Intro to Ada

cguan@

Ada

- Strong, Static types
- Concurrent
- Imperative (+ Object-oriented)
- Expressive Contract-based programming
- Originally Designed for Embedded Systems by DoD

Hello, world

```
with Ada.Text_IO;

procedure Hello_World is
begin
    Ada.Text_IO.Put_Line("Hello, world!");
end Hello_World;
```

Basics (prime.adb)

```
with Ada.Text_IO;          use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
procedure Prime is -- Our main function is "Prime"
    Num : Integer; -- Declare local variables here
begin
    Put("Enter an integer: ");
    Get(Num);
    Put("The value "); Put(Num, 0);
    for idx in 2 .. (Num - 1) loop
        if Num rem idx = 0 then -- Assignment uses :=
            Put_Line(" is not prime.");
            return;
        end if;
    end loop;
    Put_Line(" is prime.");
end Prime;
```

```
> gnatmake prime.adb
gcc -c prime.adb
gnatbind -x prime.ali
gnatlink prime.ali
> ./prime
Enter an integer: 5
The value 5 is prime.
> ./prime
Enter an integer: 4
The value 4 is not prime.
```

Enumeration Types and Arrays (lunch.adb)

```
with Ada.Text_IO;          use Ada.Text_IO;
procedure Lunch is
   type Lunch_Spot_t is (WS, Nine, Home);
   type Day_t is (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
   subtype Weekday_t is Day_t range Mon .. Fri;
   Where_To_Eat : array (Day_t) of Lunch_Spot_t;
begin
   Where_To_Eat := (Home, Nine, WS, Nine, WS, Nine, Home);
   for Day in Weekday_t loop
      case Where_To_Eat (Day) is
         when Home =>
            Put_Line("Eating at home.");
         when Nine =>
            Put_Line("Eating on 9.");
         when WS =>
            Put_Line("Eating at Wise Sons");
      end case;
   end loop;
end Lunch;
```

```
> ./lunch
Eating on 9.
Eating at Wise Sons
Eating on 9.
Eating at Wise Sons
Eating on 9.
```

Why

- Safety matters, again
- Concurrency Support
- Modern software design principles
  - Expressive
  - Reusable Code
- Finally easy to get started with Ada

Bounds-Checking (bad_lunch.adb)

```
with Ada.Text_IO;          use Ada.Text_IO;
with Ada.Exceptions;       use Ada.Exceptions;
procedure Bad_Lunch is
   type Lunch_Spot_t is (WS, Nine, Home);
   type Day_t is (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
   subtype Weekday_t is Day_t range Mon .. Fri;
   Where_To_Eat : array(Weekday_t) of Lunch_Spot_t;
begin
   Where_To_Eat := (Nine, WS, Nine, WS, Nine);
   for Day in Day_t loop
      case Where_To_Eat (Day) is
         when Home =>
            Put_Line("Eating at home.");
         when Nine =>
            Put_Line("Eating on 9.");
         when WS =>
            Put_Line("Eating at Wise Sons");
      end case;
   end loop;
exception
   when Error : Constraint_Error =>
      Put_Line("It's the weekend, no lunch at Square.");
      Put_Line(Exception_Information(Error));

end Bad_Lunch;
```

```
> ./bad_lunch
It's the weekend, no lunch at Square.
raised CONSTRAINT_ERROR : bad_lunch.adb:17 index check
failed
```

C

1.

```c
uint32_t arr[4];
for (size_t i = 0; i <= sizeof(arr); i++) {
    arr[i] = 0;
}
```

2.

```c
struct type_t *s;
// (Allocate memory for s)
// zero-initialize the struct
memset(s, 0, sizeof(s));
```

3.

```c
unsigned int slew = 15000;
int error = -3300;
int neg_slew = -1 * slew;
int delta = SQ_CLAMP(error, slew, neg_slew)
// delta == 15000
```

Ada Records = Flexible C Structs

```
procedure RecordExample is
    type Date is
        record
            Day : Integer range 1 .. 31;
            Month : Integer range 1 .. 12;
            Year : Natural;
        end record;
    Today : Date := (Day => 7, Month => 6, Year => 2017);
    Tomorrow : Date; -- Uninitialized
begin
    Tomorrow := Today;
    Tomorrow.Day := Tomorrow.Day + 1;
end RecordExample;
```

Ada Records = Cleaner BSP

```
--  GPIO port bit set/reset register
type BSRR_Register is record
   BS : BSRR_BS_Field := (As_Array => False, Val => 16#0#);
   BR : BSRR_BR_Field := (As_Array => False, Val => 16#0#);
end record
  with Volatile_Full_Access, Size => 32,
       Bit_Order => System.Low_Order_First;

type BSRR_BS_Field --  GPIO port bit-set field
  (As_Array : Boolean := False)
is record
   case As_Array is
      when False =>
         Val : HAL.UInt16; --  BS as a value
      when True =>
         Arr : BSRR_BS_Field_Array; --  BS as an array
   end case;
end record
```

```
-- Set full mask
procedure Set (This : in out GPIO_Point) is
begin
   This.Periph.BSRR.BS.Val := GPIO_Pin'Enum_Rep (This.Pin);
end Set;

-- Set single bit
procedure Set (This : in out GPIO_Point) is
begin
   This.Periph.BSRR.BS.Arr (This.Pin) := True;
end Set;

-- From Ada Driver Library
```

Access Types and Records (access_types.adb)

```
with Ada.Unchecked_Deallocation;
procedure Access_Types is
    type Integer_Access is access all Integer;
    type Date_Access is access all Date;
    Int_P : Integer_Access; -- initializes to null
    -- P1 = P2 if they point to the same object
    D1, D2 : Date_Access;
    D3 : Date;

    -- `Unchecked_Deallocation` is a generic procedure
    procedure Delete_Integer is new
        Ada.Unchecked_Deallocation(Integer, Integer_Access);
    procedure Delete_Date is new
        Ada.Unchecked_Deallocation(Date, Date_Access);
begin
   -- Need to handle if `new` fails
    Int_P := new Integer'(0); -- Ada 83 to create Integer
access type, initialized to 0
    Int_P.all := 1; -- .all "dereferences" the pointer
    Delete_Integer(Int_P);
    -- For maximum portability, programmers must assume GC
is not done.

    D1 := new Date;
    D1.Day := 1;
    D2 := D1; -- D2 points to the same object as D1
    D3 := D2.all;
    Delete_Date(D1);
end Access_Types;
```

Preconditions and Postconditions (conditions.adb)

```
procedure Conditions is
   pragma Assertion_Policy(Check);

   function Increment (x : Integer) return Integer
     with Post => Increment'Result = (x + 1);
   procedure Integer_Division (Num :     Integer;
                               Den :     Integer;
                               Res : out Integer)
     with Pre => (Num rem Den) = 0;

   function Increment (x : Integer) return Integer is
   begin
      return (x + 1);
   end Increment;

   procedure Integer_Division (Num :     Integer;
                               Den :     Integer;
                               Res : out Integer) is
   begin
      Res := Num / Den;
   end Integer_Division;

      x : Integer := 10;
      y : Integer := 3;
      Res : Integer;
   begin
      Put ("The value of x is "); Put (x, 0); Put_Line (".");
      Put ("The value of Increment (x) is "); Put (Increment (x), 0);
   Put_Line (".");

      Put ("Trying to divide "); Put (x, 0); Put (" by "); Put (y,0);
   Put_Line (".");
      -- Precondition will fail for `Integer_Division`
      Integer_Division (x, y, Res);
      Put ("The result is "); Put (Res, 0); Put_Line (".");
   end Conditions;
```

```
> ./conditions
The value of x is 10.
The value of Increment (x) is 11.
Trying to divide 10 by 3.
raised SYSTEM.ASSERTIONS.ASSERT_FAILURE : failed precondition from
conditions.adb:16
```

Tasks (tasking.adb)

```
procedure Tasking is
    task HelloTask;
    task body HelloTask is
    begin
        for idx in 1 .. 5 loop
            Ada.Text_IO.Put_Line("The task says
hello.");
            delay 1.0;
        end loop;
    end HelloTask;

begin
    Ada.Text_IO.Put_Line("Starting Program!");
end Tasking;
```

```
> ./tasking
The task says hello.
Starting Program!
The task says hello.
The task says hello.
The task says hello.
The task says hello.
```

Threadsafe Objects (threadsafe_containers.ads/adb)

```
-- Specification (*.ads file)
package Threadsafe_Containers is
   type IntegerArray is array (Positive range<>) of Integer;
   protected type Threadsafe_Circbuf (Size : Positive) is
      entry Insert (Value : in Integer);
      entry Remove (Value : out Integer);
   private
      Buffer : IntegerArray(1 .. Size);
      Circbuf_Size : Positive := Size;
      Head : Positive := 1;
      Tail : Positive := 1;
      Length : Natural := 0;
   end Threadsafe_Circbuf;

end Threadsafe_Containers;
```

```
-- Implementation (*.adb file)
package body Threadsafe_Containers is
   protected body Threadsafe_Circbuf is
     entry Insert (Value : in Integer)
        when (Length < Size) is
        begin
            Buffer(Tail) := Value;
            Tail := (Tail mod Size) + 1;
            Length := Length + 1;
        end Insert;
    entry Remove (Value : out Integer)
        when (Length > 0) is
     begin
        Value := Buffer(Head);
        Head := (Head mod Size) + 1;
        Length := Length - 1;
     end Remove;
  end Threadsafe_Circbuf;
```

Using the IntegerArray (circbuf_example.adb)

```
with Threadsafe_Containers; use Threadsafe_Containers;
procedure Circbuf_Example is
   circbuf : Threadsafe_Circbuf (3);
   task body ProducerTask is
   begin
      for idx in 1..5 loop
         Ada.Text_IO.Put_Line("Producing!");
         circbuf.Insert(idx);
      end loop;
   end ProducerTask;
   task body ConsumerTask is
      Value : Integer;
   begin
      for idx in 1..5 loop
         Ada.Text_IO.Put_Line("Consuming!");
         circbuf.Remove(Value);
      end loop;
   end ConsumerTask;
```

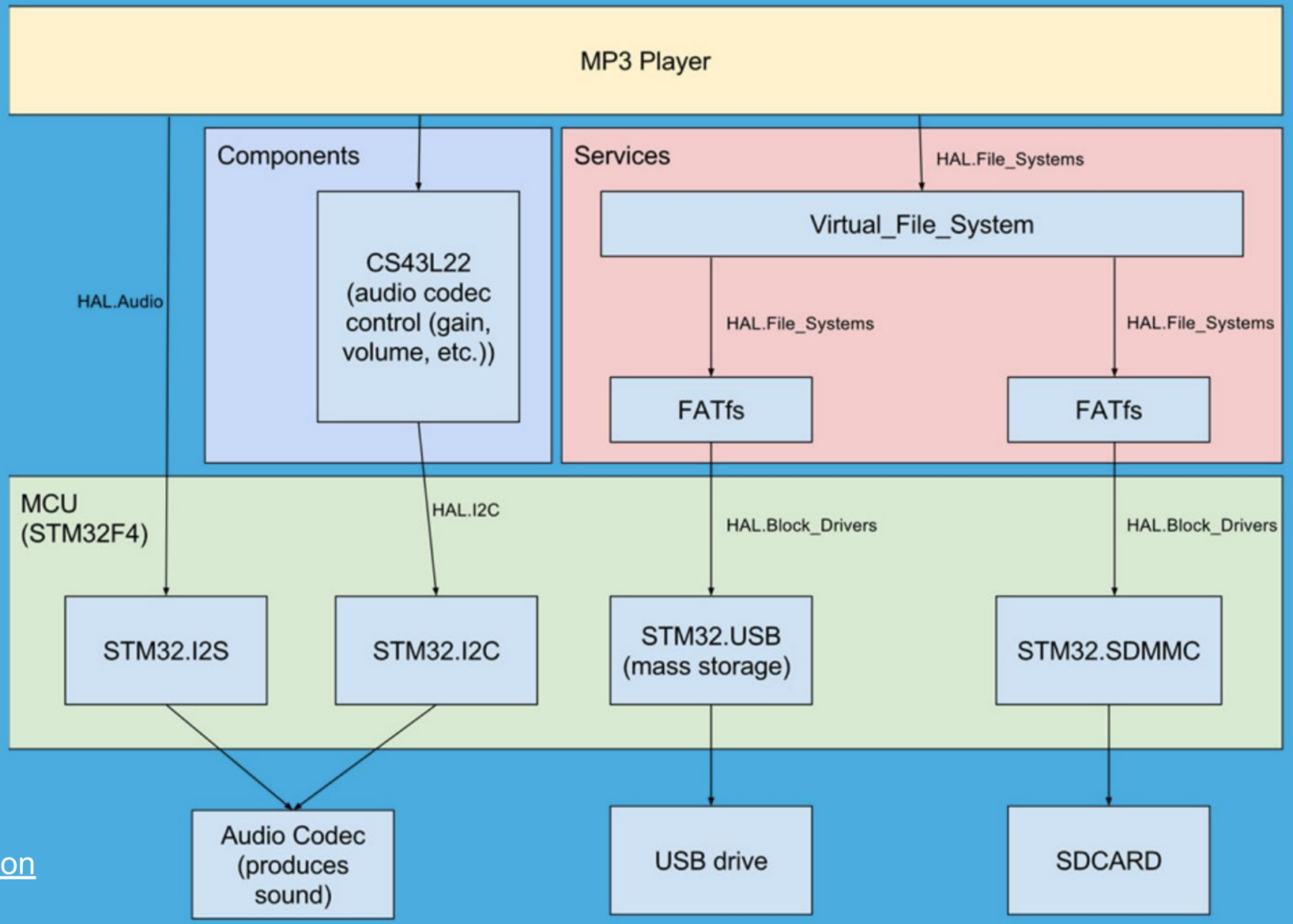
```
> ./circbuf_example
Producing!
Producing!
Producing!
Producing!
Consuming!
Consuming!
Consuming!
Consuming!
Consuming!
Producing!
```

Ada Driver Library

- GPIO
- I2C, SPI, UART
- ADC, DAC
- Audio
- LCD and touch screen drivers
- Timers, PWM
- DMA
- Flexible Memory Controller
- Camera
- Random Number Generator
- SDcard

(List copied from AdaCore Presentation)

Example: an MP3 Player



Copied from
AdaCore Presentation

Example Code Layers

- Blink an LED - Tasking
- HAL Specification File
- GPIO Driver
- SVD-Generated Register Specifications

Ada vs C

Ada Mindset

- Trust, but verify
- Programmers are human
- Depend on the compiler to produce efficient code

Ada Features

- Strong, static typing
- Almost no need for pointers
- Contract-based programming
- Object-oriented programming
- Portable concurrency support

C Mindset

- Trust the programmer
- What you see is what you get
- Make it fast, even if it won’t be portable

C Features

- Small language
- Flexible memory manipulation
- Function pointers
- Slightly more efficient
- Large user base

Thanks!

References and Resources

- Ada Programming Wikibook
- Ada on Cortex-M
- Ada Crash Course
- Ada Driver Library for ARM Cortex-M
- Comparing Ada and C
- Rust and Spark (Ada Subset)
- Make with Ada - Getting Started
- Ada Code Examples
- Ravenscar Profile Wikipedia
- Ada Posix Examples
- The Go Programming Language (for presentation format)

Questions?