A Physical Platform for Teaching Distributed Systems in a Simplified Setting

TEO KLESTRUP RÖIJEZON & THERESE KENNERBERG
A Physical Platform For Teaching Distributed Systems In A Simplified Setting

Teo Klestrup Röijezon, Therese Kennerberg

April 2, 2019

Abstract

The digitalization of our society is moving quickly, and we all want to keep up with it. This puts a lot of pressure on schools to teach programming and computational thinking in the education. For this reason The Swedish agency of education (Skolverket) is adding computational thinking to the Swedish primary school curriculum.

As a response to Skolverket’s new curriculum, we developed a platform to teach the basics of distributed and bare-metal programming to students.

With inspiration from Zachtronics Industries’ games Shenzhen I/O and TIS-100, the platform consists of a motherboard, programmable nodes, and peripheral nodes. The nodes are programmed using an easy-to-use web-based UI, intended to be served from a server running in the classroom.

Sammanfattning


Som ett svar på Skolverkets nya läroplan har vi utvecklat en plattform för att lära ut grunderna om distribuerade och maskinmära programmering.

Plattformen, som är inspirerad av Zachtronics Industries spel Shenzhen I/O och TIS-100, innehåller ett moderkort, programmerbara noder, och periferinoder som utför en förbestämt funktion (och interagerar med omvärlden). Noderna programmeras genom ett lättanvändt webbgränssnitt.
Contents

1 Introduction 4
  1.1 Research Questions 4
  1.2 Delimitations 5

2 Background 5
  2.1 Computational Thinking 5
     2.1.1 In Swedish Schools 6
     2.1.2 Prior Art 6
  2.2 Purpose 7
  2.3 Goal 7
  2.4 Ethics And Sustainability 7

3 Method 8
  3.1 Targeted User And School Selection 8
  3.2 Evaluation 8
  3.3 Constraints 9
     3.3.1 Time 9
     3.3.2 Cost 9
  3.4 Tools 9
     3.4.1 Micro-Controllers 9
     3.4.2 Programming Languages 10
     3.4.3 PCB Design 10

4 Result 11
  4.1 Communication 12
     4.1.1 PicoTalk 12
  4.2 Node Design 13
  4.3 Nodes 14
  4.4 Motherboard 14
  4.5 PicoASM 14
  4.6 PicoIDE 16
     4.6.1 Security 16

5 Analysis 16
  5.1 Evaluation Of The Final Product 18
  5.2 Alignment With Skolverket’s Course Goals 18
     5.2.1 Future Opportunities 18
  5.3 Development Issues 18
     5.3.1 PicoTalk Timing 19
     5.3.2 Conflicting Special-Use Pins 19
     5.3.3 Power Usage 19
     5.3.4 Keyed Connectors 20
  5.4 Node Selection 20
6 Further Opportunities
6.1 More Nodes ........................................ 20
6.2 More Exercises ..................................... 20
6.3 User Management ................................. 20
6.4 Studies ............................................. 21
6.5 Compliance ......................................... 21

A PicoNode-Compute manual
A.1 Introduction ....................................... 23
A.2 Features ........................................... 23
A.3 Getting started .................................... 23
A.4 Hardware and layout .............................. 23
A.5 Instructions ....................................... 24
  A.5.1 Values ......................................... 24
  A.5.2 Basic instructions ............................. 24
  A.5.3 Arithmetic instructions ....................... 24
  A.5.4 Test instructions ............................... 24

B PicoNode-Switch manual
B.1 Introduction ....................................... 26
B.2 Features ........................................... 26
B.3 Getting started .................................... 26
B.4 Hardware and layout .............................. 26
B.5 Switches ........................................... 26

C PicoNode-LED manual
C.1 Introduction ....................................... 27
C.2 Features ........................................... 27
C.3 Getting started .................................... 27
C.4 Hardware and layout .............................. 27
C.5 LEDs ............................................... 27

D PicoStorm ........................................... 28
D.1 Messages .......................................... 28
  D.1.1 Endianness .................................... 28
  D.1.2 Length ........................................ 28
  D.1.3 Type ........................................... 28
  D.1.4 Payload ........................................ 28
  D.1.5 Empty messages ............................... 28
D.2 Commands .......................................... 28
  D.2.1 Download Bytecode (Type = 1) ............... 28
  D.2.2 Reset (Type = 2) ............................. 28
D.3 Events ............................................. 29
  D.3.1 Downloaded Bytecode (Type = 1) .......... 29

Terminology and Acronyms

BOM  Bill of Materials
CAD  Computer-Assisted Design
1 Introduction

The goal of this project is to provide a hands-on physical pedagogical education platform for schools. The platform will be tailored towards Skolverket’s new curriculum and learning goals for the subjects Computational Thinking and Programming[1], which is effective from July 2018[2].

The students will assemble a network out of pre-assembled pieces ("Nodes"), which either perform a fixed function (such as a sensor or LED, we call these "Peripheral Nodes") or are programmable by the student ("Compute Nodes") in our custom assembler language (PicoASM). The students program the nodes using our web-based editor environment.

The goal of the platform is aimed to teach parallel programming and distributed systems in an otherwise very simplified environment, where the students don’t have to care about what clock skew or inductance are. We also wanted to force the students to split their projects and work with multiple Compute Nodes early, which is why each one has very limited capabilities.

1.1 Research Questions

The study has examined the following questions:
1. Implementation of a platform to teach the basics of bare metal programming to students.
2. Evaluation of the pedagogical usefulness of the platform and how to implement the idea to motivate students.

1.2 Delimitations

The project is limited to 2.5 months, so the product will not be ready for the market in this time. Rather, the project aims to build up a solid base for a future project that can be represented on the market. It will be a simple prototype, primarily so that the students can try it out. Some suggestions for further development opportunities are mentioned in Chapter 6.

The product is targeted towards students in eighth grade, and the goal is for it to be accessible to both students who have programmed before, and those who have not.

2 Background

The digitalization of our society is moving quickly, and we all want to keep up with it. This puts a lot of pressure on the schools to teach programming and computational thinking in the education. For this reason Skolverket is adding computational thinking to the Swedish primary school curriculum.

Several countries have implemented programming into their schools, and this is creating market demand for tools to use in the teaching process. Many products to help teach this subject already exist, but they are generally all focused on providing a simple environment for programming a single isolated computer. This is great, but the course goals also include an understanding of how networks of computers work and interact[3].

2.1 Computational Thinking

Computational thinking is stated in a project by The Swedish agency of innovation (Vinnova) and Linköping’s University[4] to be a process of problem-solving which includes:

- Formulating a problem in such a way that it becomes possible to use computers to solve it.
- Breaking down complex problems into smaller parts.
- Finding repeating patterns and reusing them.
- Creating algorithms to automate the solution of a problem.
- Logically organizing and analyzing data.
- Represent data through abstractions such as models and simulations.
- Identifying, analyzing, and implementing possible solutions to find the most effective one.
• To generalize and use this kind of problem-solving process on other kinds of problems.

According to Heintz and Mannila[4], teaching computation in schools should have three key stages:

1. Ages 5-6: Algorithms and instructions, to create and debug simple programs, with a focus on logical thinking.
2. Ages 7-11: Develop and debug greater problems with given goals and use programming concept with variables, sequences, selection and repetition.
3. Ages 12-14: Two or more programming languages, Boolean logic, binary numbers, connection between the program and hardware.

This project is aimed at students aged 14-15, and fits the last key stage quite well.

2.1.1 In Swedish Schools

Skolverket is in the process of adopting programming and computational thinking in schools and has set up a number of goals for the schools to implement. However, the specific requirements are still quite unclear at this stage of the project, leaving a lot of room for municipalities, schools and individual teachers to experiment in their implementation.

Many teachers also feel poorly prepared for the change. A study made in 2017 by Lärarnas Riksförbund[5] shows that 8 of 10 math teachers with students in the 8:th and 9:th grades feel insecure about teaching programming, and that 54% of the teachers don’t have any programming experience at all. According to another study at Macquarie University[6] teachers tend to be anxious to develop new learning resources, and especially when dealing with new and unfamiliar teaching material. This shows that the tools must also be easy to pick up by inexperienced teachers, requiring a heavy focus on good and clear documentation as well as a simple and encouraging user interface.

The User Interface (UI) should be kept as simple as possible, so that it is easy to use with few setup steps, letting them focus on the important part: the programming.

2.1.2 Prior Art

So far, "educational" programming has mostly taken one of two tracks: "toy" languages and environments for conventional computers that are designed to be visually exciting (such as Scratch[7]), or pre-assembled microcontroller kits (such as the BBC micro:bit[8]). There are also some devices, such as the Raspberry Pi[9], which sell what is essentially a very cheap conventional computer in a tiny form factor.

We’ve tried to go in a slightly different route, inspired by Zachtronics Industries’ games Shenzhen I/O[10] and TIS-100[11]. Instead of
giving them a single computer or Micro-controller Unit (MCU) for the whole project, we give them a whole bunch of extremely limited MCUs, in order to force them to think about splitting their projects a long time before a traditional approach would have. To compensate for this, we’ve also provided a simplified way to communicate between them, without having to worry about more complicated concerns, such as bits, timing, and clock skew.

Compared to the Zachtronics games, building this as a physical kit required us to consider practicality to a bigger degree. Shenzhen I/O’s free-form Printed Circuit Board (PCB) layout system gives the user an extra degree of freedom, but requiring the user to wait for (and pay for) the PCBs to be re-manufactured after each design change would have been very stifling. Alternately, you could have a pre-manufactured PCB but let the user set up connections as they pleased, using jumper wires. However, we also rejected this design due to space constraints and the messy design it would have produced.

2.2 Purpose
Improving the digital literacy of school students, in Sweden and abroad.

2.3 Goal
To develop a basic prototype of an educational platform to be used in the Swedish schools.

2.4 Ethics And Sustainability
It’s important to make sure that the product is produced in a sustainable manner. For example, care should be taken to avoid conflict minerals, child labor, and so on.

The system also shouldn’t contain any materials that are harmful to the user. All major components\(^1\) in the prototype are compliant with the Restriction of Hazardous Substances Directive (RoHS). The same care should also be taken when sourcing generic components for manufacturing. The whole system runs on a quite low voltage (3.3 V logic and some 5 V power delivery), so the risk of electrical hazards should be minimal.

It should also be secure enough that it can’t be snooped on, or abused by malware and botnets. Currently all traffic between the downloaders and the server is encrypted and authenticated, and reasonable care has been taken to ensure that the communication is handled safely. However, the security has not been professionally audited, and it has been designed under the assumption that it will be used in an isolated network.

The server also doesn’t currently authenticate users.

\(^1\)The MCUs (and modules) such as the STM32F030, the downloader module (the Adafruit Huzzah32), and the voltage regulator (TI LM317).
Finally, as an educational product we believe that it’s very important that the students have the freedom to tinker with it, and that they have access to all the design materials. Any educational product is ultimately about setting up "illusions" so that the user can focus on what’s important, but it’s just as vital that they are allowed to peek behind the curtain once they feel ready.

3 Method

Skolverket is about to implement new learning goals for computational thinking in July 2018. Their vision for 2020 is that every student has developed an adequate competence for digital technology [12].

The primary question for this thesis project is: "How can you implement computational thinking for an parallel and hardware-close programming?". To try to answer this we developed a platform to be used in schools when teaching programming. The evaluation of the platform was based on letting school students and teachers play around with a prototype, and then letting them fill out a questionnaire about their experience.

3.1 Targeted User And School Selection

The product is aimed at students in the 8:th grade. The platform aims to be a good base for students without programming practice, but also challenging enough for a more experienced programmer. The manual is in English, so the students must understand English. Initially we wanted to gather feedback from as diverse a set of schools as possible, but due to a lack of interest from schools we only found one class to evaluate the prototype.

3.2 Evaluation

We used a questionnaire to get a basic understanding of the efficacy from the students and teachers testing the prototype. Initially we wanted to use the feedback from the survey in an agile process to shape future development of both the prototype and the evaluation metrics, but since we failed to produce a testable Minimum Viable Product (MVP) for our milestones we ended up only performing one survey at the end of the project.

The data was gathered through a questionnaire with two sections. The first is about their programming experience in general, while the other is about their experience with the prototype. This is to evaluate what disposition the students have about programming, and whether we manage to change that disposition. We also customized the questionnaire, so that teachers got one that was more focused around the teaching experience, while the students’ were focused around their learning experience.

The data will be used to evaluate the product, and for suggestions towards future work. We will not present any statistics from the survey,
since our data is too limited and biased to produce any significant results.

An alternative way to evaluate the platform could be to do a metric-driven user study, but this would take far more time, and require involving multiple competitive platforms to establish a clear baseline to compare against.

3.3 Constraints

3.3.1 Time

We had 2.5 months to complete the project, our goal being to have a finished and testable prototype in the end.

3.3.2 Cost

We didn’t have a hard budget in mind, but our goal was always to keep the cost as low as possible, in order keep the price point attractive for schools. However, we ended up making some concessions due to the time constraints. We’ll note where this has happened.

3.4 Tools

3.4.1 Micro-Controllers

We knew from the start that the Compute Node required an MCUs with the following properties:

- Four General Peripheral Input Output (GPIO) pins, one for each direction when communicating between the Nodes
- A software-controlled timer, for keeping the communication between nodes consistent
- A bidirectional Universal Asynchronous Receiver/Transmitter (UART) channel, for communicating with the Downloader
- "Enough" RAM, Flash, and interrupts to hold both our PicoASM runtime and the UART-controlled flashing program
- An external reset
- Some kind of debugger, to avoid making our lives hell

We chose to use the STM32 MCUs platform for the Compute Nodes because we both had prior experience with it, and because the ARM[13] Cortex-M cores it is based on are quite well-supported by various compilers. For example, the Rust compiler (which we used) has mainline support for Cortex-M, but requires an experimental fork for AVR[14] support.

We then chose the F030 series in particular because it contained both the cheap F4P6 chip (while still satisfying our requirements), as well as the beefier but compatible C8T6, which we used for debugging our firmware.

Our Peripheral Nodes had the same requirements as the Compute Nodes, with the following exceptions:
• No UART, since they are not connected to the Downloader
• Generally less beefy hardware, since the peripheral programs tended
to be much simpler
• Whatever the Node’s functionality in particular required (for the
  LED and Button nodes, four extra GPIO pins)

To keep things simple we chose to use the same STM32F030 MCUs
for the Peripheral Nodes, since it still fulfilled all of their requirements
too. It also meant that we could reuse much of our existing firmware
and board layouts.

Given enough time you could probably port it to a cheaper plat-
form, such as AVR[14], but you might need to simplify and remove
features. However, at least the current Peripherals should be fairly
safe to port.

For our downloader unit we used the ESP32[15] MCU, because it
has a relatively large developer community, plenty of resources, and a
built-in Wi-Fi transceiver. Cost was also a far smaller problem here,
since we need much smaller quantities of these. ST also has a few Wi-
Fi-capable chips, but they are all marked Not Recommended for New
Designs.

3.4.2 Programming Languages

We used Rust[16] to program the Node MCUs. This was both because
of Rust’s improved safety features over C (such as pointers with limited
lifetimes, and move semantics), as well as the improved type system.

For example, the support for tagged unions (also known as algebraic
data types or sum types) allowed us to represent our PicoTalk state
machines without needing either a lot of repeated states (Wait1, Wait2,
Wait3), or manually reinterpreting variables depending on the current
state.

We used C for programming the downloader, because the ESP32
MCU is based on the rather uncommon Xtensa[17] architecture, which
Low-Level Virtual Machine (LLVM) (used by the Rust compiler) does
not support. This wasn’t as much of a hindrance, since the downloader
is mostly responsible for relaying messages between the server and the
connected node.

The website was developed using Scala[18] for both the front- and
back end. This was very useful for keeping the code bases consistent,
and helped avoid code repetition when defining view models, for ex-
ample. In theory it would also have been nice to stay consistent with
the MCUs, but because of the resource constraints we wouldn’t have
been able to share as much code anyway.

3.4.3 PCB Design

We started out prototyping using solder-less breadboards and jumper
cables, so that we would have more flexibility when changing plans.
However, since our MCUs are Surface-Mount Design (SMD)-based we
still had to manufacture our breakout boards. We also designed and
manufactured a "debug" board for the STM32F030C8T6, with the same pinout as the STM32F030F4P6 that we were using as our design target.

Once we were happy with our design we made a more formal design in our Computer-Assisted Design (CAD) program KiCad[19]. This consisted of Node boards based on the MCU reference design (see Chapter 4.3) with pin headers for easy attachment, a Motherboard (see Chapter 4.4) holding the Nodes and the downloader, and a mezzanine board exposing the debug pins.

4 Result

Our final prototype consists of four primary components, which are shown in Figure 1: the Nodes, the downloader, the server, and the Integrated Development Environment (IDE). The Nodes are composed (and, for the Compute Node, programmed) by the user to perform the desired function. The downloader acts as a bridge between the nodes and the server, and also provides system-wide tasks such as resetting all nodes. The IDE is used by the user to write programs, and to download them onto the Compute Nodes. The server stores users’ programs, and also acts as a gateway for the users to access the downloader, which can be used to enforce access control as well as validate that all programs are valid (although neither of these two tasks are implemented today).

All source code and design files are available at https://gitlab.com/PicoNodes/PicoNodes.

Figure 1: The PicoNodes communication architecture

Nodes are programmed using an in-browser IDE. Since browsers are heavily sandboxed we use a server for communicating with Nodes, as well as for storing the users’ code. We use a Downloader which is responsible for keeping the server connection open, as well as for storing authentication keys. The Server ↔ Downloader ↔ Compute
Node communication happens using the PicoStorm protocol, which is documented in detail in Appendix D.

Nodes communicate between each other using a protocol called PicoTalk, which is documented in Chapter 4.1.1.

4.1 Communication

We decided to use a mesh interconnect (see Figure 2), where each node is connected to its direct neighbors. This is relatively simple to route, has no chance of collisions, and requires no form of absolute addressing. The connections between the nodes use PicoTalk (see Chapter 4.1.1).

![Figure 2: The PicoNode 2D mesh layout](image)

4.1.1 PicoTalk

The Nodes communicate using an asynchronous single-wire protocol that we designed, called PicoTalk. We did this because we wanted to use a mesh interconnect instead of a shared bus, and because we didn’t have the hardware support or pin count to connect each pair using a more common protocol, such as UART. It is not a master/slave protocol, the transmitter/receiver roles are negotiated for each message.

Because it is asynchronous, each side must provide its own clock. The clocks must be calibrated to 10kHz. The clocks are also reset for each message, in order to avoid clock skew corruption.

The communication line is held high while idle by a 4.7KΩ pull-up resistor on the motherboard. Thus all participants should be configured in open drain mode, with their internal pull-ups disabled.

PicoTalk can be divided into four primary stages, as shown in Figure 3: Idle, Handshake, Preamble, and Message.

In the Idle stage neither party is trying to transmit anything. When a party wants to transmit they enter the Handshake stage, by transmitting two low states. They then wait for a single low state in response.
If no response is heard after three states then the transmitter restarts the Handshake stage. The preamble stage is 5 states of alternating high and low signals. This is used to verify that the clocks are synchronized correctly. Afterwards the message (a single byte) is sent, least significant bit first. The bus then returns to the Idle stage.

Figure 3: The stages of PicoTalk

4.2 Node Design

There are a few design elements that are common to all nodes, to ensure that they can all be placed in any position on the motherboard, and so that they can all be debugged using the same tools.

Using the LED node (Figure 4) as an example, they all share the following features:

Figure 4: The front side of the LED Node PCB

- They are all 30.48x27.94mm
- The angled corner marks the correct orientation
- The grips (in the top and bottom) help you to remove the nodes from a packed motherboard
- The pins all follow this layout (from left to right):
  - Top row:
    * 3.3V power (VDD)
    * SWCLK (used for debugging only, normally NC)
    * Ground
    * SWDIO (used for debugging only, normally NC)
    * Reset (active low)
  - Bottom row:
    * Down
They are all powered by STM32F030F4P6 MCUs

4.3 Nodes

We designed and fabricated three different Nodes: a Light-Emitting Diode (LED) display Node (see Figure 4), a button input Node (see Figure 5), and a programmable compute Node (see Figure 6) to connect them. They are documented further in their respective appendices (see Appendix C, B, and A, respectively).

Figure 5: The front side of the button Node PCB

Figure 6: The front side of the compute Node PCB (the J3 pins are supposed to be female, but limited by the CAD software)

4.4 Motherboard

We designed a motherboard based on the communication layout mentioned earlier, visualized in Figure 7. It also provides power, ground, and a reset control to all the nodes.

4.5 PicoASM

We designed a simple assembler dialect, based on Shenzhen I/O’s assembler language[10]. It is documented in further detail in Appendix
Figure 7: The front side of the motherboard PCB (female connectors are not shown due to CAD software limitations)

Listing 1: An example PicoASM program evaluating the C expression \( \text{down} = \text{left} + \text{right} \times \text{up} \). Note: The \text{mul} instruction is not currently implemented, but should be interpreted as \( \text{acc} = \text{acc} \times R/I \).

```
1  mov right acc
2  mul up
3  add left
4  mov acc down
```

A. This was chosen because it’s very simple (there are few instructions to learn), and that it provides a transparent view of what the processor will do, and when. For example, the C expression \( \text{down} = \text{left} + (\text{right} \times \text{up}) \) shows what the desired output is, but the C compiler is allowed to evaluate the components in any order. The equivalent PicoASM code (Listing 1) forces the developer explicitly decide an evaluation order.

PicoASM also provided a natural way for restricting code complexity. A C compiler is responsible for allocating registers and instruction blocks based on the code, which would make it confusing to abort compilation when those resources run out. On the other hand, PicoASM exposes those constraints directly to the user, where one line in the code always corresponds to exactly one operation.

It would also be possible to base the constraints on higher-level metrics, such as the number of C statements. However, these metrics tend to be easy to game and work around. For example, \( x1 = a + b \);
Listing 2: An example PicoASM program: a "wheel of fortune" that spins while an input greater than zero is given

x2 = x1 * c; contains two statements, but is equivalent to x2 = (a + b) * c; which only contains one statement.

Contrary to typical x86 desktop processors it does not have any jump or branch instructions. Instead any instruction can be associated with a conditional flag, which is set by our test instructions. An example PicoASM program is given in Listing 2, which acts as a simple "wheel of fortune", given a Button Node on the left side, and a LED Node on the right side.

4.6 PicoIDE

We provided a web-based editor called PicoIDE, which allows you to edit programs and upload them to Compute nodes. It be seen in Figure 8. It is intended that this runs on a server on the same network as the Downloaders.

4.6.1 Security

Downloaders communication is encrypted using Transport-Level Security (TLS), with verification of client and server certificates. Browser communication is not encrypted by default, but it is recommended that this is done using a reverse proxy, such as Nginx. There is currently no user authentication system.

5 Analysis

During the project we have developed an educational platform, with the aim that the platform can be used by teachers to late primary school to teach programming.

We also wanted to get an evaluation of the prototype, from surveys and observation in different schools. A prototype was developed, with some time left for an evaluation. However, the teachers’ lack of interest and time constraints were limiting factors. In the end, one teacher offered to try out PicoNodes. The evaluation went better than we had anticipated; most of the students said they were motivated by PicoNodes, and wanted to learn more programming afterwards.
Figure 8: A screenshot of PicoIDE, with a few saved files and showing the program in Listing 2
5.1 Evaluation Of The Final Product

To motivate students, the prototype is designed to be very simple, with little end user setup required. Our goal is for it to be easy to get started, but with the opportunity to develop challenging tasks for more advanced students to practice with.

We had planned to let 2-4 students test it in pairs, but due to time constraints we only had 1 kit ready, while 8 students showed up. This meant that we had to improvise, and they ended up passing the kit around the table, while the others were either observing or preparing their own programs on their tablets.

5.2 Alignment With Skolverket’s Course Goals

We believe that PicoNodes can help teachers meet the following goals in the Technology (Teknik) subject (for grades 7-9)[3].

**Electronics and programming** The pupils program the nodes in an environment that is likely unfamiliar to them.

**How components cooperate when forming a larger system** Pupils are forced to consider how their components should interact with each other.

**Discussion of technical solutions** Many tasks have many possible solutions, with various pros and cons, that pupils will (hopefully) want to compare and contrast.

**The pros and cons of distributed systems** Pupils that have already used other other systems (such as Scratch) will likely notice that the distributed system is usually harder to reason about and debug, which is a good starting point for discussion about when/how these systems are used in practice.

5.2.1 Future Opportunities

There are also a number of learning goals that PicoNodes does not directly explore, but where it may be a useful tool for teachers and/or advanced pupils.

**Information exchange and networks** PicoTalk uses an unrouted mesh topology, but would other topologies have other advantages? For example, a Ring topology could be implemented on top.

**Electronics and programming** The development of a new Node could be an interesting starting point for discussion, or even a project for advanced pupils.

**Control systems, and how mechanical and digital technology cooperate** Further nodes could be developed that interact with physical control systems, both inputs (sensors) and outputs (motors, valves, etc).

5.3 Development Issues

Some miscalculations where made during the development process.
5.3.1 PicoTalk Timing

For example, our original PicoTalk implementation had severe clock skew issues, where the transmitter and receiver would start synchronized, but slowly go out of sync during the message. To solve this we ended up requiring the receiver to re-synchronize their internal clock for each message (see also Chapter 4.1.1). This also limited the speeds that the nodes could communicate at. Another way to solve this could have been a shared clock bus, equalizing clock skew between the nodes. However, that would have led to a higher Bill of Materials (BOM) cost, and a more complicated PCB design. Also, we liked having an uneven number of pins, since that also acted as a guide to orient the Node correctly in its socket.

5.3.2 Conflicting Special-Use Pins

Another issue was that the downloader refused to boot when connected to more than 2 Nodes. At first we thought this was a power issue (see Chapter 5.3.3), but this didn’t solve the issue. Instead, it turns out that the ESP32 pin we had used to control node reset was used to detect the logic voltage level, expecting a low input for $3.3\,\text{V}$ logic. Since this is the default it has a pull-down resistor. At the same time, the node reset is active-low, and so has a built-in pull-up resistor so that device is usable when the pin is floating. When more than 2 Nodes were connected the pull-ups would overpower the ESP32’s pull-down. To fix this we attached a stronger pull-down to the pin, which the ESP would easily overpower once running, and activate the Nodes.

5.3.3 Power Usage

To dimension our voltage regulators, and to make sure that we would be able to power our system using USB, we had to model our (worst-case) power usage.

In these formulas, $W_{\text{node}}$ is the number of Node columns (3), $H_{\text{node}}$ is the number of Node rows (3), $I_{\text{node}}$ is the maximum current draw of each Node, $I_{\text{downloader}}$ is the maximum current draw of the Downloader, and $R_{\text{pullup}}$ is the resistance of the PicoTalk pull-up resistors (4.7 kΩ). Based on their respective documentation for the STM32F030F4[20] (the Nodes) and the Huzzah32[21] (the ESP32 module that we use for the Downloader) we assume that $I_{\text{node}}$ is 120 mA and $I_{\text{downloader}}$ is 250 mA.

\begin{align}
N_{\text{pullups}} &= W_{\text{node}} \times H_{\text{node}} \times 2 + W_{\text{node}} + H_{\text{node}} \\
I_{\text{pullups}} &= \frac{U}{R_{\text{pullup}}} \times N_{\text{pullups}} \\
I_{\text{nodes}} &= W_{\text{node}} \times H_{\text{node}} \times I_{\text{node}} \\
I_{\text{board}} &= I_{\text{pullups}} + I_{\text{nodes}}
\end{align}

19
\[ I_{\text{total}} = I_{\text{board}} + I_{\text{downloader}} \]  

Plugging in our numbers, this gives us a total power usage of 1.4 A (Equation 5), and a voltage regulator capable of feeding (at least) 1.1 A (Equation 4).

This is outside of the baseline USB specifications (500 mA and 900 mA for USB 1/2.x and 3.x, respectively), but inside the requirements of the more modern USB Power Delivery, Battery Charging, and USB-C specifications (3 A, 1.5 A, and 1.5 A, respectively).

5.3.4 Keyed Connectors

Originally we wanted to use keyed connectors to prevent users from misaligning the Node connectors, but we ended up scrapping that due to cost issues. In the end we ended up using an odd number of pins, so that mistakes would at least be visible due to the pins sticking out into the air.

5.4 Node Selection

Another problem we had was defining what Nodes to make. We obviously needed a programmable Node, but we also wanted some way to let the user affect the computation, and to let them see the result.

At first we wanted to make a digit display node, similar to a Lixie display[22]. However, due to both space and time constraints we ended up making a simplified 4-LED Node instead.

6 Further Opportunities

6.1 More Nodes

More Nodes could be developed for special visual effects, or for specific purposes such as games or exercises. For example, you could develop nodes to communicate with physical control systems (sensors, motors, etc), or to enable more advanced programs (such as different memory systems).

6.2 More Exercises

Another area of interest would be to create more exercises. It could also be interesting to have LEGO-style tutorials[23] for building specific projects.

6.3 User Management

The current approach of letting everyone access everything might get chaotic in a classroom environment. Instead, files and downloaders should be linked to specific users.
6.4 Studies
It would be useful to have a more in-depth study of the platform’s educational efficacy.

6.5 Compliance
Before the product is taken to market it must be tested with the regards to the relevant compliance standards, such as CE, RoHS, UL, and FCC. There should also be a formal security audit.

References


A  PicoNode-Compute manual

A.1  Introduction

The PicoNode-Compute allows you to add custom behavior to your PicoNode setup. It is programmable by the user.

A.2  Features

The PicoNode-Compute is able to store up to 20 instructions, and can communicate freely in all four directions. It also features a single memory slot, which can store a value.

The program will restart after the last instruction is executed.

A.3  Getting started

Follow these steps to try out the PicoNode-Compute with a simple sample program:
1. Power the motherboard
2. Place a PicoNode-Switch on a available slot in the leftmost column of the motherboard.
3. Place the PicoNode-Compute to the right of the Switch node
4. Place a PicoNode-LED to the right of the Compute node
5. Connect the TX pin on the downloader to the RX pin on the Compute node, and the RX pin on the downloader to the TX pin on the node.
6. Open the PicoIDE website
7. Select the downloader named "Main"
8. Enter the code from Listing 3
9. Click "Download"
10. Press a button
11. A LED should toggle every time you push the button

```
tcp left 0
+ tcp acc 1
+ mov 0 acc
+ mov 2 acc
mov acc right
```

Listing 3: Example that toggles a LED when any button is pressed

A.4  Hardware and layout

The hardware and the PCB layout are available in the data sheet provided for the PicoNode kit.
A.5 Instructions

All supported instructions are below:

A.5.1 Values

The instructions below have slots labeled <R> or <R/I>. This is where you plug in the values. A slot labeled I accept any value between -100 and 100 (this is usually called an "Immediate"), while slots labeled R ("Register") accept either a direction (up, down, left, or right) or the local memory slot acc.

Writing to a direction will transmit the value in that direction, while reading from it will receive a value.

Reading from acc will receive the last value written to it.

For the rest of this chapter, "value" will refer to either an immediate, or the value you receive when reading from a register.

A.5.2 Basic instructions

1. mov <R/I> <R>
   mov (move) copies the first value to the second register.

A.5.3 Arithmetic instructions

1. add <R/I>
   add reads the value and adds it to acc. In other words, acc = acc + value.
2. sub <R/I>
   sub reads the value and subtracts it to acc. In other words, acc = acc - value.

A.5.4 Test instructions

You can put the flags + or - before an instruction, in which case it will only execute if that flag has been set by a test instruction.

1. teq <R/I> <R/I>
   teq (test if equal) sets the + flag if the two values match. Otherwise it sets the - flag.
2. tlt <R/I> <R/I>
   tlt (test if less than) sets the + flag if the second value is smaller than the first value. Otherwise it sets the - flag.
3. tgt <R/I> <R/I>
   tgt (test if greater than) sets the + flag if the second value is bigger than the first value. Otherwise it sets the - flag.
4. tcp <R/I> <R/I>
   tcp (compare) sets the + flag if the second value is greater than the first value, or the - flag if the second value is smaller.
   No flag is set if the values are equal.
5. Comparison table
   Here’s a handy cheat sheet for the test instructions:
<table>
<thead>
<tr>
<th>instruction</th>
<th>case</th>
<th>+</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>teq</td>
<td>first=second</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>teq</td>
<td>first&lt;second</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>teq</td>
<td>first&gt;second</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>tlt</td>
<td>first=second</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>tlt</td>
<td>first&lt;second</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>tlt</td>
<td>first&gt;second</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>tgt</td>
<td>first=second</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>tgt</td>
<td>first&lt;second</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>tgt</td>
<td>first&gt;second</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>tcp</td>
<td>first=second</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>tcp</td>
<td>first&lt;second</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>tcp</td>
<td>first&gt;second</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>
B PicoNode-Switch manual

B.1 Introduction

The PicoNode-switch is designed to be used with the PicoNode kit and is compatible with the PicoNode kit’s motherboard. It is meant to give a more hands-on and visual understanding of the communication between the PicoNodes. This node is not programmable by the user but is already running a program designed to be used with the motherboard.

B.2 Features

The PicoNode-switch offers four switches, and can detect one press at a time.

B.3 Getting started

Follow these steps to learn how to use the PicoNode-switch:
1. Power the motherboard
2. Place the PicoNode-switch on a available slot on the motherboard. This slot should also have one available slot to the right.
3. Place the PicoNode-LED on the available slot to the right OR program a PicoNode to receive a value from the left.
4. If you placed a LED then press one button and a LED should light ELSE if you placed a programmed PicoNode, press one of the buttons and a value from 1-4 is transmitted to the PicoNode from the PicoNode-switch.

B.4 Hardware and layout

The hardware and the PCB-layout are available in the data sheet provided for the PicoNode kit.

B.5 Switches

The switches are used to symbolize a value transmitted from the node. There is one output pin to the right for transmitting a value from 1 to 4. The top left button holds the value of one, the bottom left button hold the value of two, the top right button holds the value of three and the bottom right button hold the value of four. When the button is pressed, its respective value is transmitted to node placed to its right. Only one value can be transmitted at a time and if no button is pressed the value transmitted is 0.
C  PicoNode-LED manual

C.1 Introduction
The PicoNode-LED is designed to be used with the PicoNode kit and is compatible with the PicoNode kit’s motherboard. It is meant to give a more hands-on and visual understanding of the communication between the PicoNodes. This node is not programmable by the user but is already running a program designed to be used with the motherboard.

C.2 Features
The PicoNode-LED offers four red LEDs, where one can be enabled at a time.

C.3 Getting started
Follow these steps to learn how to use the PicoNode-LED:
1. Power the motherboard
2. Place the PicoNode-LED on a available slot on the motherboard. This slot should also have one available slot in any direction next to the PicoNode-LED.
3. Place the PicoNode-Switch on the available slot next to the PicoNode-LED
4. If you placed a switch then press a button, and a LED will light up symbolizing the value received.

C.4 Hardware and layout
The hardware and the PCB-layout are available in the data sheet provided for the PicoNode kit.

C.5 LEDs
The LEDs are used to symbolize a value received from one of the input pins. There are four input pins for receiving a value, one in each direction. The PicoNode-LED will only receive a value in the range of 1-4. If it receives a value out of range, then no LED will light up. (Should it be able to receive more than one direction?)
D PicoStorm

D.1 Messages

All PicoStorm messages consist of three fields, the length, the type, and the payload.

Messages moving towards the Node are called Commands, while messages moving towards the server are called Events.

D.1.1 Endianness

All PicoStorm messages are big-endian.

D.1.2 Length

The length is a 32-bit (4-byte) unsigned integer, defining the total length of the type and payload fields (but not the length field itself).

1. Limitations

   The server will reject any message with a length field greater than 1024, while the Nodes will reject any message with a length greater than 60. This is subject to change.

D.1.3 Type

The type is a 32-bit (4-byte) unsigned integer, identifying which kind of message it is. The type namespaces are separate for Commands and Events.

D.1.4 Payload

The rest of the message is called the payload, and depends on the message in question.

D.1.5 Empty messages

Empty messages (consisting of a length field, but no type or payload) are allowed, and MUST be ignored by the recipient.

D.2 Commands

D.2.1 Download Bytecode (Type = 1)

Instructs the node to download the bytecode stored in the payload. The node should respond with Downloaded Bytecode once done.

D.2.2 Reset (Type = 2)

Instructs the Downloader to reset all nodes.
D.3 Events

D.3.1 Downloaded Bytecode (Type = 1)

The node is done downloading the bytecode. The payload contains a checksum of the saved bytecode, and the server should respond with a new Download Bytecode message if it does not match.

1. Checksum
   The checksum is the CRC32 of the downloaded bytecode. We’re using the polynomial 0x4C11DB7.