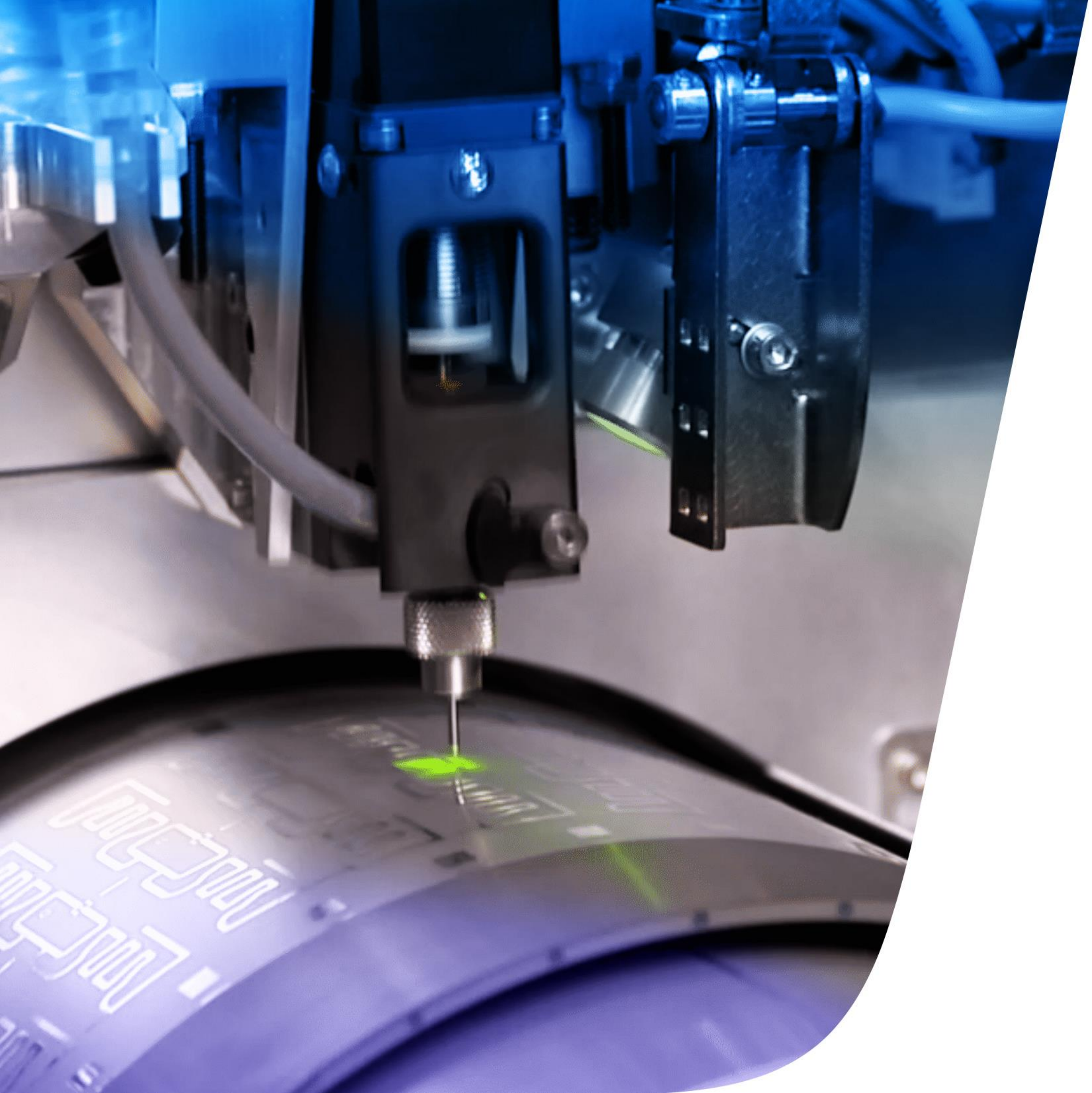
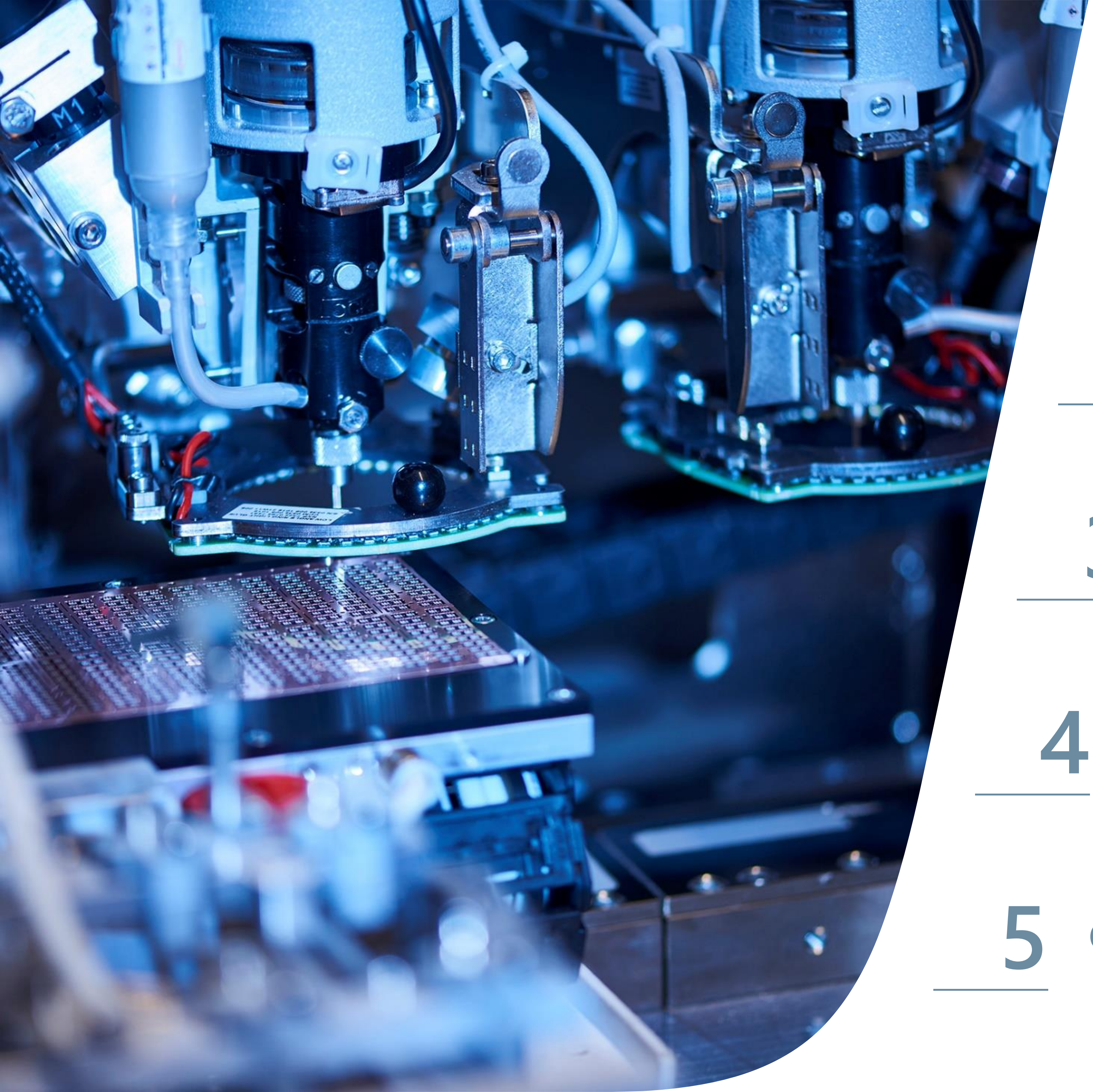


itec Redefining semiconductor manufacturing

Why is Ada better then Rust

But everybody is still using C++

040coders meetup April 18, 2024 | AnSp @IMEC



1 A bit of Intro

2 History of Languages

3 Feature sets

4 The verdict

5 Crystal ball

ITEC established as the equipment department of Philips Semiconductors

1991

ADAT2 6" die bonder

1992

µParset discrete tester for final and wafer test

2000

Philips Semiconductors became NXP

2006

ADAT3 8" die bonder/ sorter

2006

µParset power discrete tester for final and wafer test

2006

ITECore Smart Manufacturing

2010

PHIXEL IHW post-seal in-tape inspection handler

2012

ADAT3 8" flip chip

2014

30+ YEARS OF REDEFINING SEMICONDUCTOR MANUFACTURING

In 1987, “Breakthrough in Manufacturing” (BIM) was simply a new concept. Now, over 30 years later, BIM aptly describes ITEC`s own approach to pushing the boundaries of semiconductor productivity. We develop breakthrough technologies and build state-of-the-art equipment that enables our customer to mass-produce semiconductor products of the highest quality.

2016

ADAT3 XF 12" die bonder/ sorter

2016

ADAT3 XF Tagliner inlay die bonder

2017

NXP Standard Products became Nexperia

2018

PHIXEL CMR optical inspection for contact(less) modules

2021

ITEC became a separate entity

2022

nanoParset fastest discrete tester for final and wafer test

2022

PHIXEL WIF post dicing wafer AOI

2023

ADAT3 XF 12” flip chip

ITEC solutions summary

ADAT3 Assembly Platform

- Lowest Cost of Ownership
- Process portability
- Thinner wafers
- Flip-chip iso wires
- Placement accuracy
- 360º optical inspections
- Predictive maintenance
- Versatility and flexibility

Record-breaking output

Parset Test platforms

- Lowest Cost of Ownership
- High test speed
- Multi-site testing
- Integrated digital and analog functions
- Test platform consolidation and wide test coverage

Lowest test cost for low-pin-count semiconductors

Inspection Platforms

- Lowest Cost of Ownership
- Tailored for specific mass-production applications
- Powered to drive uncompromising Quality
- Integration in IT infrastructure for traceability

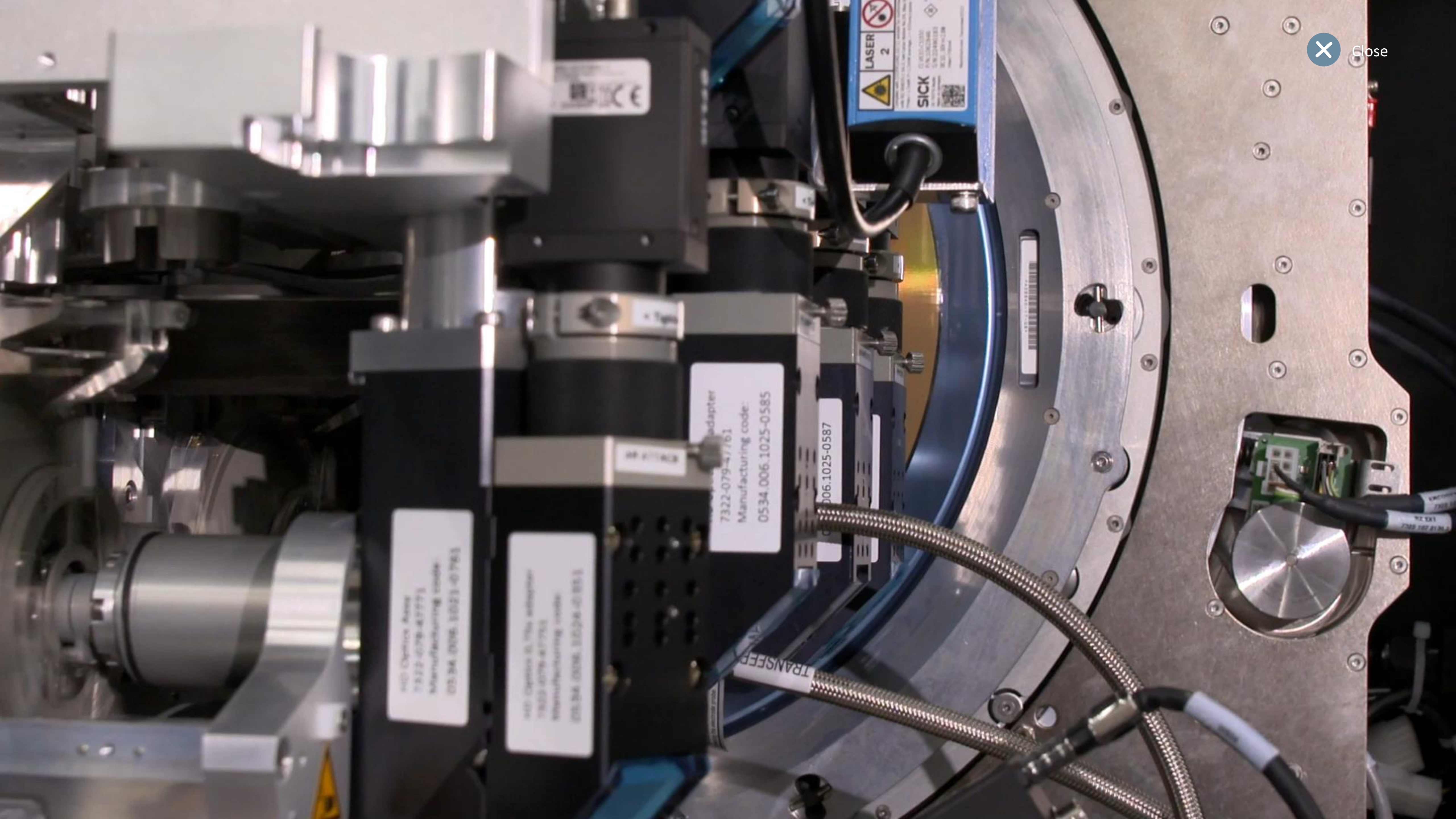
Unbeatable efficiency

Smart Manufacturing

ITECore Smart Manufacturing

- Full die-level traceability
- Big data analytics
- Data fusion
- Autonomous loops and machine learning

Leading in Industry 4.0 for mass-production


Close
LASER
2
SICK
adapter
7322-079-47761
Manufacturing code:
0534.006.1025-0585
006.1025-0587
TRANSFER

ITEC uses Ada

ITEC originated from the ‘Bedrijfsmechnisatie’ van Philip Semiconductors.
It is building chip assembly equipment since the 70’s.

The first automated equipment was based on RTL2 (on DEC VAX).
Both RTL2 and Ada have a Pascal like syntax.
Most of the conversion was performed automatically halfway the 90’s.
(our first entry in the source repository is from October 24, 1997)

The machine control applications runs on a standard off-the-shelf ASUS motherboard and Windows 10 64 bits (IoT).

History of Ada

- A request for a coming program language was initiated by the DoD in 1975
- In 1979 the name 'Ada' was selected in honor of Ada Lovelace
- Ada 83 was the first standard version
- Boeing used it for the B777 SW development
- Ada '95 was the next standard version
- AdaCore (former ACT) was formed in 1996 And is one on the leading compiler vendors
- Current ISO standard is Ada 2012[1] (after Ada 2005) and uses GCC as backend

- **C++** started as C with classes in 1979
- Initial version (as extension to C) in 1985
- ISO Standards C++98, C++03, C++11, C++14, C++17, C++20[2]

- **Rust** started in 2006
- First official release in 2015
- Mozilla supports it since 2009
- Since 2021 there is a Rust foundation
- There is no Rust standard[3] (yet)

---

1. https://www.adaic.org/advantages/ada-2012/
2. https://www.iso.org/obp/ui/#iso:std:iso-iec:9899:ed-4:v1:en
3. https://github.com/rust-lang/rfcs

Basic language stuff

All 3 languages are structural and meant for systems programming.
In the basic language constructs the semantics are different, but the features the same

	Ada	Rust	C++
Basic types	Character, String Integer, Float, Boolean, Enum	i8, string i32, f32	char, std::string int, float bool, enum
Conditional	if .. then .. else.. endif	If .. { .. } else { .. }	If (..) {..} else {..}
For loop	for i in l .. u loop .. end loop	for i in l .. u { .. }	for (int i = l; i < u; i++) {}

While loop	while True loop ..
	end loop
		while true { .. }	while (true) { .. }

	case n is
	when 1 .. 5 => ..
	end case
		match n
		{
		1 .. 5 => ..
		}
			switch (n) {
			case 1.. 5:
			...
			break;

Both Ada and Rust support strong typing with range checking

Functions

Ada	Rust	C++
Procedure Foo (input : Integer; output : in out Integer)	fn Foo (input : i32, output : &mut i32)	void Foo (int : input, int : &output)
Integer function Foo (input : integer)	fn Foo (inout : i32) -> i32	int Foo (int : input)
in parameter (default) in out parameter out parameter	By value or by reference (&) By mutable reference (&mut)	By value (or by const reference) By reference (&)

All languages also support passing a reference to a function as parameter.

Note 1: Constant parameters are default by Ada and Rust. Changing a parameter requires to explicit allow that.
Note 2: Ada did not allow in out parameters with functions until the 2012 standard to prevent side effects.

Memory allocation

	Ada	**Rust**	
Alloc / free (allocate bytes)	Not in the language	Alloc / free	Alloc / free
New / delete (allocate elements)	New is the standard Delete trough a generic Unchecked_Deallocation	New / Drop	New / delete
Shared pointers (allocate elements with reference count)	GNAT compiler offers a solution through Storage_Pools	Box, Rc or Arc	std::shared_ptr

Both Ada and Rust will do bounds checking to prevent buffer overrun

Object Oriented

	Ada	Rust	C++
How implemented	Define a record with abstract and/or tagged Define procedures and/or functions on the record	Define a struct Define the methods in an impl block for that struct	Define a class with data and methods
Multiple inheritance	Ada has the keyword interface for a pure abstract interface. Add these interfaces to an abstract record.	Define a trait with functions Implement this Trait for specific struct	Define a class with virtual functions Derive a new class using this base class
Constructors	If the OO record is defined as controlled: Initialize, Adjust, & Finalize	Construct by initialize all its field at once. Deconstruct via Dropped	Always available

External

Concurrency

All languages support concurrency

- Threads (tasks in Ada)
- Mutex, semaphore, lock/unlock
- Events

Ada Protected Objects

- Object can only be accessed by functions
- The protected object ensures the mutually exclusivity
-
```
Protected Counter is
  function Get return Integer;
  procedure Increment;
  procedure Decrement;
private
  Value : Integer := 0;
end Counter;
```

Ada tasks

- A thread with implicit synchronization through rendez-vous
- A task has entries
  Function calls of a task

```
task body T is
        LocalInteger : Integer := 0 ;
    begin
        accept put (A : in Integer) do
            LocalInteger := A ;
        end put ;

        accept get (A : ouInteger) do
            A := LocalInteger ;
        end get ;

    end T ;
```

Ada Attributes

Ada has attributes to query properties or perform actions on Ada entities (types, objects, subprograms)

- Integer’First and ‘Last
  (smallest and largest Integer)
- Character’Pos and ‘Val
  (convert from/to ASCII value)
- Enum’Prev and ‘Succ
  (for the previous or next enum value)
- <type>’Img to get the string representation

```
function String_To_Enumeration
      (Str         : String;
       Fail        : To_Enum_Type := To_Enum_Type'First;
       Log_Invalid : Boolean := True)
      return To_Enum_Type
   is
      CStr : constant String := Trim (Str, Left);
      Idx  : Integer := CStr'Last;
   begin
      Idx := Index (CStr, " ");
      if Idx = 0 then
         Idx := CStr'Last;
      else
         Idx := Idx - 1;
      end if;

      if not Leading_No_Case (CStr, Prefix) then
         return To_Enum_Type'Value (Prefix & CStr (CStr'First .. Idx));
      end if;
      return To_Enum_Type'Value (CStr (CStr'First .. Idx));
   exception
      when others =>
         if Log_Invalid then
            Log_Text ("String_To_Enumeration failed; Str=" & Str & ",
                  Prefix=" & Prefix & ", Caller=" & Caller);
         end if;
         return Fail;
   end String_To_Enumeration;
```

Other support

All support:

- Generics (templates)
- Collections (arrays, vectors, maps) and iterators
- File I/O (raw and streaming)
- Binding to C libraries
- It is hard to have a C++ binding in Ada and Rust
  They have binding generators, but the result is not guaranteed
  The C++ Application Binary Interface (ABI) is not supported

Invariants, SPARK, Ravenscar

Ada 2012 support **invariants** (& pre and post conditions) on types and subprograms
They can be turned on/off with the pragma Assertion_Policy

```
type Stack is private
     with Type_Invariant => Count(Stack) >= 0;

function Sum (A, B : Number) return Number
  with
    Pre  => A <= 0.0 and B >= 0.0,
    Post => Sum'Result = A + B;
```

SPARK is an Ada subset based on the 2012 aspect notation which allows statistic verification.
Not in the subset are things like access types, function side effects, exception handling

The **Ravenscar** profile is a subset of the tasking model for use in high integrity and real-time applications. The subset ensures full analyzability of the code.

It prohibits the use of things like dynamic properties, task entries, task termination and more.

Tools

Besides the compiler, some additional tools are available to improve the code

	Ada	**Rust**	**C++**
Syntax checker	Gnat Check	Clippy	Lint
Static checker	Code peer		clang-static-analyzer
Pretty printer	Gnat pp	Rustfmt	Clang-format

External

Language summary

Ada was designed with critical missions in view and was later extended with ‘modern’ SW design patterns like OO and abstract interfaces.

C++ was started as C with classes, but still lacked ‘safety’ features. Some are added (like smart pointers) but makes things more complex, because there are various options to solve a problem. To help developers build safer C code there a several guides (MISRA[2], C++ Core guidelines[1])

Ada never became a mainstream program language, and the C/C++ weak points are still there, new languages to solve these are designed. **Rust** is one of them with potential and the first to be part of the Linux kernel (next to assembly and C).
Another recognized advantage with Rust is their build system Cargo.

---

1. https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines
2. https://misra.org.uk/

Language summary

With all 3 languages you can implement probably everything you want.
And with discipline and structure safe as well.
Ada and Rust are intrinsic safer than C++, but chaos can be created in them as well.

A good application starts with a good architecture.

So why still C++?

Legacy code bases are in C/C++.
Already hard to maintain and adding another language just makes it more complex.

And it is all about the eco system (knowledge, forums, tools, libraries, people) -> the big guy wins.

Language	Tiobe index	Ratings
Ada	24	0.76%
C / C++	2 / 3	=10.21% / ^9.76%
Rust	19	^1.03%

From www.tiobe.com

Verdict AI (Google Gemini)

I need to select the program language for a mechatronic production machine, which includes motion control, vision inspections and all kind of I/O. Can you give me a comparison between C++, Rust and Ada for such a machine control application?

Verdict on C++

Pros:

- Most mature language of the three, with a vast ecosystem of libraries and tools for real-time systems and embedded programming.
- Highly performant, offering fine-grained control over memory management.
- Extensive community support and learning resources available.

Cons:

- Manually memory management can be error-prone, leading to memory leaks and crashes if not done carefully.
- Complex syntax can make code harder to read and maintain, especially for large projects.
- Not as safe as Rust by default, requiring stricter coding practices to avoid vulnerabilities.

Verdict on Rust

Pros:

- Focuses on memory safety, preventing dangling pointers and buffer overflows that can plague C++ programs.
- Modern language with features like pattern matching for cleaner and more concise code.
- Offers good performance and real-time capabilities.

Cons:

- Younger language with a smaller community and ecosystem compared to C++.
- Steeper learning curve due to its ownership system and borrowing concepts.
- Might have a smaller pool of developers familiar with Rust for embedded systems.

Verdict on Ada

Pros:

- Designed specifically for embedded systems and real-time applications.
- Enforces strong typing and program correctness, leading to more reliable code.
- Offers features like tasking and rendezvous for coordinating concurrent processes.

Cons:

- Not as widely used as C++, potentially leading to fewer libraries and tools.
- Can be considered more verbose compared to C++ or Rust for some tasks.
- Finding developers familiar with Ada might be more challenging.

Overall recommendation

- If you prioritize performance, a vast ecosystem of libraries, and a large pool of developers, **C++** remains a strong choice. However, be mindful of the manual memory management challenges.
- If memory safety and code reliability are paramount, and you're willing to invest in learning a newer language, **Rust** is an excellent option.
- If strong typing, real-time features, and code correctness are your primary concerns, **Ada** is a solid choice, but consider the trade-off in terms of wider adoption and developer availability.

Additional considerations

- Existing codebase, consider the cost of porting to a different language
- Development team expertise

Crystal ball

- C/C++ will stay, the legacy codebases are too big
- Ada probably will stay a niche market of bare metal, high integrity systems (defense, airplanes)
- Rust adaption is growing, but still far behind
- AI could be the deal breaker
  - Code conversion, move the legacy code base more effective to another language
  - Code generation
    - Still requires a good architecture and design
    - Need to be able to write the right requirements (pre & post conditions, invariants, formal or plain text)
    - Or just start with generating empty functions based on a specified interface, UML diagram, …

Code generation

Can you make a Histogram function on an image.

Pre condition I = Image (1 .. W, 1.. H)

Invariant P(k) = n(k)/N

Where I = 2 ^ q and q number of bits per pixel

n(k) is the number of pixels in image with intensity k

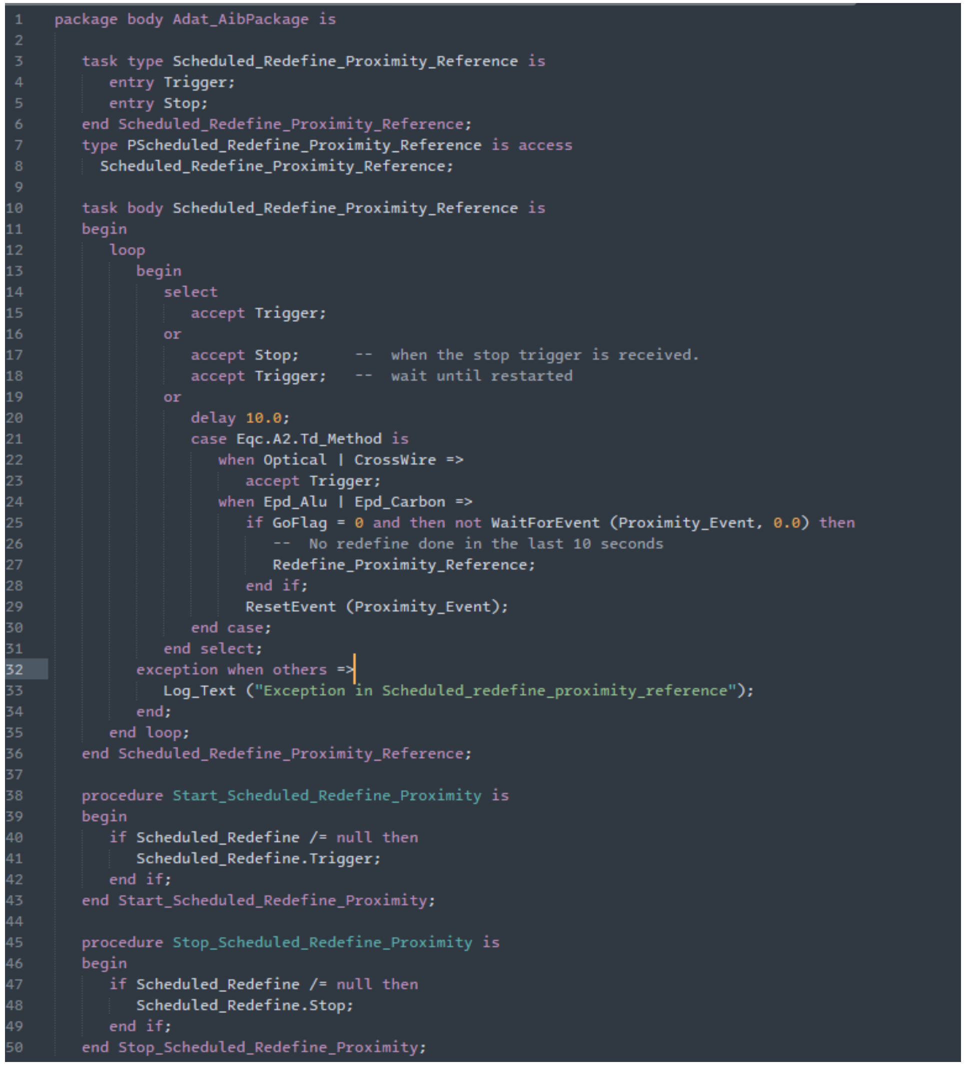
N = W * H, the total number of pixels in image

Post condition return P (0 .. 255)

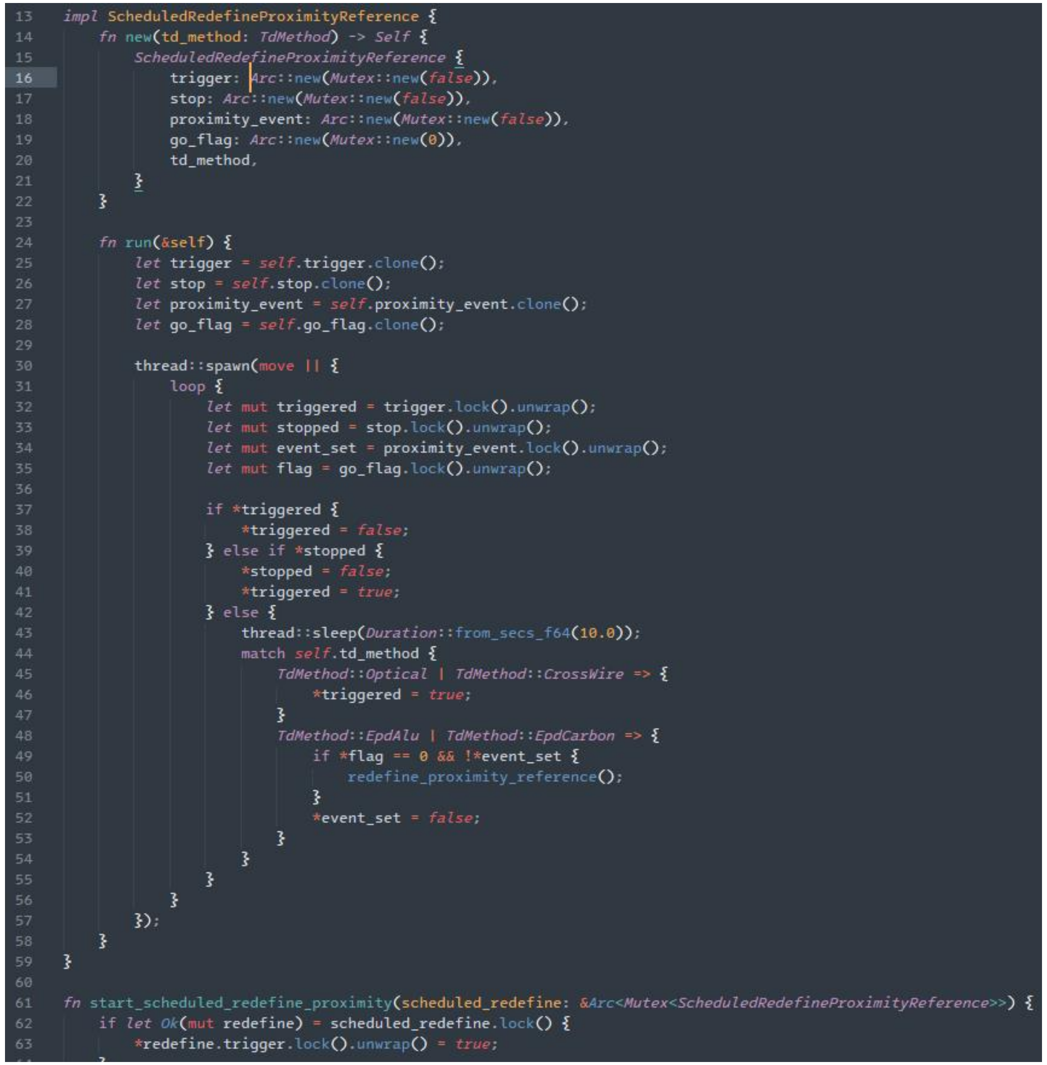
```
with Ada.Numerics.Discrete_Random;

function Histogram (I : Image) return Histogram_Array is
   W : constant Positive := I'Length (1);
   H : constant Positive := I'Length (2);
   N : constant Natural := W * H;
   P : Histogram_Array := (others => 0.0);
begin
   for X in I'Range (1) loop
      for Y in I'Range (2) loop
         declare
            Intensity : constant Natural := Natural (I (X, Y));
         begin
            P (Intensity) := P (Intensity) + 1.0 / Natural (N);
         end;
      end loop;
   end loop;

   return P;

end Histogram;
```

https://www.codeconvert.ai/free-code-generator

Code conversion Adat -> Rust



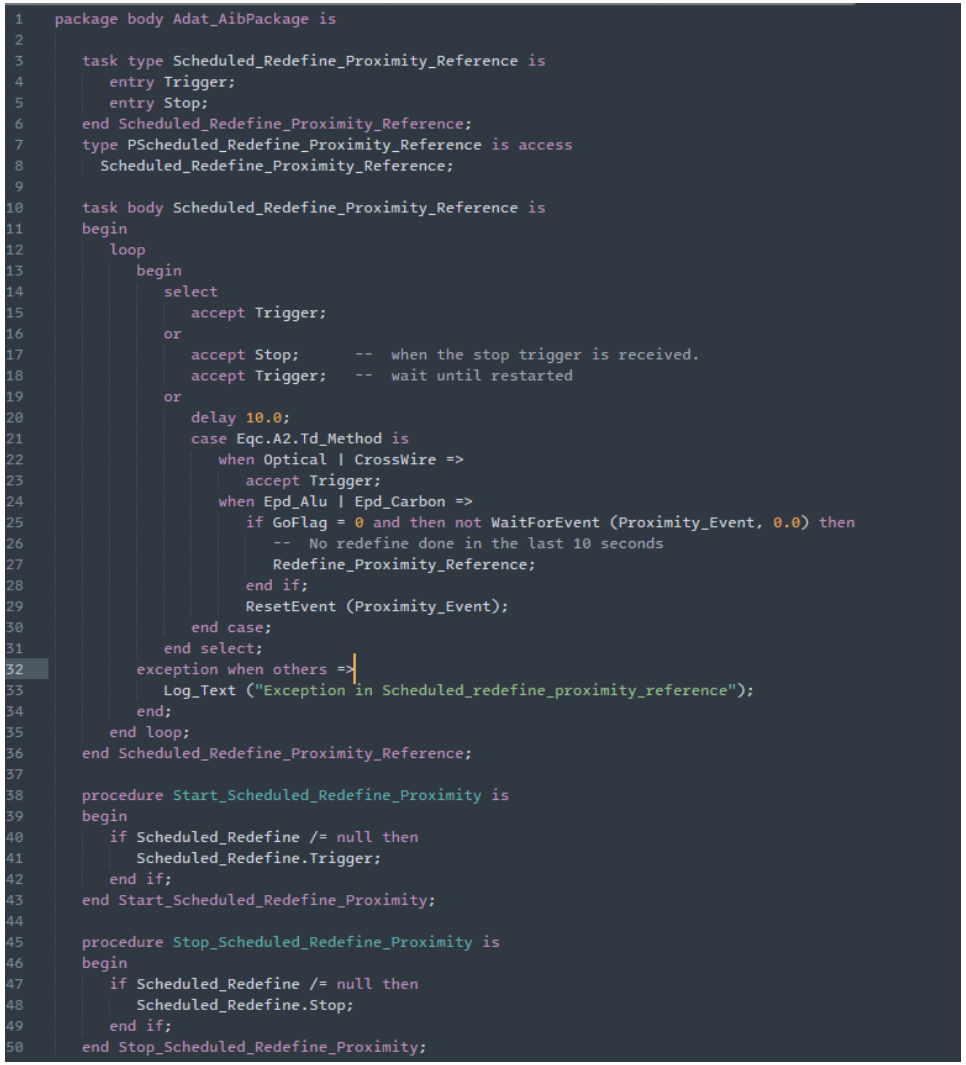
```
package body Adat_AibPackage is

   task type Scheduled_Redefine_Proximity_Reference is
      entry Trigger;
      entry Stop;
   end Scheduled_Redefine_Proximity_Reference;
   type PScheduled_Redefine_Proximity_Reference is access
     Scheduled_Redefine_Proximity_Reference;

   task body Scheduled_Redefine_Proximity_Reference is
   begin
      loop
         begin
            select
               accept Trigger;
            or
               accept Stop;      --  when the stop trigger is received.
               accept Trigger;   --  wait until restarted
            or
               delay 10.0;
               case Eqc.A2.Td_Method is
                  when Optical | CrossWire =>
                     accept Trigger;
                  when Epd_Alu | Epd_Carbon =>
                     if GoFlag = 0 and then not WaitForEvent (Proximity_Event, 0.0) then
                        --  No redefine done in the last 10 seconds
                        Redefine_Proximity_Reference;
                     end if;
                     ResetEvent (Proximity_Event);
               end case;
            end select;
         exception when others =>
            Log_Text ("Exception in Scheduled_redefine_proximity_reference");
         end;
      end loop;
   end Scheduled_Redefine_Proximity_Reference;

   procedure Start_Scheduled_Redefine_Proximity is
   begin
      if Scheduled_Redefine /= null then
         Scheduled_Redefine.Trigger;
      end if;
   end Start_Scheduled_Redefine_Proximity;

   procedure Stop_Scheduled_Redefine_Proximity is
   begin
      if Scheduled_Redefine /= null then
         Scheduled_Redefine.Stop;
      end if;
   end Stop_Scheduled_Redefine_Proximity;
```



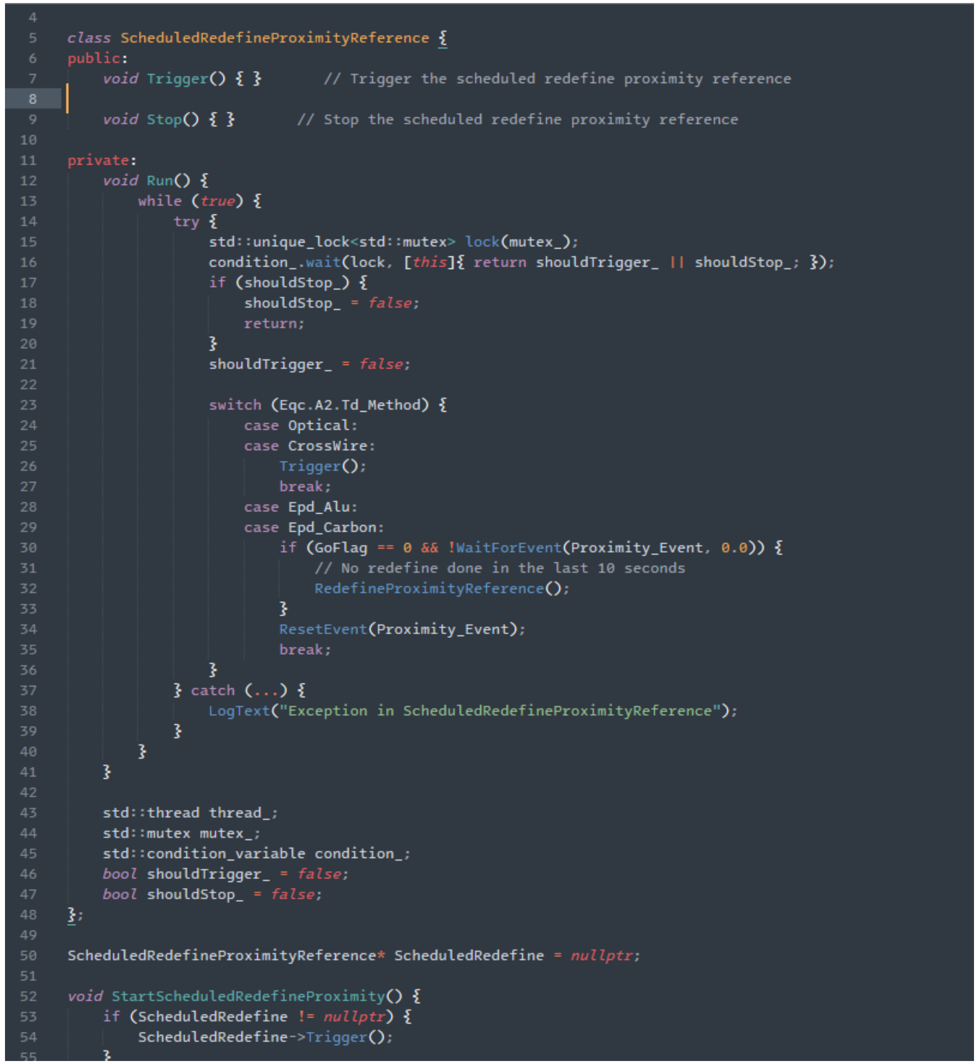
```
impl ScheduledRedefineProximityReference {
    fn new(td_method: TdMethod) -> Self {
        ScheduledRedefineProximityReference {
            trigger: Arc::new(Mutex::new(false)),
            stop: Arc::new(Mutex::new(false)),
            proximity_event: Arc::new(Mutex::new(false)),
            go_flag: Arc::new(Mutex::new(0)),
            td_method,
        }
    }

    fn run(&self) {
        let trigger = self.trigger.clone();
        let stop = self.stop.clone();
        let proximity_event = self.proximity_event.clone();
        let go_flag = self.go_flag.clone();

        thread::spawn(move || {
            loop {
                let mut triggered = trigger.lock().unwrap();
                let mut stopped = stop.lock().unwrap();
                let mut event_set = proximity_event.lock().unwrap();
                let mut flag = go_flag.lock().unwrap();

                if *triggered {
                    *triggered = false;
                } else if *stopped {
                    *stopped = false;
                    *triggered = true;
                } else {
                    thread::sleep(Duration::from_secs_f64(10.0));
                    match self.td_method {
                        TdMethod::Optical | TdMethod::CrossWire => {
                            *triggered = true;
                        }
                        TdMethod::EpdAlu | TdMethod::EpdCarbon => {
                            if *flag == 0 && !*event_set {
                                redefine_proximity_reference();
                            }
                            *event_set = false;
                        }
                    }
                }
            }
        });
    }
}

fn start_scheduled_redefine_proximity(scheduled_redefine: &Arc<Mutex<ScheduledRedefineProximityReference>>) {
    if let Ok(mut redefine) = scheduled_redefine.lock() {
        *redefine.trigger.lock().unwrap() = true;
```

https://www.codeconvert.ai/free-converter

Code conversion Adat -> C++



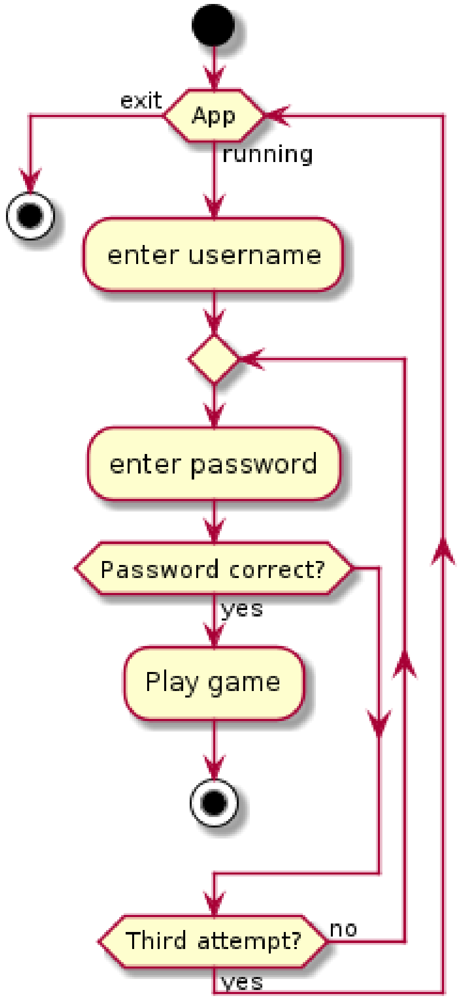
```
package body Adat_AibPackage is

   task type Scheduled_Redefine_Proximity_Reference is
      entry Trigger;
      entry Stop;
   end Scheduled_Redefine_Proximity_Reference;
   type PScheduled_Redefine_Proximity_Reference is access
     Scheduled_Redefine_Proximity_Reference;

   task body Scheduled_Redefine_Proximity_Reference is
   begin
      loop
         begin
            select
               accept Trigger;
            or
               accept Stop;      --  when the stop trigger is received.
               accept Trigger;   --  wait until restarted
            or
               delay 10.0;
               case Eqc.A2.Td_Method is
                  when Optical | CrossWire =>
                     accept Trigger;
                  when Epd_Alu | Epd_Carbon =>
                     if GoFlag = 0 and then not WaitForEvent (Proximity_Event, 0.0) then
                        --  No redefine done in the last 10 seconds
                        Redefine_Proximity_Reference;
                     end if;
                     ResetEvent (Proximity_Event);
               end case;
            end select;
         exception when others =>
            Log_Text ("Exception in Scheduled_redefine_proximity_reference");
         end;
      end loop;
   end Scheduled_Redefine_Proximity_Reference;

   procedure Start_Scheduled_Redefine_Proximity is
   begin
      if Scheduled_Redefine /= null then
         Scheduled_Redefine.Trigger;
      end if;
   end Start_Scheduled_Redefine_Proximity;

   procedure Stop_Scheduled_Redefine_Proximity is
   begin
      if Scheduled_Redefine /= null then
         Scheduled_Redefine.Stop;
      end if;
   end Stop_Scheduled_Redefine_Proximity;
```

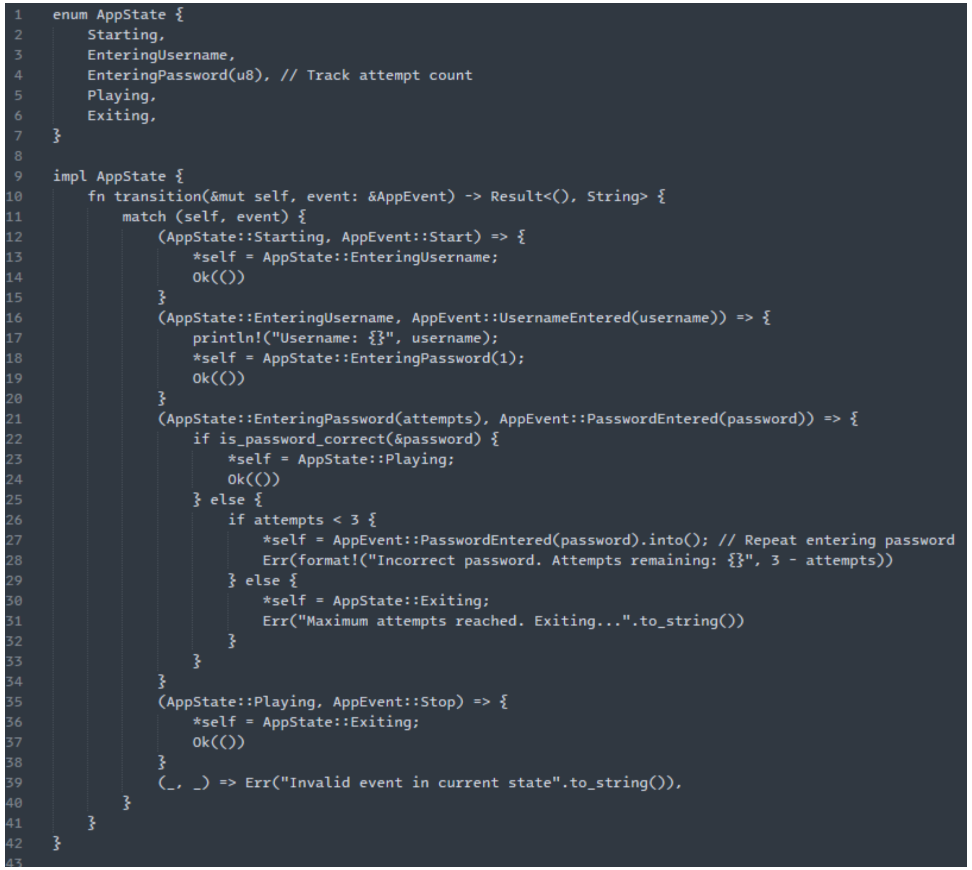


```

class ScheduledRedefineProximityReference {
public:
    void Trigger() { }       // Trigger the scheduled redefine proximity reference

    void Stop() { }       // Stop the scheduled redefine proximity reference

private:
    void Run() {
        while (true) {
            try {
                std::unique_lock<std::mutex> lock(mutex_);
                condition_.wait(lock, [this]{ return shouldTrigger_ || shouldStop_; });
                if (shouldStop_) {
                    shouldStop_ = false;
                    return;
                }
                shouldTrigger_ = false;

                switch (Eqc.A2.Td_Method) {
                    case Optical:
                    case CrossWire:
                        Trigger();
                        break;
                    case Epd_Alu:
                    case Epd_Carbon:
                        if (GoFlag == 0 && !WaitForEvent(Proximity_Event, 0.0)) {
                            // No redefine done in the last 10 seconds
                            RedefineProximityReference();
                        }
                        ResetEvent(Proximity_Event);
                        break;
                }
            } catch (...) {
                LogText("Exception in ScheduledRedefineProximityReference");
            }
        }
    }

    std::thread thread_;
    std::mutex mutex_;
    std::condition_variable condition_;
    bool shouldTrigger_ = false;
    bool shouldStop_ = false;
};

ScheduledRedefineProximityReference* ScheduledRedefine = nullptr;

void StartScheduledRedefineProximity() {
    if (ScheduledRedefine != nullptr) {
        ScheduledRedefine->Trigger();
    }
```

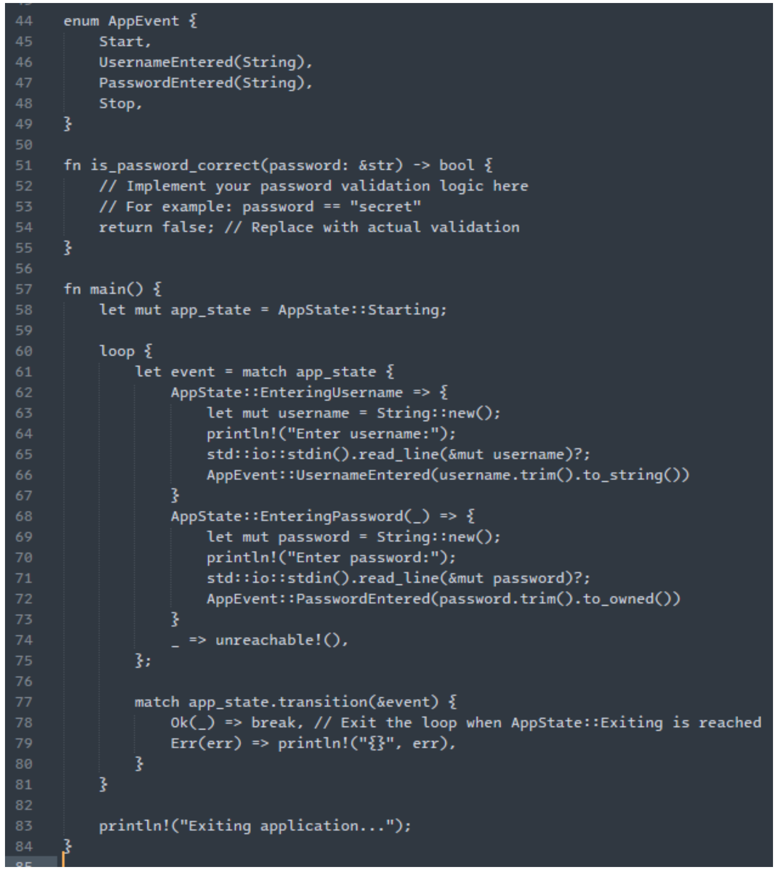
https://www.codeconvert.ai/free-converter

Generate a state machine (in Rust)





```rust
enum AppState {
    Starting,
    EnteringUsername,
    EnteringPassword(u8), // Track attempt count
    Playing,
    Exiting,
}

impl AppState {
    fn transition(&mut self, event: &AppEvent) -> Result<(), String> {
        match (self, event) {
            (AppState::Starting, AppEvent::Start) => {
                *self = AppState::EnteringUsername;
                Ok(())
            }
            (AppState::EnteringUsername, AppEvent::UsernameEntered(username)) => {
                println!("Username: {}", username);
                *self = AppState::EnteringPassword(1);
                Ok(())
            }
            (AppState::EnteringPassword(attempts), AppEvent::PasswordEntered(password)) => {
                if is_password_correct(&password) {
                    *self = AppState::Playing;
                    Ok(())
                } else {
                    if attempts < 3 {
                        *self = AppEvent::PasswordEntered(password).into(); // Repeat entering password
                        Err(format!("Incorrect password. Attempts remaining: {}", 3 - attempts))
                    } else {
                        *self = AppState::Exiting;
                        Err("Maximum attempts reached. Exiting...".to_string())
                    }
                }
            }
            (AppState::Playing, AppEvent::Stop) => {
                *self = AppState::Exiting;
                Ok(())
            }
            (_, _) => Err("Invalid event in current state".to_string()),
        }
    }
}
```



```rust
enum AppEvent {
    Start,
    UsernameEntered(String),
    PasswordEntered(String),
    Stop,
}

fn is_password_correct(password: &str) -> bool {
    // Implement your password validation logic here
    // For example: password == "secret"
    return false; // Replace with actual validation
}

fn main() {
    let mut app_state = AppState::Starting;

    loop {
        let event = match app_state {
            AppState::EnteringUsername => {
                let mut username = String::new();
                println!("Enter username:");
                std::io::stdin().read_line(&mut username)?;
                AppEvent::UsernameEntered(username.trim().to_string())
            }
            AppState::EnteringPassword(_) => {
                let mut password = String::new();
                println!("Enter password:");
                std::io::stdin().read_line(&mut password)?;
                AppEvent::PasswordEntered(password.trim().to_owned())
            }
            _ => unreachable!(),
        };

        match app_state.transition(&event) {
            Ok(_) => break, // Exit the loop when AppState::Exiting is reached
            Err(err) => println!("{}", err),
        }
    }

    println!("Exiting application...");
}
```

https://gemini.google.com/app

itec

Redefining
semiconductor
manufacturing