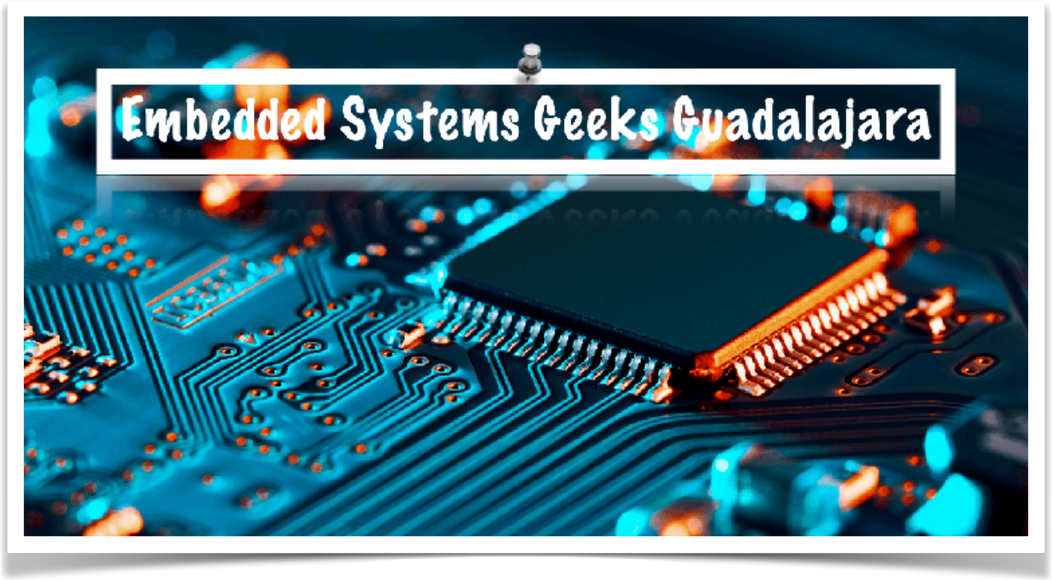
Embedded Systems Geeks GDL
Kick-off Meetup

Dr Frank Zeyda - 06 August 2023 @ HackerGarage GDL



Frank Zeyda

Dipl.-Inform. BSc PhD PgCLTHE

Independent Consultant
Safety-Critical Systems

https://www.linkedin.com/in/frank-zeyda/

frank.zeyda@gmail.com

Research Interests:

- Semantic Foundations / Unifying Theories (UTP)
- Formal Methods and Verification
- Rigorous Digital Engineering
- Automated Theorem Proving (Isabelle/HOL)

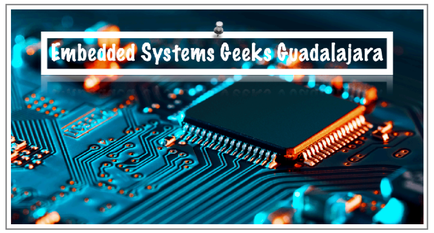
Academic Involvement (2001 to 2018):

- PhD from Teesside University (UK) in 2007.
- Research Associate/Fellow at the University of York (UK).
- Senior Lecturer at Teesside University and the University of York (UK).
- Lead RA on several EPSRC-funded and EC-funded research projects.
- Most publications are listed on: **https://dblp.org/pid/23/6727.html**

Industrial Involvement (1999 to 2000 & 2018 to 2022):

- Siemens Mobility / Transportation Systems (Germany)
- Verified Systems International GmbH (Germany)
- Galois, Inc. (U.S.) (consultant on industrial R&D projects)

My story with “embedded” …

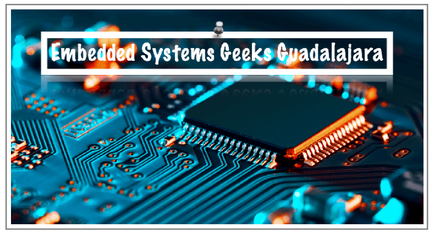


- Electronics was probably my first love of engineering and science (burning myself umpteen times when learning to solder at age 5 …).
- Grew up with home computers such as the **TI-99/4A**, **C64**, **Amstrad CPC** and Commodore **Amiga** (and a few more exotic ones).
- Built an 8-bit computer (**Z80**) from scratch in my first year at university

  ➡ Building mini-computers was a popular pass-time in the 80s …

  ➡ **Arduino**, **Raspberry Pie**, and **co** did not even exist back than …
- Never had a chance to study hardware and embedded systems **properly** due to delving “too quickly” into academic research and my PhD work.
- Taught myself various HW topics in the last five years, from programming MCU to building SoC on FGPA, and deploying **Linux** systems on embedded devices.

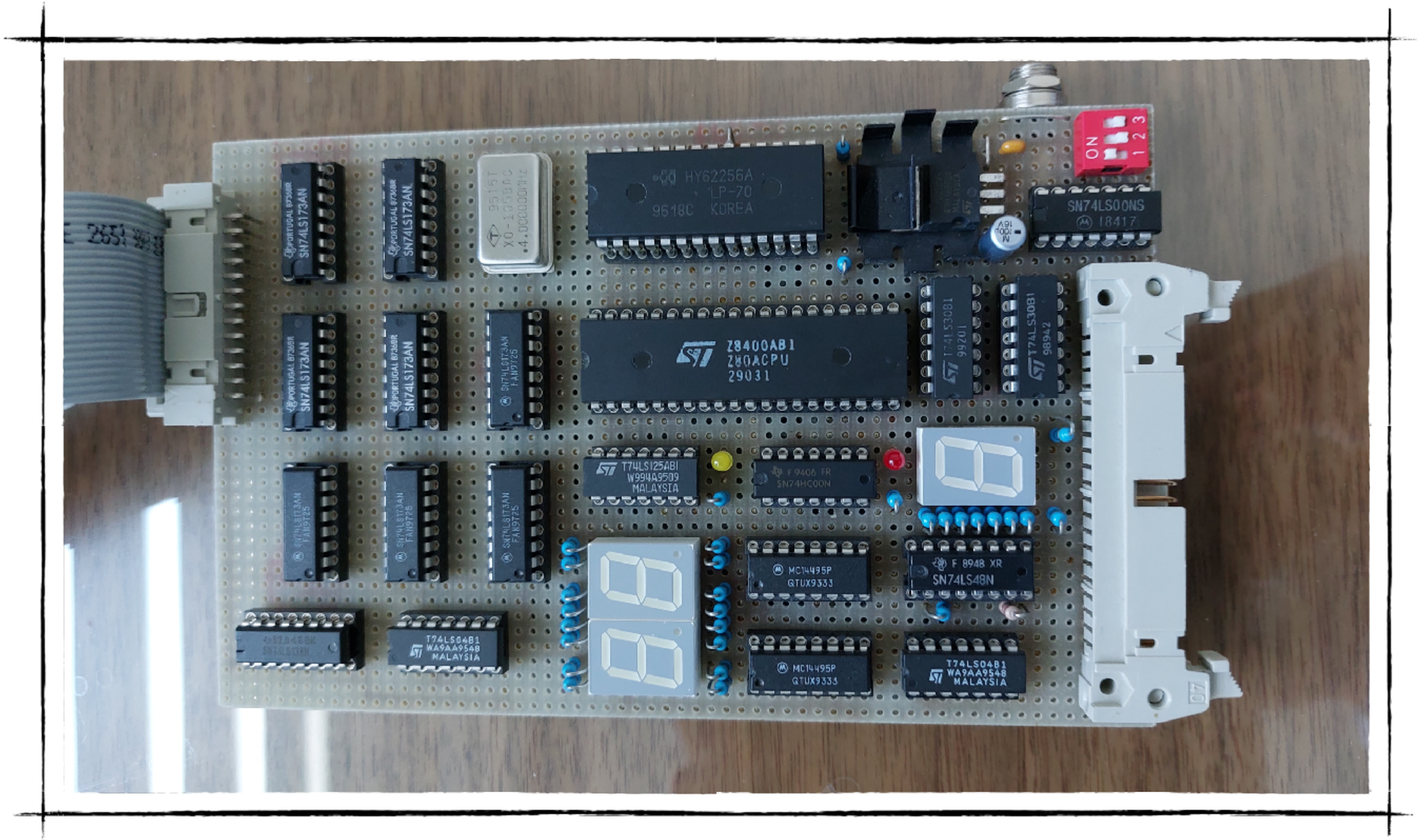
My story with “embedded” …



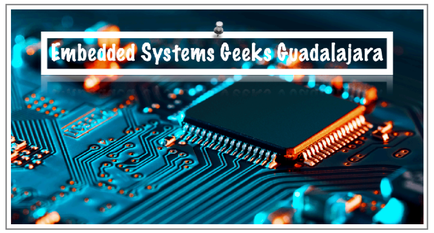
- Electronics was probably my first love of engineering and science (burning myself umpteen times when learning to solder at age 5 …).
- Grew up with home computers such as the **TI-99/4A**, **C64**, **Amstrad CPC** and Comm
- Built an 8-b rsity
  - ➡ Building .
  - ➡ **Arduino** …
- Never had roperly due to delving “too quickly” into academic research and my PhD work.
- Taught myself various HW topics in the last five years, from programming MCU to building SoC on FGPA, and deploying **Linux** systems on embedded devices.

I am now offering consultancy on rigorous (model-based) engineering of **trustworthy** and **secure** embedded systems, considering the full HW/SW life-cycle and V&V stack.

Mini-computer built from parts

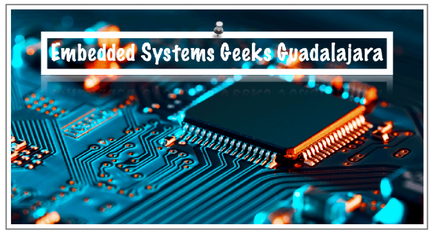


Overview

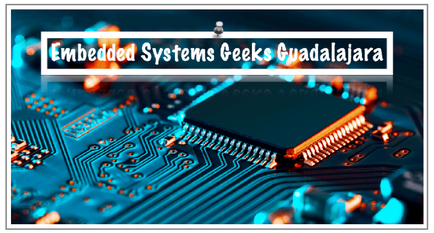


- **Motivation** and **rationale** for the group and community
- Introduction to Embedded Systems
- Topical area: microcontrollers (MCU)
- Topical area: FPGA / ASIC
- Topical area: vintage and retro-computing
- Miscellaneous topics
- **Conclusion** and **future plans**

REVIEWED UNTIL HERE

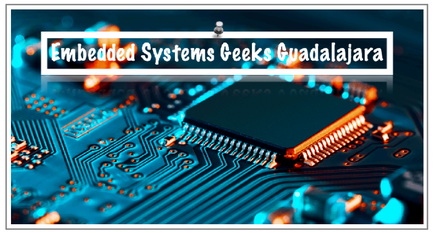


Why another Meetup Group?



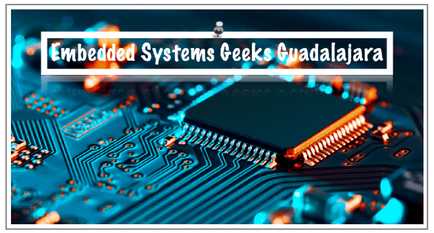
- Guadalajara has several well-established Meetup groups related to **software development**, **programming**, and **technology** (JUG, Pythonauts, DataLab, j4Guanatos, Golang / Rust Lang / Kotlin, etc.)
- I believe there is a "gap in the market" to cater for **embedded systems** and **hardware** engineering, as well as **IoT** communities in GDL.
- Embedded System engineering is in many ways different and (I believe) **more challenging** than other domains of software engineering.
- It has become so much more **accessible** due to low-cost boards for **microcontrollers**, **FPGA** and **SoC** being available of-the-shelf.
- It is a lot of fun to build something that is **physical** and **tangible**, and **interacts** with the world around us.

What are Embedded System?



- Here is my **personal take** on it (you might have more to add):
  - run on their own specialized hardware / platform / OS …
  - directly interact with **electrical** components and devices …
  - operate within a cyber-physical environment …
  - use **sensors** to observe the state of the world …
  - use **actuators** to change the state of the world …
  - may interact with humans …
  - may interact with online / www and cloud resources …
  - may be autonomous and self-learning …
  - are often **real-time** and **safety-critical** and subject to **security attacks**.

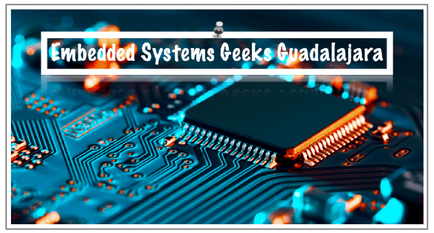
What are Embedded System?



- Here is my **personal take** on it (you might have more to add):
  - run on their own specialized hardware / platform / OS …
  - directly interact with **electrical** components and devices …
  - operate within
  - use **sensors** to
  - use **actuators**
  - may interact with humans …
  - may interact with online / www and cloud resources …
  - may be autonomous and self-learning …
  - are often **real-time** and **safety-critical** and subject to **security attacks**.

This makes the **design** of such systems extremely challenging.

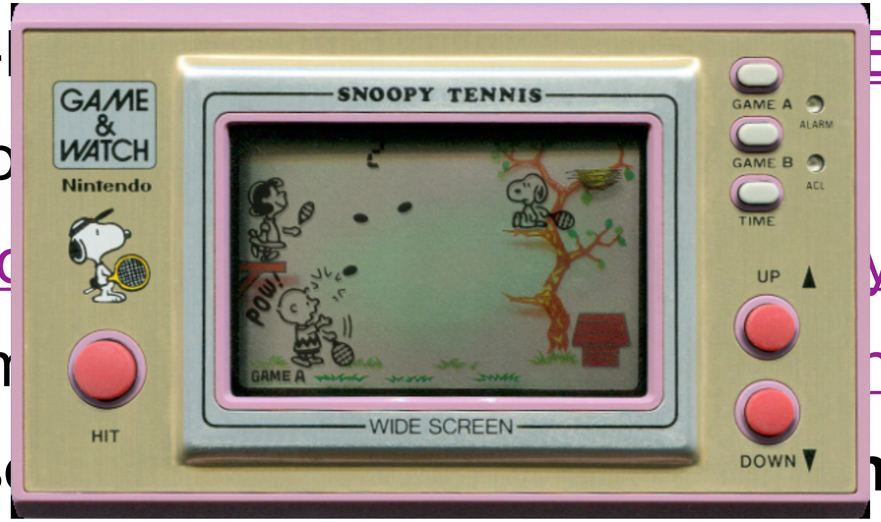
Examples of Embedded Systems



- Self-balancing control system in a Segway.
- Medial devices such as pace-makers or radiotherapy accelerator.
- Fly-by-wire control system of the Eurofighter Typhoon aircraft.
- Google Maps navigation software (accesses remote cloud)
- Guidance, Navigation & Control Systems of the Mars Orbiter
- Compare this to the Apollo Guidance Computer of the moon landing
- Ensemble of **interconnected components** in a modern car
- Chip in a credit card to **sign** / **authorize** payments
- From a Tiger LCD games and Tamagotchi to XBOX and PS5.

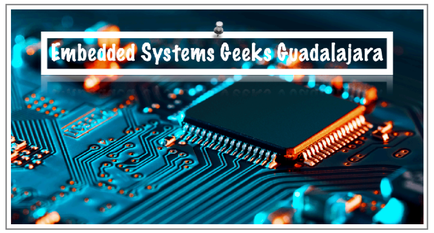
Examples of Embedded Systems

- Self-balancing control system in a Segway.
- Medial devices such as pace-makers or
- Fly-
- Goo
- Guid
- Con
- Ens
- Chip in a credit card to **sign** / **authorize** payments
- From a Tiger LCD games and Tamagotchi to XBOX and PS5.



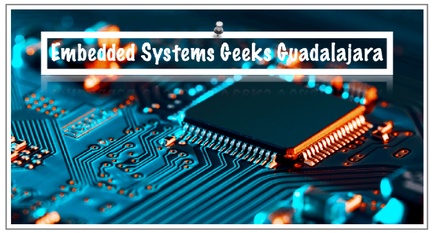
Sometimes systems can be fooled too …

Engineering Challenges



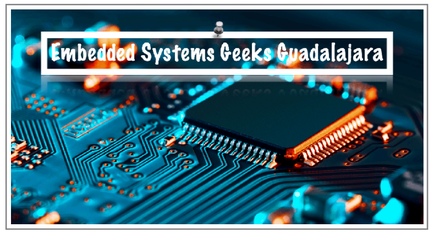
- We are constrained by the **hardware** (microcontroller, signal protocols, clock frequency, memory, FPGA logic resources, input / output pins, etc.)
- We are constrained what **languages** we can use to program the device:
  - C/C++ are still the *de facto* standard I believe …
  - **assembly** and bare **machine language** are not uncommon …
  - code may be generated from high-level models (Simulink, AADL, etc.)
  - CircuitPython, Rust and other modern languages on the rise …
- We are constrained by the availability of software libraries, APIs, and tool-chains for **cross-compilation** (not always superb and complete …)
- We are constrained by the **devices** that we interact with (digital vs analog, frequency of signal carrier, voltage/current specifications, and so on …)

“Technology Confusion”



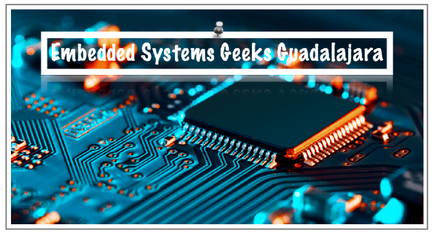
- Digital industries **churn out** new technologies, programming languages, tools and frameworks on a daily basis …

“Technology Confusion”



- Digital industries **churn out** new technologies, programming languages, tools and frameworks on a daily basis …
- But this does not relieve us form **engineering** (which is what strangely many developers seem to believe …)

“Technology Confusion”

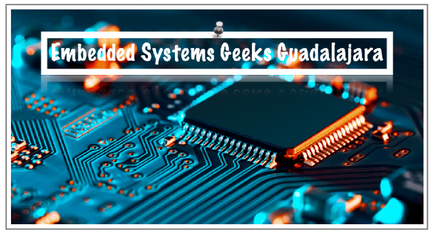


- Digital industries **churn out** new technologies, programming languages, tools and frameworks on a daily basis …
- But this does not relieve us form **engineering** (which is what strangely many developers seem to believe …)
- My simplified “technology stack”:

Technology

Programming Languages, Hardware, Frameworks

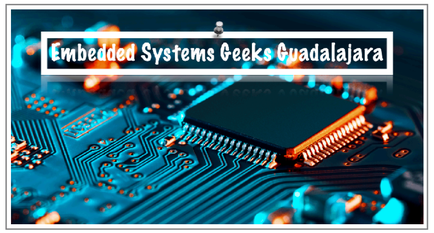
“Technology Confusion”



- Digital industries **churn out** new technologies, programming languages, tools and frameworks on a daily basis …
- But this does not relieve us form **engineering** (which is what strangely many developers seem to believe …)
- My simplified “technology stack”:

Technology	**Programming Languages, Hardware, Frameworks**
Engineering	**Modeling Languages, Analysis Methods & Tools**

“Technology Confusion”



- Digital industries **churn out** new technologies, programming languages, tools and frameworks on a daily basis …
- But this does not relieve us form **engineering** (which is what strangely many developers seem to believe …)
- My simplified “technology stack”:

Technology	**Programming Languages, Hardware, Frameworks**
Engineering	**Modeling Languages, Analysis Methods & Tools**
Foundations	**Mathematics, Semantics, Control Theory, Physics, …**

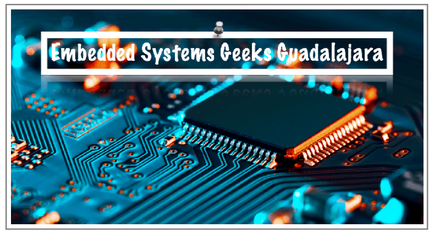
“Technology Confusion”

- Digital industries **churn out** new technologies, programming languages, tools and frameworks on a daily basis …
- But this does not relieve us form **engineering** (which is what strangely many developers seem to believe …)
- My simplified “technology stack”:

Sadly, most developers end up being stuck here …

Technology	Programming Languages, Hardware, Frameworks
Engineering	Modeling Languages, Analysis Methods & Tools
Foundations	Mathematics, Semantics, Control Theory, Physics, …

“Technology Confusion”

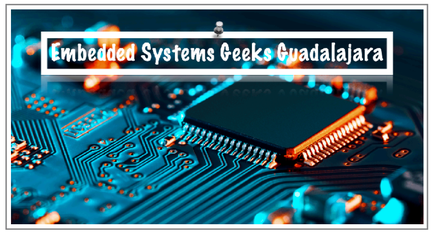


- Digital industries **churn out** new technologies, programming languages, tools and frameworks on a daily basis …
- But this does not relieve us form **engineering** (which is what strangely many developers seem to believe …)
- My simplified “technology stack”:

Some manage to make the transition to engineering …

Technology	Programming Languages, Hardware, Frameworks
Engineering	Modeling Languages, Analysis Methods & Tools
Foundations	Mathematics, Semantics, Control Theory, Physics, …

“Technology Confusion”

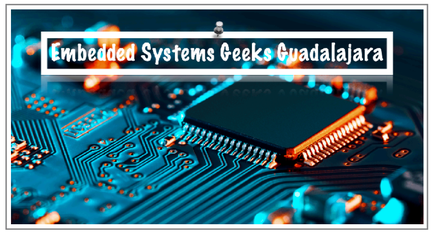


- Digital industries **churn out** new technologies, programming languages, tools and frameworks on a daily basis …
- But this does not relieve us form **engineering** (which is what strangely many developers seem to believe …)
- My simplified “technology stack”:

Technology	**Programming Languages, Hardware, Frameworks**
Engineering	**Modeling Languages, Analysis Methods & Tools**
Foundations	**Mathematics, Semantics, Control Theory, Physics, …**

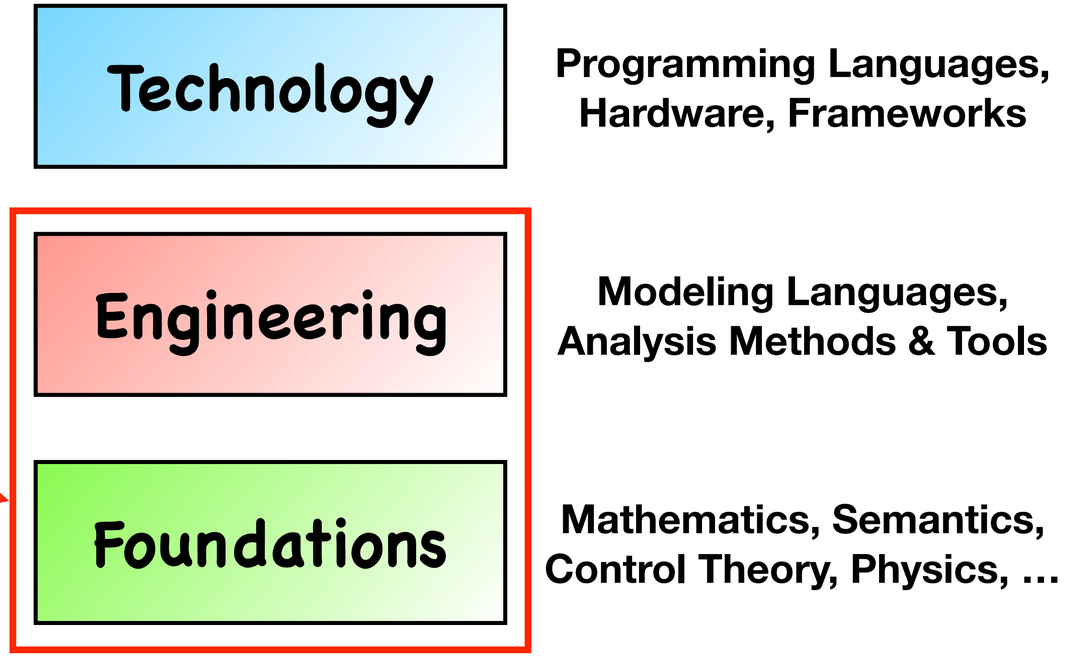
Very few are concerned with sound foundations …

“Technology Confusion”

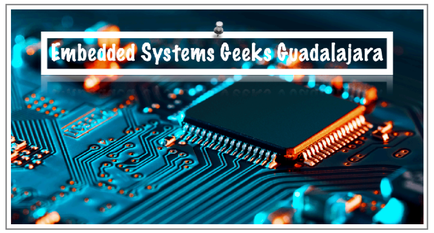


- Digital industries **churn out** new technologies, programming languages, tools and frameworks on a daily basis …
- But this does not relieve us form **engineering** (which is what strangely many developers seem to believe …)
- My simplified “technology stack”:

New tech. hypes, business jargon, buzzwords, do not replace either of them!

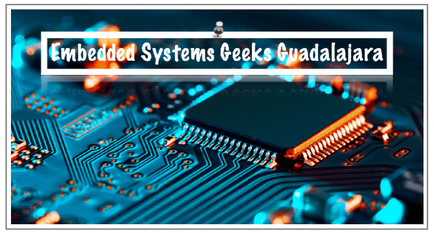


Aspiration for the Group



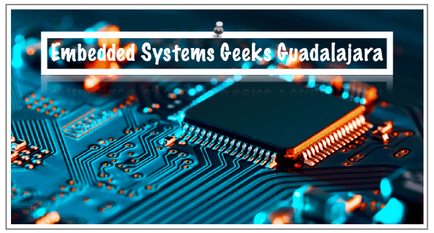
- Not just to illustrate **technology** but also talk about **engineering**
- Not blindly apply **engineering**, but also question/review **foundations**
  - Have a balance of talks that focus on these three key areas.
  - Illustrate where the overlaps, risks and pitfalls are.
  - Draw parallels between “traditional” software engineering and embedded systems HW/SW engineering.
  - How do IoT, cloud-based and A.I. topics fit into this picture?
  - Can we share case studies, projects, experiences?

Topical Area: MCU

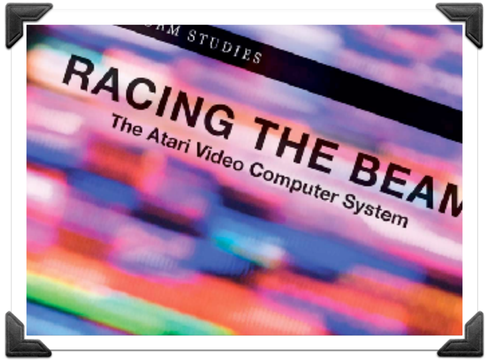


- Microcontrollers are perhaps the **easiest** and **cheapest** way to explore the world of embedded systems engineering:
  - MCU typically provide **everything** you need in a single package: CPU, static RAM / flash memory, analog & digital IO pins, standard protocols like UART, I^2C, SPI, etc.
  - plethora of possible options for low-cost chips and boards
  - active and vibrant communities, e.g., around Arduino (AVR chips). Raspberry Pie Pico (RP2040), ESP32, STM32, and so on
  - typically include tool-chains for development, cross-compilation, deployment and testing (Arduino IDE, Eclipse-based, etc.)

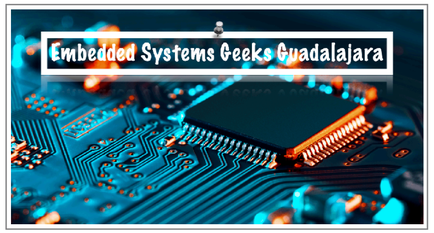
What makes MCUs interesting?



- Often come with restrictions, e.g., on available **memory**, **clock speed**, **communication protocols**, input / output **interfaces**, and so on.
- How can we make *most of the chip* in achieving our goal? E.g.,
  - try not to waste precious RAM or flash memory …
  - make the available features of a chip work together …
  - use nifty ways to circumvent constraints …
  - be aware of the low-level impact of your high-level code
  - understand in detail the **functional** and **timing** properties of a chip
- Watch: Pitfall postmortem and design challenges (David Crane) on the Atari 2600

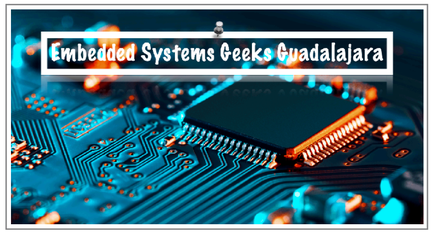


MCU Challenges



- Even simple ($4 USD MCUs) come with **extensive** data sheets:
  - Atmega328P (294 pages)
  - RP2040 (639 pages)
- Open-source software is often **amateurish** and **sloppily** written …
  - But well, let's not complain — it is free and open source after all.
- Still, high-end software-engineering principles are seldomly applied, even by professional vendors (docs lacking, APIs incomplete, no TCK, etc.)
- Firmware and libraries may be **bloated**, limiting their combined use.
- Even low-cost MCUs are rather complex these days …
- You see now that I am Germany 🇩🇪: that is, **complaining** a lot …

Future topics on MCU



- We may look at particular **chips** and **boards**.

  ➡ Am definitely looking forward to volunteers here to give talks.

- We can share some personal experience with particular projects, framework, encountered obstacles, and HW/SW design solutions. E.g.,

  ➡ How to tease out a VGA signal from an ESP32 or RP2040.

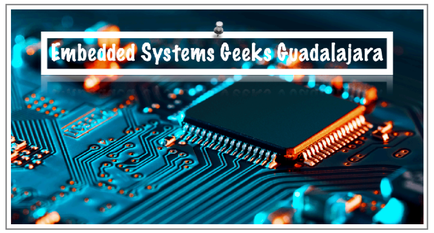
  ➡ Developing a **Pong!** game on a microcontroller of your choice …

  ➡ Third-party libraries and your experience using them …

  ➡ Using wireless **networking** and TCP/IP communication …

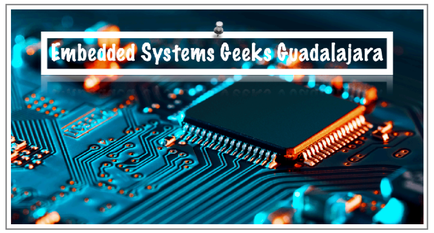
  ➡ Nifty solutions to difficult problems (and how you found them!)

A small problem …



- The modulus operator "`x % m`" of C/C++, Java, Python, etc., is typically evaluated via a hardware **CPU instruction**.
- Assume we have a very simple CPU that does not support hardware <u>multiplication</u> and <u>division</u>.
- However, you will have arithmetic, bitwise and bit-shift operations at your disposal.
- Can you implement an efficient algorithm / strategy to calculate the remainder of some number `n` modulus 3 and 5?
- Can you, moreover, give a mathematical analysis that your algorithm is **correct**, and evaluate the number of clock cycles it takes on some MCU?

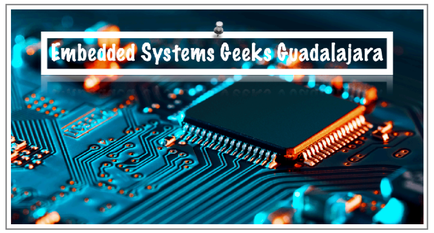
A small problem …



- The modulus operator "`x % m`" of C/C++, Java, Python, etc., is typically evaluated via a hardware **CPU instruction**.
- Assume w dware multiplicat
- However, ons at your disposal.
- Can you i e the remainder of some number `n` modulus 3 and 5?
- Can you, moreover, give a mathematical analysis that your algorithm is **correct**, and evaluate the number of clock cycles it takes on some MCU?

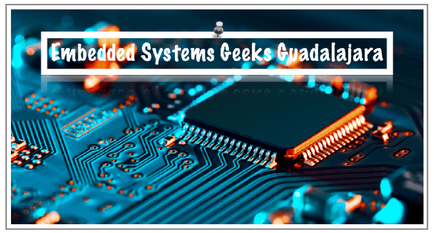
Send me your solutions (e.g. in C/C++, Java or Python) and we may look at them in the next Embedded Systems Geeks social. There may be a price for the neatest solution …

Field Programmable Gate Arrays



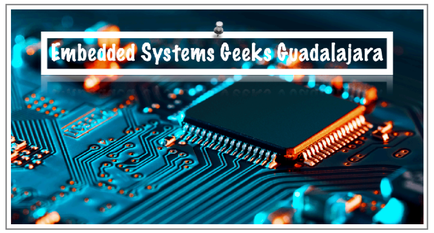
- Field Programmable Gate Arrays (**FPGA**) are altogether a different technology:
  - MCU have a specified and fixed behavior that is hardwired into the chip
  - FPGA can be programmed to execute any **kind of logic circuit**
  - Think of them as construction kits for digital circuits that consists of
    - (i) elementary logic gates and functions (AND, OR, NOT)
    - (ii) flip-flops to store values
    - (iii) auxiliary components like block RAM, PLLs, output drivers, etc.
- With FPGA, we can create our CPU/MCU/SoC entirely from scratch …

FPGA advantages



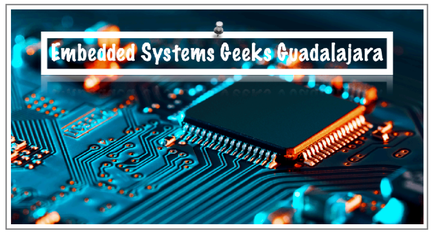
- FPGAs were pricey in my university days but are now widely available and relatively cheap (i.e., you'll find basic boards for less than $15 USD).
- We are not constrained by a particular fixed **architecture** for the CPU and peripherals, as is the case for **MCU** units.
- We can build an entire system of interconnected components within a single (sufficient powerful) FPGA, known as a System on Chip (**SoC**).
- Specialized tasks can be carried out very efficiently (cryptography, image and sound processing, data processing for A.I., etc.)
- There is a **wealth** of open-source IP components that we can reuse.
- They are extensively used in retro-computing to duplicated chips that have gone out of production or entire vintage computers and games consoles …

FPGA advantages

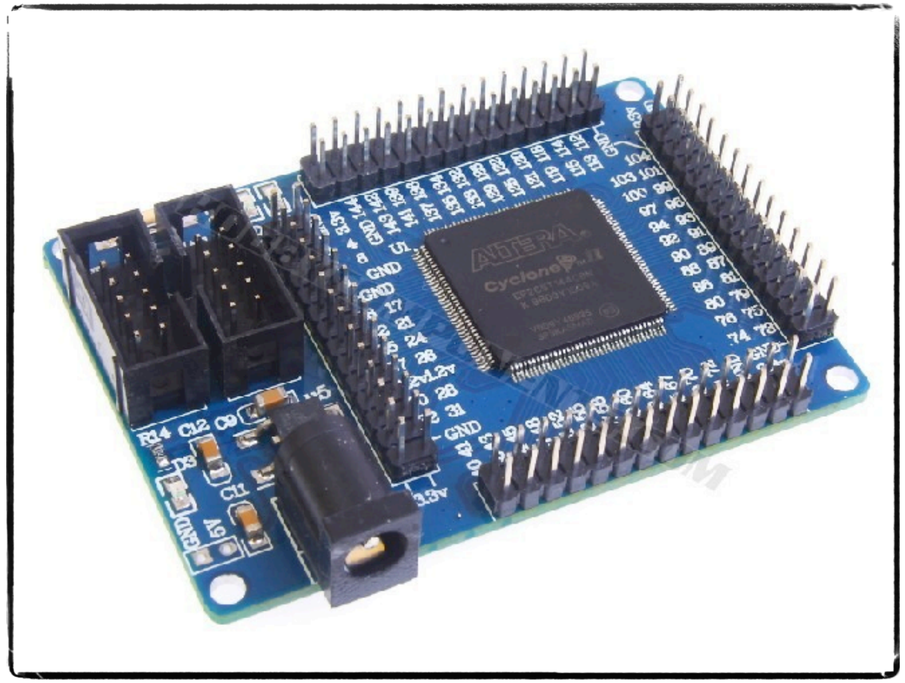


- FPGAs were pricey in my university days but are now widely available and relatively cheap (i.e., you'll find basic boards for less than $15 USD).
- We are not constrained by a particular fixed **architecture** for the CPU and peripherals, as is the case for **MCU** units.
- We can build an entire system of interconnected components within a single (sufficient powerful) FPGA, known as a System on Chip (**SoC**).
- Specialized tasks can be carried out very efficiently (cryptography, image and sound processing, data processing for A.I., etc.)
- There is a **wealth** of open-source IP components that we can reuse.
- They are extensively used in retro-computing to duplicated chips that have gone out of production or entire vintage computers and games consoles …

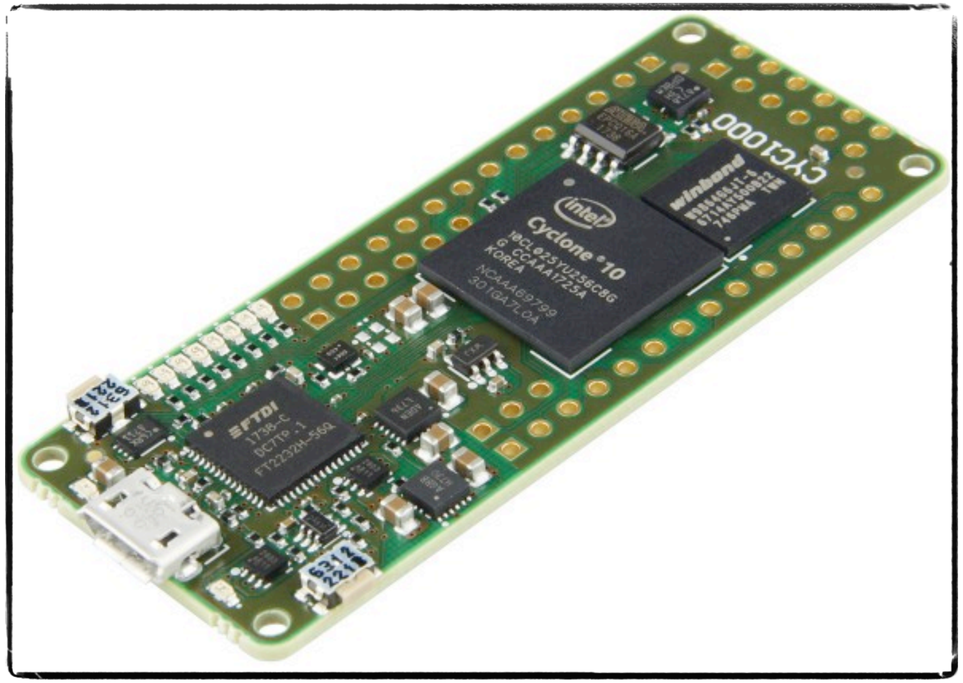
Two low-cost FPGA boards



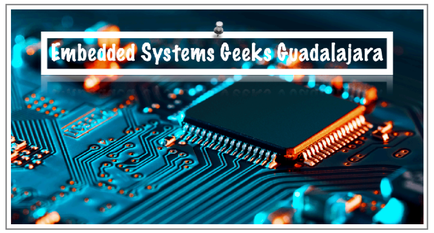
Cyclone II



Cyclone 10 LP



FGPA disadvantages



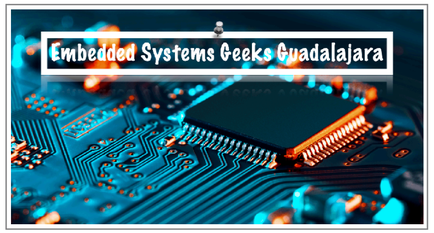
- Step **learning curve** to learn how to program them.
- Buy doing things from scratch, we sometimes have to develop our own **tool-chain**, **cross-compiler**, development **framework**, etc.
- Alternatively, there exist proprietary solution that supply us with a GUI to configure and build our SoC (Nios II / MicroBlaze soft processor cores)
- There are also open-source solutions, e.g., for RISC-V CPUs and SoC.
- But even using these is not a walk in the park:

➡ we may need some knowledge of Verilog and/or VHDL

➡ we need to appreciate synthesis issues (e.g. timing violations)

➡ we may need to understand on-chip communication bus protocols

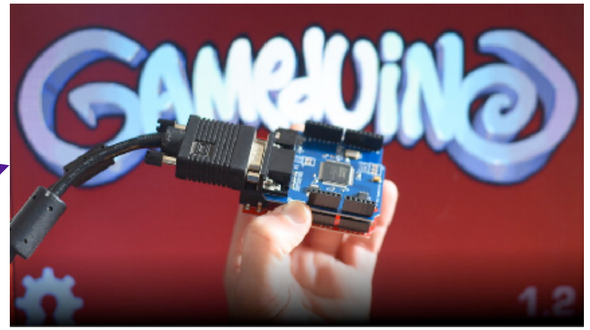
The J1 CPU for FPGA



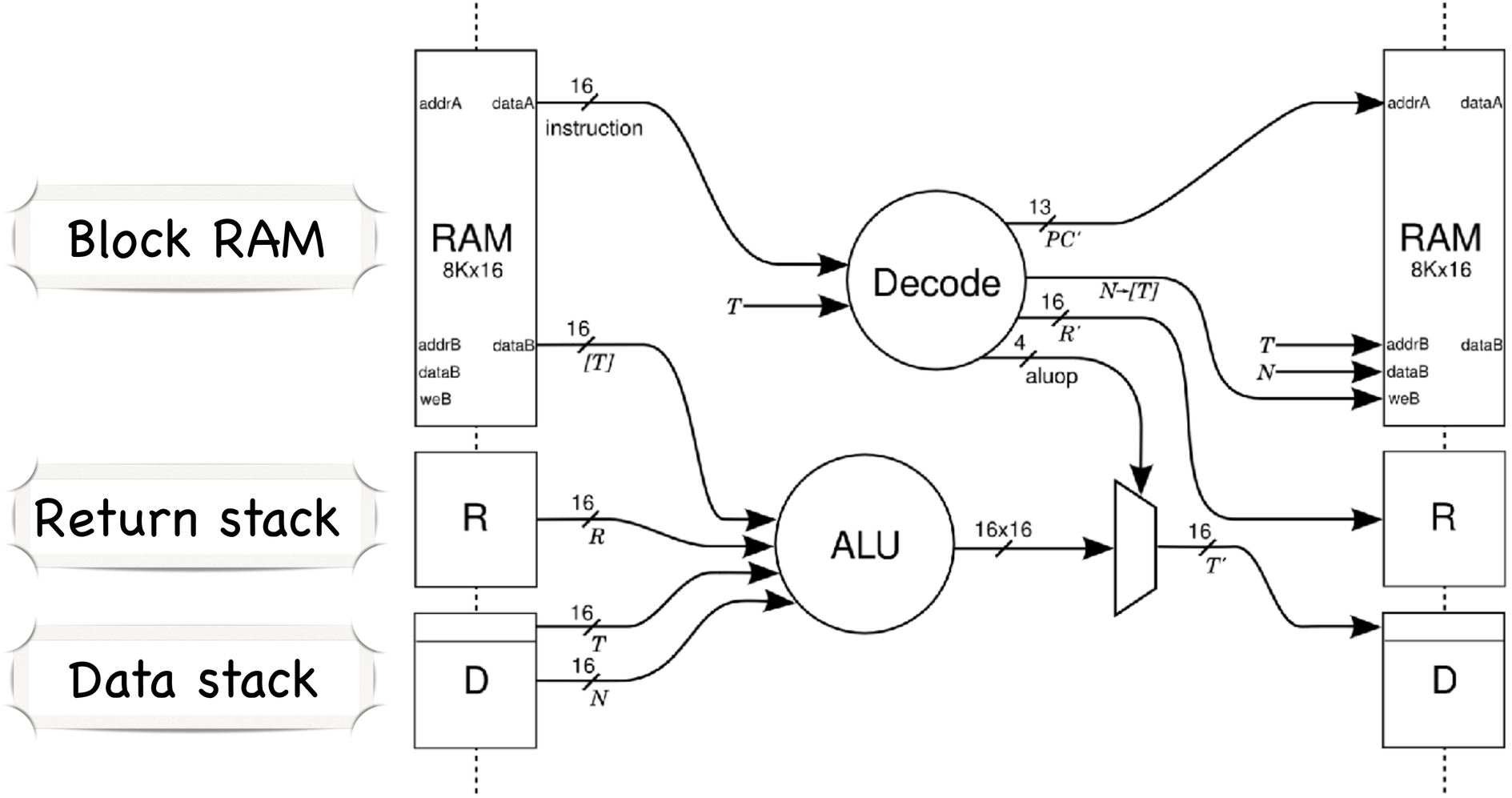
James Bowman

- The J1 is a **tiny** CPU for building a Forth runtime system.
- It fits into about **120 lines** of Verilog HDL code.
- It is stack based and does not provide any registers.
- It does not provide any pipelined architecture …

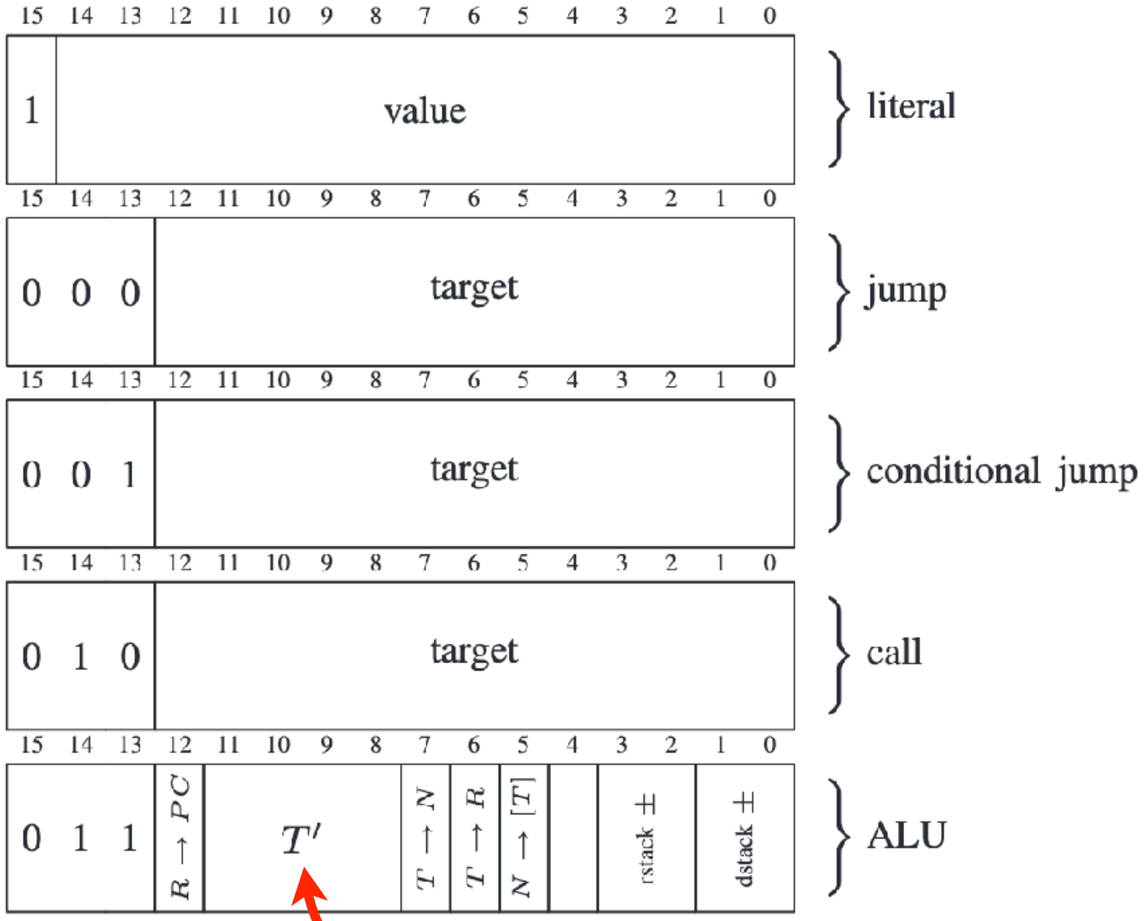
  ➡ … but can run above 100 MHz (short critical signal path)
- Neither provides an MMU or interrupt controller.
- Used as a **coprocessor** in the Gamedruino.
- Quite a bit of **fun** to play with …



J1 machine model

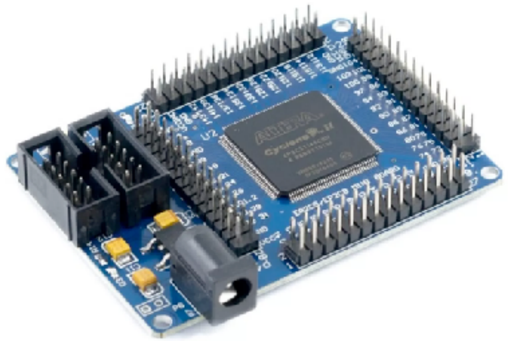


J1 instruction encoding

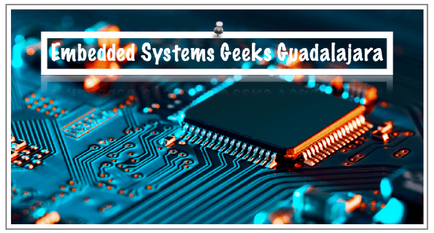


code	operation
0	T
1	N
2	$T+N$
3	$T$and$N$
4	Tor$N$
5	Txor$N$
6	$\sim T$
7	$N=T$
8	$N<T$
9	Nrshift$T$
10	$T-1$
11	R
12	$[T]$
13	Nlshift$T$
14	depth
15	Nu<T

TABLE II: ALU operation codes

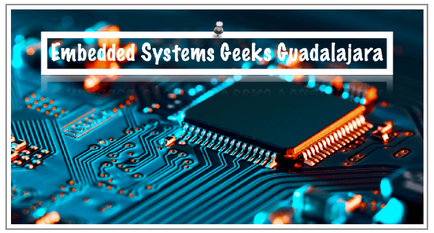


Extending the J1 core



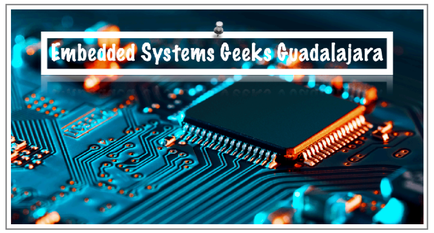
- As part of a self-study project on to learn Chisel, I extended the original J1 core with the following **features**:
  - Support for 128 KB addressable program memory (instead of 8 KB)
  - Absolute as well as **relative** branches
  - Sign-extended immediate pushes (15 bit)
  - Several alternative configurations for bit shifters
  - Auxiliary instructions for byte / word swapping
  - Memory protection, e.g., for boot-loader code
  - Run-time stack **overflow** and **underflow** checks, **and more** …

Extending the J1 core



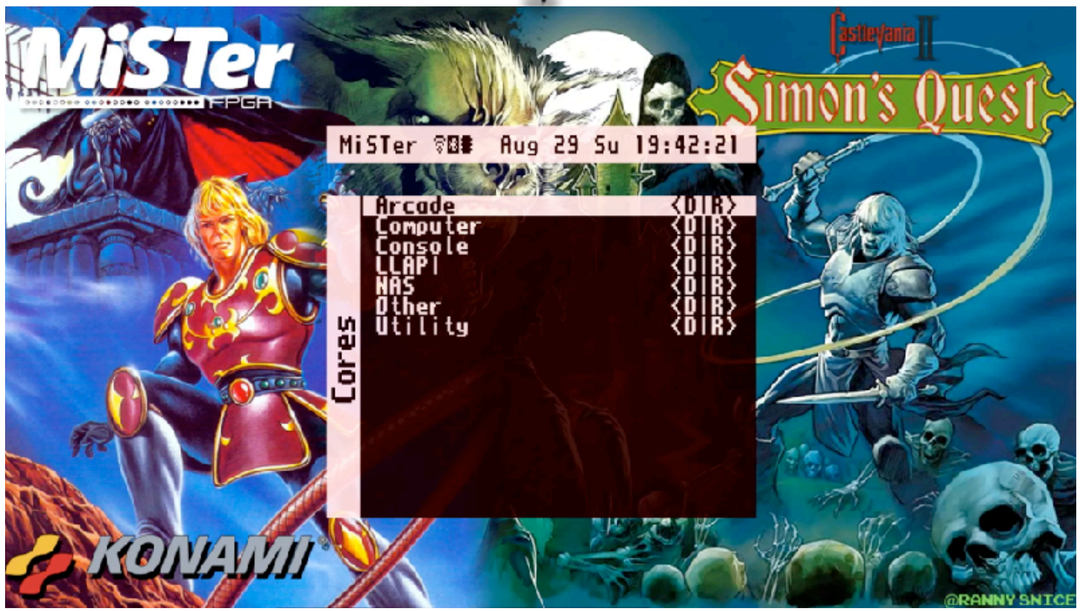
- Problem: implement a **cross-compiler** with **hardware emulator** as part of a remote development environment.
  - ➡ I have already done that for the original J1 core in **SML**
  - ➡ I started to do it for the extended J1 core in **Scala**
  - ➡ Am using this also as a means to **study** and **evaluate** Scala 3 …
- The project is available on GitHub:
  - https://github.com/Frank-Zeyda/j1.chisel
- I might talk about it in one of the subsequent meet-ups, if interest …

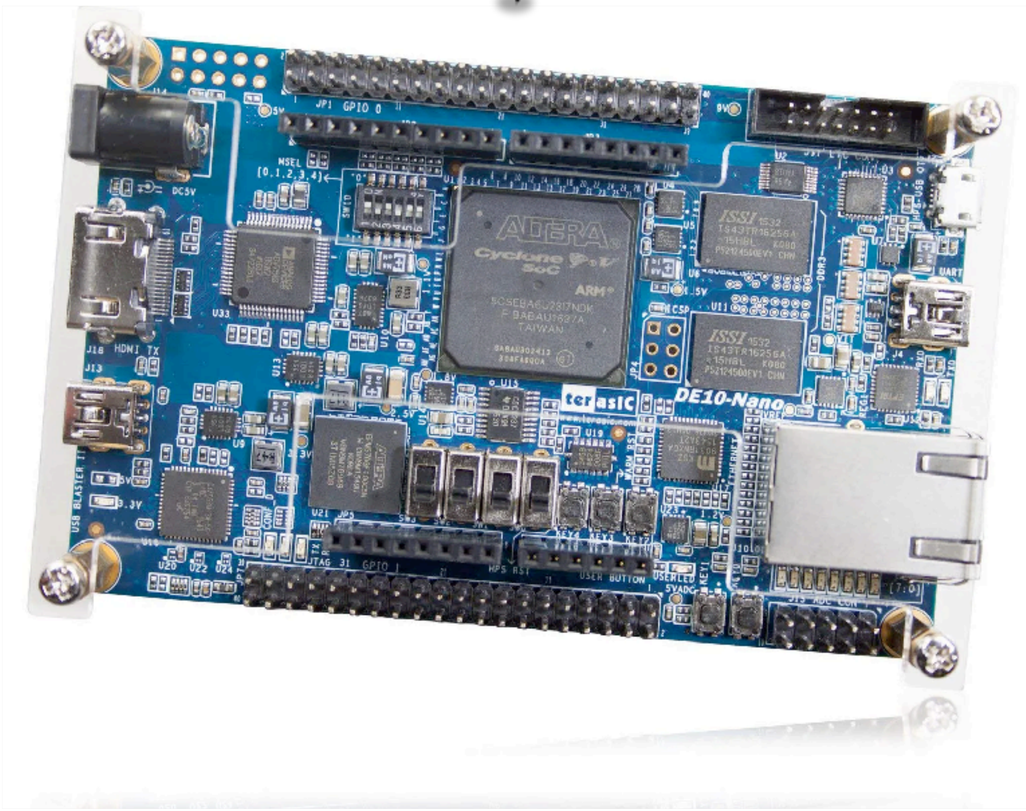
Future topics on FPGA



- Introductory course: explaining the basics of FPGA programming.
- A simple introduction to **Verilog** and/or **VHDL**.
- Functional FPGA programming languages: Clash, Chisel, Lava, etc.
- Hands-on **demonstrations** on how to use respective EDA tools.
- Showcase for building an FPGA-based SoC on a complex board.
- Having a look at the MiSTer project for replication of vintage consoles.
- Collaborative project to produce a simple FPGA-based **arcade game**!

The MiSTer Project

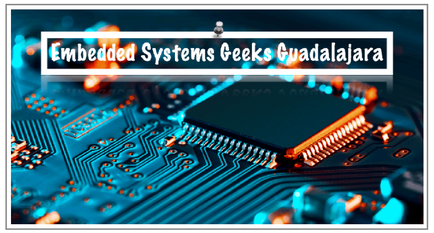




A bit more pricey, about $ 250 USD for the FPGA alone

The MiSTer Project

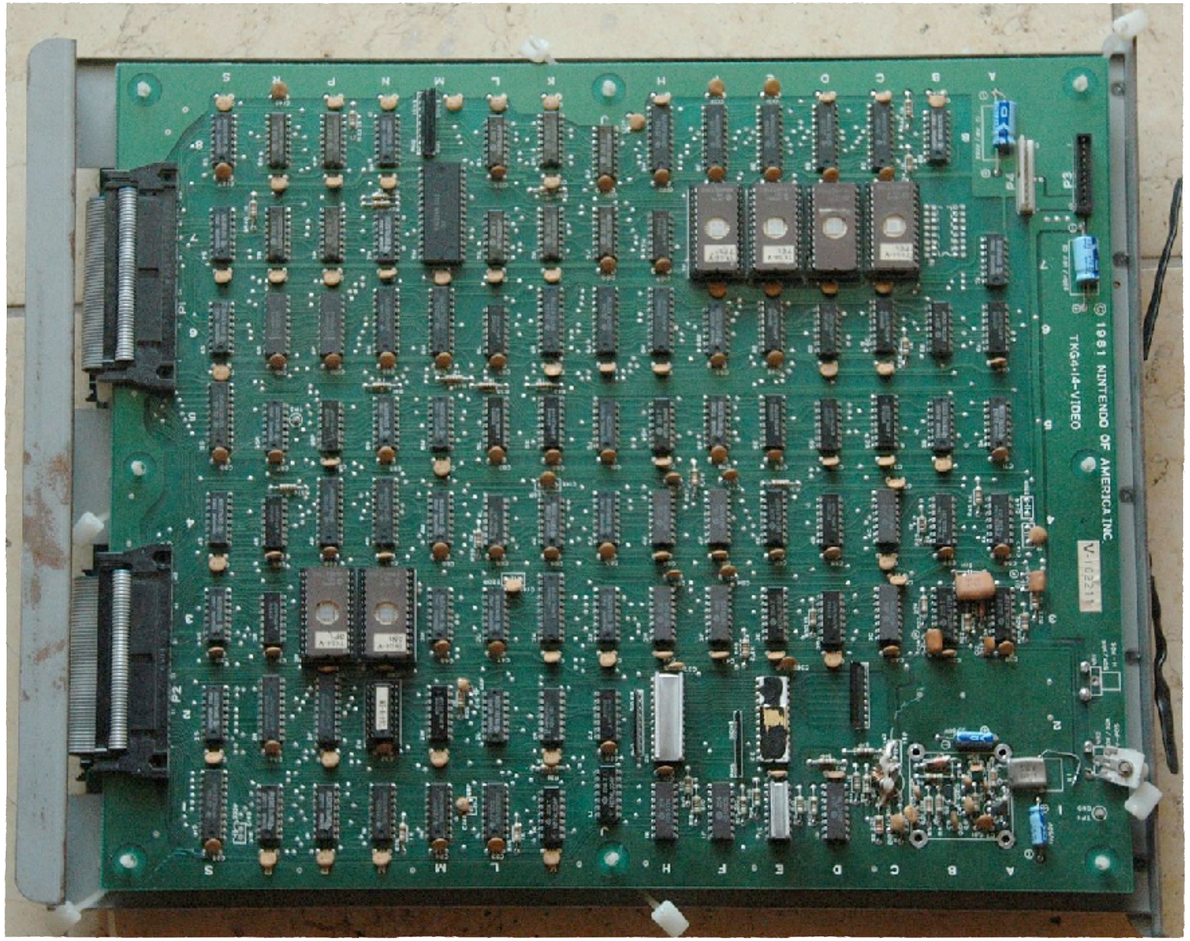
Future topics on vintage HW



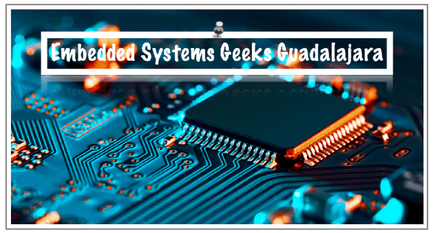
- Look at some emulators and tool-chains to program vintage home computers, such as the **Atari 2600**, **C64**, **Amstrad CPC**, **Amiga**, etc.
  - Perhaps some events where we play games on our favorite consoles …
- Opportunity to **study** and **teach** low-level hardware programming …
- … and associated issues such as generating a **VGA** or **Composite** video signals, sound (**ADC** & **DAC**), connecting controllers, and so on …
- Issue of reverse engineering game PCB and replicate them in FPGA.
- Explore the demo scene and how the community pushed the boundaries of vintage hardware.
- Design and create our **very own** vintage computer together …

Folks are actually doing that …

Original PCB of Donkey Kong



Or rebuild a 6502 CPU from scratch …

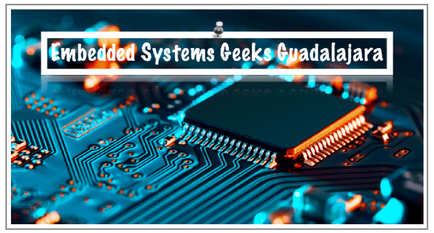


Miscellaneous topics

- There is even more to embedded systems design:
  - **Signal processing**, **coding**, and **telecommunications**
  - **PCB design** and **layout**
  - **ASIC** design (application-specific integrated circuit)
  - **Hacking** and **cyber-security** concerns

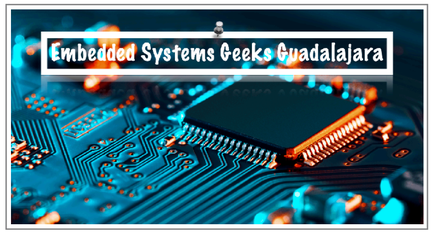
    ➡ side-channel attacks, serial terminal,

Conclusion



- We had a brief look at a few topics surrounding embedded systems
- There is a lot of scope for areas to be covered
- Naturally, they divide into **technology**, **languages**, **devices** & **boards**, and **engineering** approaches.
- It is your turn to suggest what you like to learn more about.
- Workshops could include practical guidance in getting started with a particular **hardware** or **board**.
- Specialized courses, i.e., on **FPGA** or **SoC** design may be more interesting to professional working in the area.

Future Planning



- What **topics** would you like to hear more about?
- What **interests** would you like to be represented?
- Are you hoping to **work** in hardware / embedded systems design?
- Would you be interested in attending **workshops**?
- Would you be keen to give a short or longer **presentation**?
- Would you be keen to **collaborate** on a toy project?

Thanks for Listening!

Time for questions and comments.