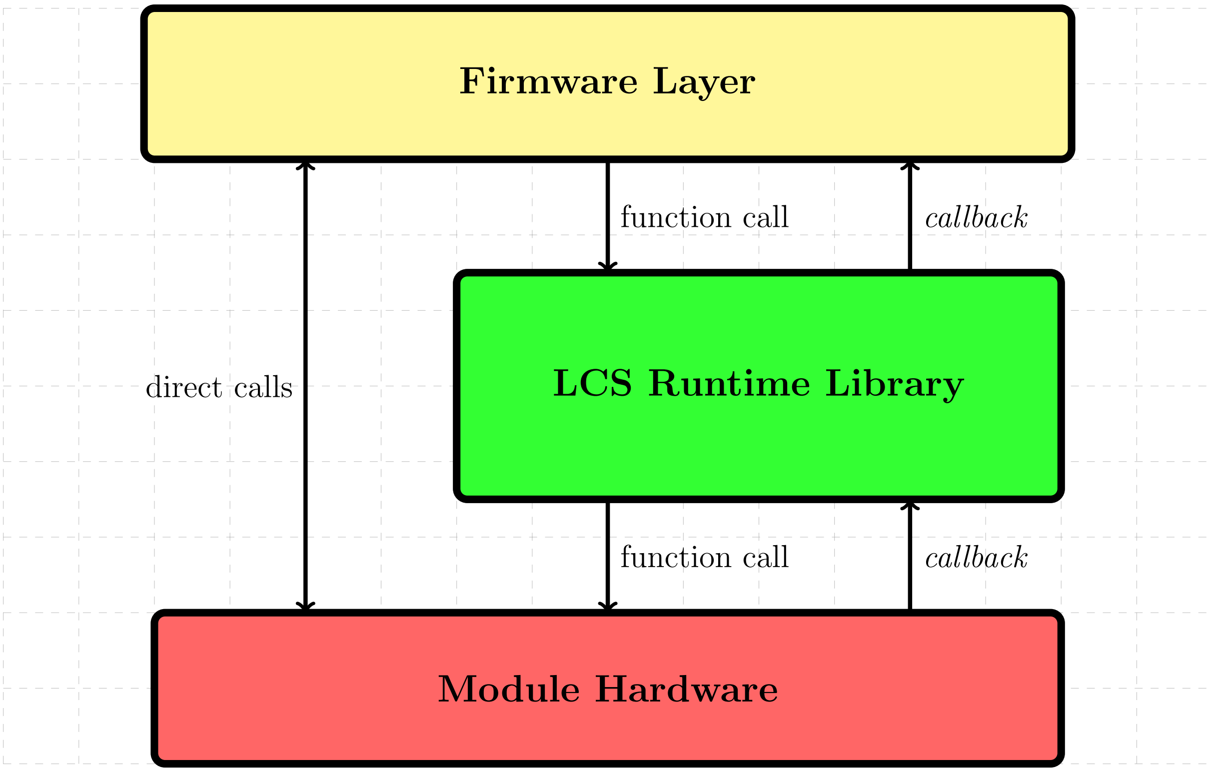
We would need the split rectangle for the runtime area maps....

B.5 Using tikzstyle

Another test with tikzstyle. It still is a lot of work to even make simple pictures look nice :-)


Gateways
Base Station
Cab Handheld
LCS bus
Block Controller
Sensors
Actors
Layout Blocks
Layout Elements

B.6 New picture



B.7 New picture - DCC Subsystem