OUTLINE

- Design languages
- Arx motivation and alternatives
- Main features of Arx
- Arx language elements
  - Components and functions
  - Data types
  - Statements
- Code generation and simulation

GENERAL PURPOSE VS. DOMAIN-SPECIFIC DESIGN LANGUAGES

- Should one adopt (and adapt) existing programming languages for the design of parallel embedded systems, signal processing systems?
- Yes, because:
  - This alleviates the burden of making new compilers, debuggers, etc.
- No, because:
  - One wants to model only the semantics of some domain and wants to keep the language clean of peculiarities of the host language.

ON LEARNING NEW LANGUAGES

- Reusing an existing language for specific modeling domains is not necessarily a good idea.
- What matters, is mastering the semantics of the domain.
- Learning to think in the paradigms of the domain takes much longer than learning a new programming language.
- It is e.g. a mistake to think that one convert a C programmer into a hardware designer by providing her with a tool that synthesizes hardware from C.

THE LANGUAGE SUBSET ISSUE

- When an existing language is used for describing models in a new language, one is confronted with the fact that not all language constructs make sense in the application domain.
- One necessarily needs to isolate a language subset that should be used.
- This is true for e.g. C.
- But also for e.g. VHDL, originally a simulation language, later used for synthesis.

DOMAIN-SPECIFIC LANGUAGES

- Languages specifically designed for well-defined, constrained, modeling are called domain-specific languages.
- No design mistakes due to subset violations: all language constructions are meaningful in domain.
- Tools such as parsers can be kept simple as they only need to deal with a small language rather than a large and complex one.

DATA-FLOW LANGUAGES

- They have the single-assignment property: a variable is only assigned a value once.
- This means that, after conversion into a DFG, the variable can be associated to the output of a single vertex.
- Because of single assignment, ordering of statements is not relevant.
- Think also of VHDL: a process should in principle write a signal only once (unless it contains wait statements).
- They can have syntactic support for typical data-flow elements such as the delay node.

DATA-FLOW LANGUAGE EXAMPLE: SILAGE

\[
\begin{align*}
\text{y} & = b_0 \times x + z_2 @ 1 \\
\text{z}_2 & = a_1 \times y + b_1 \times x + z_1 @ 1 \\
\text{z}_1 & = a_2 \times y + b_2 \times x
\end{align*}
\]

DATA FLOW IN C

- Voluntarily stick to single-assignment code.
- Use static variables for delay elements and read these values before writing them.

```c
T_out sec(T_in x) {
    static T_reg z1 = 0;
    static T_reg z2 = 0; ...
    y = b0*x + z2;
    z2_nxt = a1*y + b1*x + z1;
    z1_nxt = a2*y + b2*x;
    z2 = z2_nxt; z1 = z1_nxt; // register update
    return(y);
}
```

Can this go wrong?

PRACTICE IN SYSTEM-LEVEL-TO-RTL TRANSITION

System models often described in C or Matlab

Costly manual translation (months)

Hardware models in VHDL

C-BASED HARDWARE DESIGN

- Arguments in favor of C-based design:
  - Everybody knows C; we don't want to teach new languages.
  - Lots of legacy C code.
  - High execution speed.
- Many commercial products based on translation from C/C++/SystemC including:
  - Catapult (Calypso part of Mentor Graphics)
  - Stratus (Cadence, replaces C-to-Silicon and Synthesizer)
  - Synphony C Compiler (Synopsys, formerly Synfora PICO)
  - Vivado (Xilinx, formerly AutoESL)
  - Intel HLS Compiler (Intel/Altera, front-end to Platform Designer/Quartus)
  - CyberWorkBench (NEC System Technologies)

GRAPHICAL DESIGN ENTRY

- Many solutions based on dedicated blocksets to be used in Simulink:
  - Mathwork's HDL Coder (from graphics and text source)
  - Synphony Model Compiler (Synopsys)
  - Xilinx System Generator for DSP
  - Intel/Altera DSP Builder
- Graphical design entry can be cumbersome compared to text-based entry:
  - One does not always want to instantiate an adder for every addition, a multiplexer for every if-statement, etc.
**DOMAIN-SPECIFIC DESIGN LANGUAGES**

- All language constructs make sense in domain:
  - Entire language is synthesizable.
  - Designer does not need to bother about allowed subsets.

- Straightforward language constructions:
  - Improve designer efficiency.
  - Lead to elegant designs.

- Examples:
  - Bluespec (commercial)
  - GEZEL (university tool)

**ARX: A DOMAIN-SPECIFIC RTL LANGUAGE**

- One language (Arx) for multiple levels.
- Developed at University of Twente.
- **Arx eliminates manual translation from C to VHDL!**
- Correct by construction.

**LANGAUGE FEATURES**

- Explicit distinction between wires and registers.
- Implicit clock and reset.
- Generic data types allowing propagation of data types down hierarchy (e.g. floating-point to fixed-point refinement).
- Data types for DSP, especially fixed-point data types.
  - Support for overflow and quantization modes.
  - Efficient simulation of fixed-point data types.
- **No semicolons!**
- Simple: can be learned in one day!
ON-LINE FEATURES

- Please visit: www.bibix.nl

- The website gives access to:
  - On-line wiki-style manual,
  - Web-based demonstration (upload Arx, download corresponding C++ and VHDL),
  - An IP library of basic blocks: FIR filter, CORDIC, FFT, etc.
  - A GFSK receiver.

- Feedback on Arx, requests for cooperation, very welcome.

THE ARX LANGUAGE: BUILDING BLOCKS

- Components
  - Same as entities (VHDL), modules (Verilog/SystemC)
  - Contain sequential logic
  - Can be instantiated inside other components (hierarchical descriptions are allowed)
  - **In current version:** entire design in one file.

- Functions
  - Contain only combinational logic
  - **In current version:** not supported in VHDL generation (you need to write the VHDL function by hand)

EXAMPLE: COMPONENT INSTANTIATION

```vhd
# subcomponent
component reg
  generic integer = 8
  T_I0 : generic type = bitvector(word_length)
  data_in : in T_I0
  data_out : out T_I0
begin
  storage : T_I0 = 0
  assignment = data_in
  data_out = storage
end
```

ARX DATA OBJECTS

- Registers:
  - They store data are updated at the end of clock cycle.
  - Assignment is concurrent.

- Variables:
  - Correspond to wires.
  - Assignment is sequential (“single assignment” not required).
DATA TYPES

• Scalar types:
  – bit
  – boolean
  – integer
  – real
• Enumerated types (e.g. for state specification)
• Vector types:
  – bitvector
  – signed
  – unsigned

FIXED-POINT DATA TYPES

• Refinement of signed/unsigned:
• By supplying additional optional arguments for:
  – Integer word length
  – Overflow mode
  – Quantization mode
• Examples:
  – signed(8)
  – unsigned(8, 3): fixed-point with 5 fractional bits, wrap-around for overflow, truncate for quantization
  – unsigned(8, 3, saturate, round)

FIXED-POINT SUPPORT

• Use of fixed-point data type implies automatic code generation for:
  – Binary-point alignment
  – Sign extension
  – Handling of overflow and quantization mode.

EXAMPLE: USE OF CONSTANTS

```plaintext
register
# three registers initialized with the same value
bval1: bitvector(8) = 0b10101010
bval2: bitvector(8) = 0b111
bval3: bitvector(8) = 170
# more examples of constants
bval4: unsigned(8) = 0b11111111
bval5: unsigned(8,2) = 1.75 # no loss of precision
bval6: signed(8,2) = -1.5 # no loss of precision
bval7: signed(8,4) = 3.14 # will be converted to 3.125 = 50/16
```
EXAMPLE: ENUMERATION DATA TYPE

```vhdl
type
  input_state = enum(start, processing, ready)

# a registered signal of type input_state with its reset value
register
  current_state: input_state = input_state.start

# later on in the code
begin
  if current_state = input_state.start
  begin
    if start_of_processing
    output_state = output_state.processing
  end

  when output_state, processing
  if end_of_processing
  output_state = output_state.ready
end
```

EXAMPLE: ARRAYS

```vhdl
component top
  T_IO : generic type = signed(10, 5, sat, round)
data_in : in T_IO
data_out : out T_IO
type
  T_enum: enum(one, two, three)
  T_arr1: array[3] of T_IO
t_arr2: array[3] of T_enum
end component
class register
  v1 : T_arr1 = 0
  v2 : T_arr2 = [T_enum.three, T_enum.two, T_enum.one]
end class
class begin
  v1[1] = data_in
  for i in 0..1
    v2[i] = v2[i+1]
end
```

EXAMPLE: CASE STATEMENT

```vhdl
case output_state
  when output_state, start
  if start_of_processing
  output_state = output_state.processing
end

  when output_state, processing
  if end_of_processing
  output_state = output_state.ready
end
else # default case; no action
end
```

FOR STATEMENT

- Iteration based on an index variable
  - Index can only be incremented by 1
- Specifies iteration in space not in time (as in e.g. VHDL).
- Example:

  ```vhdl
  for i in 1:half_size
  delay_group[i] = delay_line[half_size-i] + delay_line[half_size+i]
end
  ```
CODE GENERATION

- Based on data-flow analysis & static scheduling.
- C++-code generation (targeted for fast simulation):
  - Flattens description
  - Maps fixed-point data types on integers (limited to 64 bits)
  - C++ object with:
    - reset method
    - run method to simulate one clock cycle
  - Optional VCD generation for waveform viewing (now: all or none)
- VHDL-code generation (targeted for synthesis):
  - Preserves component hierarchy

SUMMARY

- A domain-specific language for the RTL MoC, e.g. Arx, bridges wall when descending from the system level.
- Arx brings about that one source code generates:
  - C++-based simulation model optimized for simulation speed
  - VHDL code for synthesis.
- The Arx approach:
  - Saves manual recoding time!
  - Is correct by construction!