RTL DESIGN WITH ARX

IMPLEMENTATION OF DIGITAL SIGNAL PROCESSING

Sabih H. Gerez
University of Twente

OUTLINE

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 (software) language for a specific modeling domain (hardware) 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.

Edwards, S.A., *The Challenges of Synthesizing Hardware from C-Like Languages*, IEEE Design and Test of Computers, pp. 375-385, (September/October 2006).

THE LANGUAGE SUBSET ISSUE

- When an existing language is used for describing models in a new domain, 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 VHDL, originally a simulation language, later used for synthesis.
- And also for Matlab for the purpose of HDL Coder.

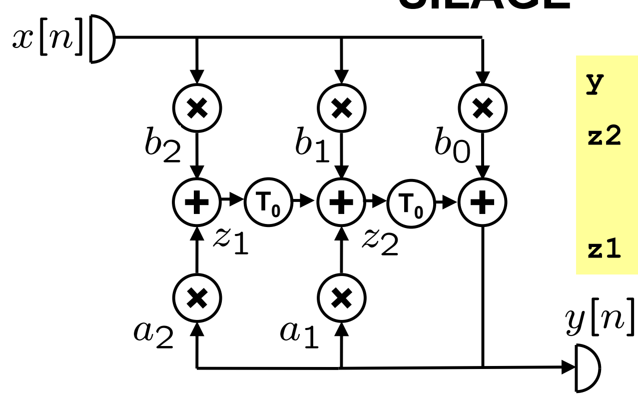
DOMAIN-SPECIFIC LANGUAGES

- 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

- Idea: specify data-flow graphs using text.
- Example feature: 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




```
y  = b0*x + z2@1
z2 = a1*y + b1*x +
     z1@1
z1 = a2*y + b2*x
```

Hilfinger, P.N., "A High-Level Language and Silicon Compiler for Digital Signal Processing", *Custom Integrated Circuit Conference*, pp. 213-216, (1985).

RTL IN C

- Voluntarily stick to single-assignment code.
- Use *static* variables for registers and read these values before writing them (initial value = reset value).

```
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);
}
```

One function call advances one clock cycle.

Can this go wrong?

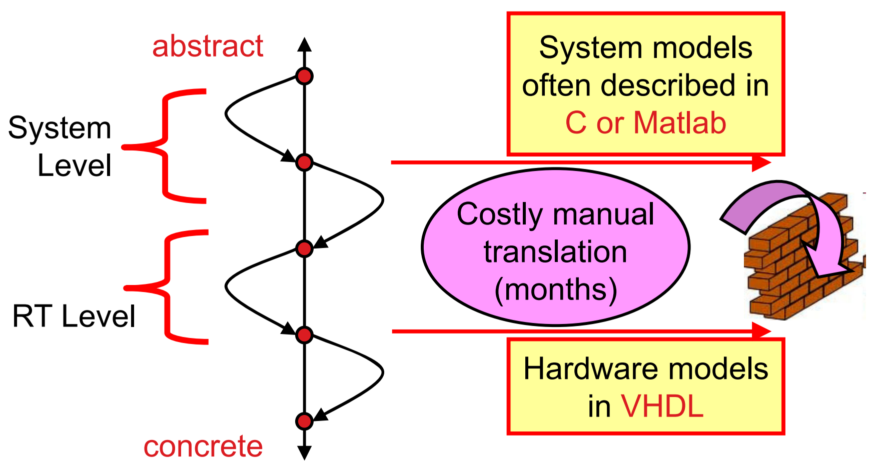
RTL IN MATLAB

```
function y = sec(x)
  persistent z1;
  persistent z2;
  if is_empty(z1)
     z1 = fi(0, T_reg);
     z2 = fi(0, T_reg);
  end

  y = fi(b0*x + z2, T_out);
  z1_nxt = fi(a1*y + b1*x + z1, T_reg);
  z2_nxt = fi(a2*y + b2*x, T_reg);

  z1(:) = z1_nxt;
  z2(:) = z2_nxt;
end
```

Equivalent of *static* in C

Check for “not initialized” needs to be used to specify reset values!

PRACTICE IN SYSTEM-LEVEL-TO-RTL TRANSITION



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 (Siemens)
  - Stratus (Cadence)
  - Vivado (Xilinx/AMD)
  - Intel HLS Compiler (Intel/Altera, front-end to Platform Designer/Quartus)
  - CyberWorkBench (NEC System Technologies)
- See for more: https://en.wikipedia.org/wiki/High-level_synthesis

GRAPHICAL DESIGN ENTRY

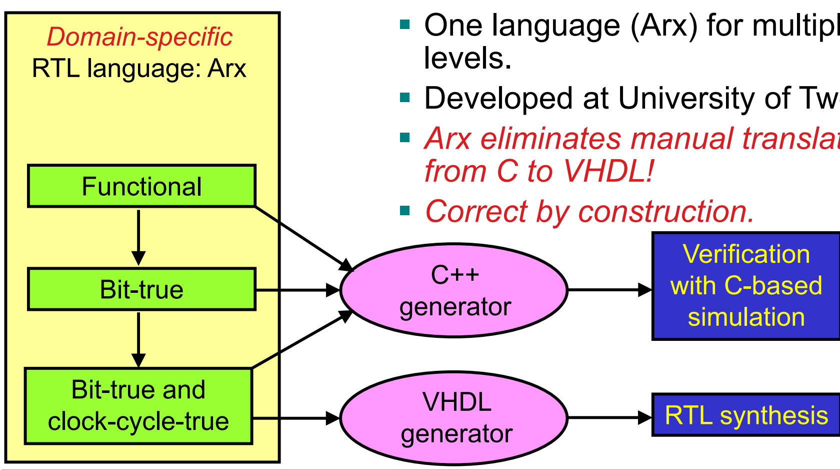
- Many solutions based on dedicated *blocksets* to be used in Simulink:
  - Mathwork's HDL Coder (from graphics and text source)
  - Synphony Model Compiler (Microsemi)
  - Xilinx System Generator for DSP
  - Intel DSP Builder
- Graphical design entry can be cumbersome compared to text-based entry:
  - 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:
  - GEZEL (university tool, https://sourceforge.net/projects/gezel/)

Schaumont, P., D. Ching and I. Verbauwhede, *An Interactive Codesign Environment for Domain-Specific Coprocessors*, ACM Transactions on Design Automation of Electronic Systems, Vol.11(1), pp. 70-87, (January 2006).

ARX: A DOMAIN-SPECIFIC RTL LANGUAGE

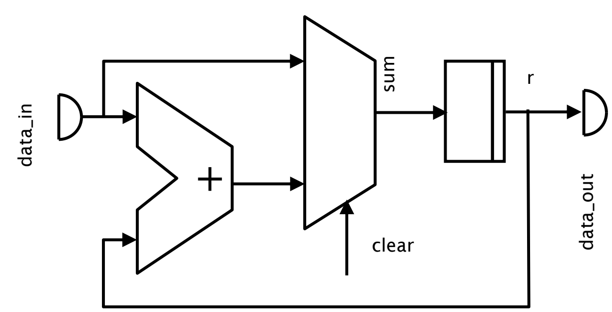


- One language (Arx) for multiple levels.
- Developed at University of Twente.
- *Arx eliminates manual translation from C to VHDL!*
- *Correct by construction.*

ARX EXAMPLE

```
component accumulator
  wl: generic integer = 10
  T_in : generic type = signed(wl  , 1)
  T_out: generic type = signed(wl-2, 1, sat, round)
  T_sum: generic type = signed(wl+5, 6)
  clear: in bit
  data_in : in  T_in
  data_out: out T_out
variable
  sum: T_sum
register
  r: T_sum = 0
begin
  if clear == 1
     sum = data_in
  else
     sum = r + data_in
  end
  r = sum
  data_out = r
end
```

Clock and *reset* are implicit.



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!

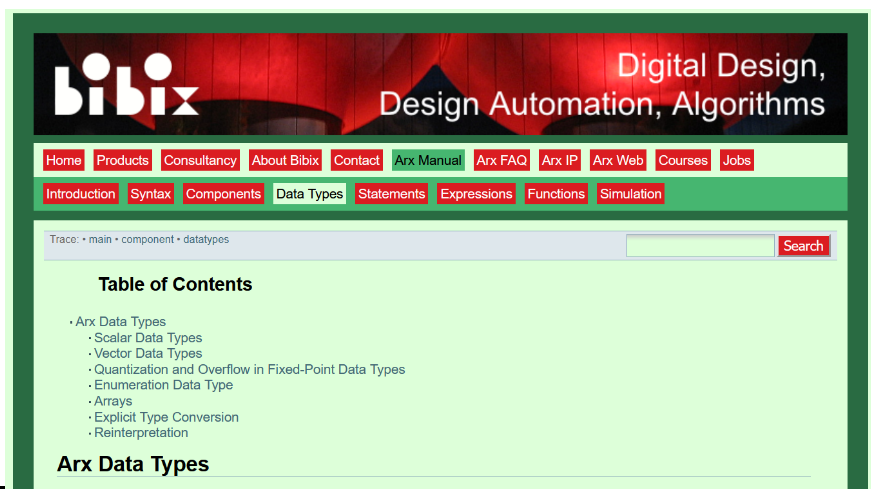
ON-LINE FEATURES

- 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.

IMPRESSION ARX MANUAL



PUBLICATIONS

- Hofstra, K.L. and S.H. Gerez, *"Arx: A Toolset for the Efficient Simulation and Direct Synthesis of High-Performance Signal Processing Algorithms"*, International Conference on High Performance Embedded Architectures and Compilers, Ghent, Belgium, (January 2007).
- Hofstra, K.L., S.H. Gerez and D. van Kampen, *"A Language and Toolset for the Synthesis and Efficient Simulation of Clock-Cycle-True Signal-Processing Algorithms"*, 16th Annual Workshop on Circuits, Systems and Signal Processing, ProRISC 2005, Veldhoven, The Netherlands, (November 2005).
- Kampen, D. van, K.L. Hofstra, J. Potman and S.H. Gerez, *"Implementation of a Combined OFDM-Demodulation and WCDMA-Equalization Module"*, Annual Workshop on Circuits, Systems and Signal Processing, ProRISC 2006, Veldhoven, The Netherlands, (November 2006).

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 INSTATIATION

```
# subcomponent
component reg
  word_length: generic integer = 8
  T_IO       : generic type = bitvector(word_length)
  data_in    : in T_IO
  data_out   : out T_IO

register
  storage   : T_IO = 0

begin
  storage = data_in
  data_out = storage
end
```

```
component top
  word_length: generic integer = 12
  T_topIO    : generic type = bitvector(word_length)
  data_in    : in T_topIO
  data_out   : out T_topIO

variable
  data_internal: T_topIO

generate
  r1: reg
    T_IO = T_topIO
    data_in  => data_in
    data_out => data_internal

  r2: reg
    word_length = word_length
    data_in  => data_internal
    data_out => data_out

begin
  # no functionality at this level
end
```

Restriction: the top-level component in Arx needs to be called `top`.

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

```
register
  # three registers initialized with the same value
  bval1: bitvector(8) = 0b10101010
  bval2: bitvector(8) = 0haa
  bval3: bitvector(8) = 170

  # more examples of constants
  bval4: unsigned(8)    = 0haa
  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

```
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
     current_state = input_state.processing
  end
```

EXAMPLE: ARRAYS

```
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_ar1: array[3] of T_IO
  T_ar2: array[3] of T_enum

register
  v1 : T_ar1 = 0
  v2 : T_ar2 = {T_enum.three, T_enum.two, T_enum.one}
  v3 : array[5] of T_IO = {5, 4, 3, 2, 1}

begin
  v1[1] = data_in
  for i in 0:1
    v2[i] = v2[i+1]
  end
  # example of accessing individual bits in an array of vectors
  v3[0][0:4] = v1[2][5:9]
  v3[0][5:9] = v1[2][0:4]
```


EXAMPLE: CASE STATEMENT

```
case output_state
  when out_state.start
    if start_of_processing
      output_state = out_state.processing
    end
  when out_state.processing
    if end_of_processing
      output_state = out_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:

```
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

C++ TESTBENCH IN IT++

- The C++ generated by Arx will need a testbench to be executed with.
- Any C++ code could be used.
- In current projects, the testbench makes use of IT++:

  https://itpp.sourceforge.net/4.3.1/

- IT++ provides Matlab-style data structures (vectors & matrices) and links with powerful math libraries to deliver efficient execution speeds.

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!