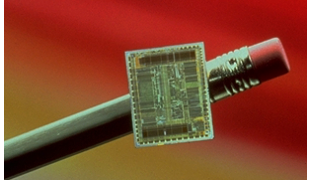
TELECOM Paris

IP PARIS



SE303/ICS901 : Soc Design

HLS : High Level Synthesis

Yves MATHIEU
yves.mathieu@telecom-paristech.fr

Institut Mines-Télécom

Outline

Outline

The designer dream...

From RTL synthesis to HLS synthesis

- Avoiding long and fastidious cycle accurate, register accurate and bit accurate coding
- We would like to describe :
  - An algorithm (signal processing, data processing...)
  - A target technology (ASIC xxx, FPGA yyy)
  - Speed constraints (target clock frequency)
  - Throughput constraints (number invocations per second of the algorithm)
  - Latency constraints (maximum allowed delay "cycles" between the invocation and the result
- Solution : High Level Synthesis

The designer dream...

From RTL synthesis to HLS synthesis

- Avoiding long and fastidious cycle accurate, register accurate and bit accurate coding
- We would like to describe :
  - An algorithm (signal processing, data processing...)
  - A target technology (ASIC xxx, FPGA yyy)
  - Speed constraints (target clock frequency)
  - Throughput constraints (number invocations per second of the algorithm)
  - Latency constraints (maximum allowed delay "cycles" between the invocation and the result
- Solution : High Level Synthesis

The designer dream...

From RTL synthesis to HLS synthesis

- Avoiding long and fastidious cycle accurate, register accurate and bit accurate coding
- We would like to describe :
  - An algorithm (signal processing, data processing...)
  - A target technology (ASIC xxx, FPGA yyy)
  - Speed constraints (target clock frequency)
  - Throughput constraints (number invocations per second of the algorithm)
  - Latency constraints (maximum allowed delay "cycles" between the invocation and the result
- Solution : High Level Synthesis

The designer dream...

From RTL synthesis to HLS synthesis

- Avoiding long and fastidious cycle accurate, register accurate and bit accurate coding
- We would like to describe :
  - An algorithm (signal processing, data processing...)
  - A target technology (ASIC xxx, FPGA yyy)
  - Speed constraints (target clock frequency)
  - Throughput constraints (number invocations per second of the algorithm)
  - Latency constraints (maximum allowed delay "cycles" between the invocation and the result
- Solution : High Level Synthesis

The designer dream...

From RTL synthesis to HLS synthesis

- Avoiding long and fastidious cycle accurate, register accurate and bit accurate coding
- We would like to describe :
  - An algorithm (signal processing, data processing...)
  - A target technology (ASIC xxx, FPGA yyy)
  - Speed constraints (target clock frequency)
  - Throughput constraints (number invocations per second of the algorithm)
  - Latency constraints (maximum allowed delay "cycles" between the invocation and the result
- Solution : High Level Synthesis

The designer dream...

From RTL synthesis to HLS synthesis

- Avoiding long and fastidious cycle accurate, register accurate and bit accurate coding
- We would like to describe :
  - An algorithm (signal processing, data processing...)
  - A target technology (ASIC xxx, FPGA yyy)
  - Speed constraints (target clock frequency)
  - Throughput constraints (number invocations per second of the algorithm)
  - Latency constraints (maximum allowed delay "cycles" between the invocation and the result
- Solution : High Level Synthesis

The designer dream...

From RTL synthesis to HLS synthesis

- Avoiding long and fastidious cycle accurate, register accurate and bit accurate coding
- We would like to describe :
  - An algorithm (signal processing, data processing...)
  - A target technology (ASIC xxx, FPGA yyy)
  - Speed constraints (target clock frequency)
  - Throughput constraints (number invocations per second of the algorithm)
  - Latency constraints (maximum allowed delay "cycles" between the invocation and the result
- Solution : High Level Synthesis

The designer dream...

From RTL synthesis to HLS synthesis

- Avoiding long and fastidious cycle accurate, register accurate and bit accurate coding
- We would like to describe :
  - An algorithm (signal processing, data processing...)
  - A target technology (ASIC xxx, FPGA yyy)
  - Speed constraints (target clock frequency)
  - Throughput constraints (number invocations per second of the algorithm)
  - Latency constraints (maximum allowed delay "cycles" between the invocation and the result
- Solution : High Level Synthesis

HLS Methodologies and tools for Systems on Chips

A plethora of situations

- Choice of input language (C, C++, SystemC)
- Culture of the destination audience :
  - Computer scientists : weak culture of ad-hoc digital hardware architectures..
  - Electronic specialists : weak culture of high level languages.
- Each HLS tool has an implicit targeted architecture :
  - CPU style : based on a control flow graph and using predefined resources (register bank, multipliers, ALUs...) : Hardly used anymore
  - DATA-FLOW style : based on a data flow graph and state machines : current trend for most tools.

HLS Methodologies and tools for Systems on Chips

A plethora of situations

- Choice of input language (C, C++, SystemC)
- Culture of the destination audience :
  - Computer scientists : weak culture of ad-hoc digital hardware architectures..
  - Electronic specialists : weak culture of high level languages.
- Each HLS tool has an implicit targeted architecture :
  - CPU style : based on a control flow graph and using predefined resources (register bank, multipliers, ALUs...) : Hardly used anymore
  - DATA-FLOW style : based on a data flow graph and state machines : current trend for most tools.

HLS Methodologies and tools for Systems on Chips

A plethora of situations

- Choice of input language (C, C++, SystemC)
- Culture of the destination audience :
  - Computer scientists : weak culture of ad-hoc digital hardware architectures..
  - Electronic specialists : weak culture of high level languages.
- Each HLS tool has an implicit targeted architecture :
  - CPU style : based on a control flow graph and using predefined resources (register bank, multipliers, ALUs...) : Hardly used anymore
  - DATA-FLOW style : based on a data flow graph and state machines : current trend for most tools.

HLS Methodologies and tools for Systems on Chips

A plethora of situations

- Choice of input language (C, C++, SystemC)
- Culture of the destination audience :
  - Computer scientists : weak culture of ad-hoc digital hardware architectures..
  - Electronic specialists : weak culture of high level languages.
- Each HLS tool has an implicit targeted architecture :
  - CPU style : based on a control flow graph and using predefined resources (register bank, multipliers, ALUs...) : Hardly used anymore
  - DATA-FLOW style : based on a data flow graph and state machines : current trend for most tools.

HLS Methodologies and tools for Systems on Chips

A plethora of situations

- Choice of input language (C, C++, SystemC)
- Culture of the destination audience :
  - Computer scientists : weak culture of ad-hoc digital hardware architectures..
  - Electronic specialists : weak culture of high level languages.
- Each HLS tool has an implicit targeted architecture :
  - CPU style : based on a control flow graph and using predefined resources (register bank, multipliers, ALUs...) : Hardly used anymore
  - DATA-FLOW style : based on a data flow graph and state machines : current trend for most tools.

HLS Methodologies and tools for Systems on Chips

A plethora of situations

- Choice of input language (C, C++, SystemC)
- Culture of the destination audience :
  - Computer scientists : weak culture of ad-hoc digital hardware architectures..
  - Electronic specialists : weak culture of high level languages.
- Each HLS tool has an implicit targeted architecture :
  - CPU style : based on a control flow graph and using predefined resources (register bank, multipliers, ALUs...) : Hardly used anymore
  - DATA-FLOW style : based on a data flow graph and state machines : current trend for most tools.

HLS Methodologies and tools for Systems on Chips

A plethora of situations

- Choice of input language (C, C++, SystemC)
- Culture of the destination audience :
  - Computer scientists : weak culture of ad-hoc digital hardware architectures..
  - Electronic specialists : weak culture of high level languages.
- Each HLS tool has an implicit targeted architecture :
  - CPU style : based on a control flow graph and using predefined resources (register bank, multipliers, ALUs...) : Hardly used anymore
  - DATA-FLOW style : based on a data flow graph and state machines : current trend for most tools.

HLS Methodologies and tools for Systems on Chips

Languages

- No precise definition of the concept of high-level representation of a hardware system.
- No common definition of the usable subset of a given language ?
- How to model communication buses and protocols ?
- Systematic use of synthesis "pragmas" to express the intention of the designer
  - Guide the tool towards a reasonable solution (parallelism / pipeline)
  - Choosing technical options for a given technology (example RAM / flip-flops)

HLS Methodologies and tools for Systems on Chips

Languages

- No precise definition of the concept of high-level representation of a hardware system.
- No common definition of the usable subset of a given language ?
- How to model communication buses and protocols ?
- Systematic use of synthesis "pragmas" to express the intention of the designer
  - Guide the tool towards a reasonable solution (parallelism / pipeline)
  - Choosing technical options for a given technology (example RAM / flip-flops)

HLS Methodologies and tools for Systems on Chips

Languages

- No precise definition of the concept of high-level representation of a hardware system.
- No common definition of the usable subset of a given language ?
- How to model communication buses and protocols ?
- Systematic use of synthesis "pragmas" to express the intention of the designer
  - Guide the tool towards a reasonable solution (parallelism / pipeline)
  - Choosing technical options for a given technology (example RAM / flip-flops)

HLS Methodologies and tools for Systems on Chips

Languages

- No precise definition of the concept of high-level representation of a hardware system.
- No common definition of the usable subset of a given language ?
- How to model communication buses and protocols ?
- Systematic use of synthesis "pragmas" to express the intention of the designer
  - Guide the tool towards a reasonable solution (parallelism / pipeline)
  - Choosing technical options for a given technology (example RAM / flip-flops)

HLS Methodologies and tools for Systems on Chips

Languages

- No precise definition of the concept of high-level representation of a hardware system.
- No common definition of the usable subset of a given language ?
- How to model communication buses and protocols ?
- Systematic use of synthesis "pragmas" to express the intention of the designer
  - Guide the tool towards a reasonable solution (parallelism / pipeline)
  - Choosing technical options for a given technology (example RAM / flip-flops)

HLS Methodologies and tools for Systems on Chips

Languages

- No precise definition of the concept of high-level representation of a hardware system.
- No common definition of the usable subset of a given language ?
- How to model communication buses and protocols ?
- Systematic use of synthesis "pragmas" to express the intention of the designer
  - Guide the tool towards a reasonable solution (parallelism / pipeline)
  - Choosing technical options for a given technology (example RAM / flip-flops)

HLS Methodologies and tools for Systems on Chips

Tools

- Replacement of the C or C++ compiler with an ad-hoc compiler
- The ad-hoc compiler is able to generate a RTL representation of the hardware rather than object code.
- Ref. gnu gcc : The '#pragma' directive is the method specified by the C standard for providing additional information to the compiler, beyond what is conveyed in the language itself.
- No standard : different pragmas from one tool to another.

HLS Methodologies and tools for Systems on Chips

Tools

- Replacement of the C or C++ compiler with an ad-hoc compiler
- The ad-hoc compiler is able to generate a RTL representation of the hardware rather than object code.
- Ref. gnu gcc : The '#pragma' directive is the method specified by the C standard for providing additional information to the compiler, beyond what is conveyed in the language itself.
- No standard : different pragmas from one tool to another.

- Replacement of the C or C++ compiler with an ad-hoc compiler
- The ad-hoc compiler is able to generate a RTL representation of the hardware rather than object code.
- Ref. gnu gcc : The '#pragma' directive is the method specified by the C standard for providing additional information to the compiler, beyond what is conveyed in the language itself.
- No standard : different pragmas from one tool to another.

HLS Methodologies and tools for Systems on Chips

Tools

- Replacement of the C or C++ compiler with an ad-hoc compiler
- The ad-hoc compiler is able to generate a RTL representation of the hardware rather than object code.
- Ref. gnu gcc : The '#pragma' directive is the method specified by the C standard for providing additional information to the compiler, beyond what is conveyed in the language itself.
- No standard : different pragmas from one tool to another.

HLS Methodologies and tools for Systems on Chips

A "long" learning time

- A difficult approach because of the paradigm shift (from RTL to HLS)
- Due to the difference in approach, switching from one HLS tool to another is not so easy.
- But a gain in development time for those who master the methodology
- Ridiculously simple examples in the tools tutorials, no realistic example.
  - Example : A beginner in HLS must code a Finite Impulse Response filter ...
  - RTL : Easy to code, if you know in advance the desired level of parallelism or pipeline.
  - HLS : Take the lead with synthesis pragmas to obtain the desired parallelism or pipeline
  - HLS : A frozen I/O diagram that does not necessarily correspond to the desired context of use

HLS Methodologies and tools for Systems on Chips

A "long" learning time

- A difficult approach because of the paradigm shift (from RTL to HLS)
- Due to the difference in approach, switching from one HLS tool to another is not so easy.
- But a gain in development time for those who master the methodology
- Ridiculously simple examples in the tools tutorials, no realistic example.
  - Example : A beginner in HLS must code a Finite Impulse Response filter ...
  - RTL : Easy to code, if you know in advance the desired level of parallelism or pipeline.
  - HLS : Take the lead with synthesis pragmas to obtain the desired parallelism or pipeline
  - HLS : A frozen I/O diagram that does not necessarily correspond to the desired context of use

HLS Methodologies and tools for Systems on Chips

A "long" learning time

- A difficult approach because of the paradigm shift (from RTL to HLS)
- Due to the difference in approach, switching from one HLS tool to another is not so easy.
- But a gain in development time for those who master the methodology
- Ridiculously simple examples in the tools tutorials, no realistic example.
  - Example : A beginner in HLS must code a Finite Impulse Response filter ...
  - RTL : Easy to code, if you know in advance the desired level of parallelism or pipeline.
  - HLS : Take the lead with synthesis pragmas to obtain the desired parallelism or pipeline
  - HLS : A frozen I/O diagram that does not necessarily correspond to the desired context of use

HLS Methodologies and tools for Systems on Chips

A "long" learning time

- A difficult approach because of the paradigm shift (from RTL to HLS)
- Due to the difference in approach, switching from one HLS tool to another is not so easy.
- But a gain in development time for those who master the methodology
- Ridiculously simple examples in the tools tutorials, no realistic example.
  - Example : A beginner in HLS must code a Finite Impulse Response filter ...
  - RTL : Easy to code, if you know in advance the desired level of parallelism or pipeline.
  - HLS : Take the lead with synthesis pragmas to obtain the desired parallelism or pipeline
  - HLS : A frozen I/O diagram that does not necessarily correspond to the desired context of use

HLS Methodologies and tools for Systems on Chips

A "long" learning time

- A difficult approach because of the paradigm shift (from RTL to HLS)
- Due to the difference in approach, switching from one HLS tool to another is not so easy.
- But a gain in development time for those who master the methodology
- Ridiculously simple examples in the tools tutorials, no realistic example.
  - Example : A beginner in HLS must code a Finite Impulse Response filter ...
  - RTL : Easy to code, if you know in advance the desired level of parallelism or pipeline.
  - HLS : Take the lead with synthesis pragmas to obtain the desired parallelism or pipeline
  - HLS : A frozen I/O diagram that does not necessarily correspond to the desired context of use

HLS Methodologies and tools for Systems on Chips

A "long" learning time

- A difficult approach because of the paradigm shift (from RTL to HLS)
- Due to the difference in approach, switching from one HLS tool to another is not so easy.
- But a gain in development time for those who master the methodology
- Ridiculously simple examples in the tools tutorials, no realistic example.
  - Example : A beginner in HLS must code a Finite Impulse Response filter ...
  - RTL : Easy to code, if you know in advance the desired level of parallelism or pipeline.
  - HLS : Take the lead with synthesis pragmas to obtain the desired parallelism or pipeline
  - HLS : A frozen I/O diagram that does not necessarily correspond to the desired context of use

HLS Methodologies and tools for Systems on Chips

A "long" learning time

- A difficult approach because of the paradigm shift (from RTL to HLS)
- Due to the difference in approach, switching from one HLS tool to another is not so easy.
- But a gain in development time for those who master the methodology
- Ridiculously simple examples in the tools tutorials, no realistic example.
  - Example : A beginner in HLS must code a Finite Impulse Response filter ...
  - RTL : Easy to code, if you know in advance the desired level of parallelism or pipeline.
  - HLS : Take the lead with synthesis pragmas to obtain the desired parallelism or pipeline
  - HLS : A frozen I/O diagram that does not necessarily correspond to the desired context of use

HLS Methodologies and tools for Systems on Chips

A "long" learning time

- A difficult approach because of the paradigm shift (from RTL to HLS)
- Due to the difference in approach, switching from one HLS tool to another is not so easy.
- But a gain in development time for those who master the methodology
- Ridiculously simple examples in the tools tutorials, no realistic example.
  - Example : A beginner in HLS must code a Finite Impulse Response filter ...
  - RTL : Easy to code, if you know in advance the desired level of parallelism or pipeline.
  - HLS : Take the lead with synthesis pragmas to obtain the desired parallelism or pipeline
  - HLS : A frozen I/O diagram that does not necessarily correspond to the desired context of use

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Example : The FFT (Fast Fourier Transform)
- There are already some libraries of optimized hardware IPs.
- In HLS, we start (for example) with a reference C code.
  - Rewrite the original C code (with the good subset of C usable by the tool)
  - Find the right combination of pragmas :
    - Loops unrollings or not.
    - Pipelining
    - Store datas in single port memories
    - Store datas in dual port memories
    - Store datas in registers
  - Implement the exploration of architectures (check results for different combinations of pragmas)
    - Provided in some tools, or using external scripting
    - But unbearably long, because it uses multiple RTL synthesis

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Example : The FFT (Fast Fourier Transform)
- There are already some libraries of optimized hardware IPs.
- In HLS, we start (for example) with a reference C code.
  - Rewrite the original C code (with the good subset of C usable by the tool)
  - Find the right combination of pragmas :
    - Loops unrollings or not.
    - Pipelining
    - Store datas in single port memories
    - Store datas in dual port memories
    - Store datas in registers
  - Implement the exploration of architectures (check results for different combinations of pragmas)
    - Provided in some tools, or using external scripting
    - But unbearably long, because it uses multiple RTL synthesis

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Example : The FFT (Fast Fourier Transform)
- There are already some libraries of optimized hardware IPs.
- In HLS, we start (for example) with a reference C code.
  - Rewrite the original C code (with the good subset of C usable by the tool)
  - Find the right combination of pragmas :
    - Loops unrollings or not.
    - Pipelining
    - Store datas in single port memories
    - Store datas in dual port memories
    - Store datas in registers
  - Implement the exploration of architectures (check results for different combinations of pragmas)
    - Provided in some tools, or using external scripting
    - But unbearably long, because it uses multiple RTL synthesis

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Example : The FFT (Fast Fourier Transform)
- There are already some libraries of optimized hardware IPs.
- In HLS, we start (for example) with a reference C code.
  - Rewrite the original C code (with the good subset of C usable by the tool)
  - Find the right combination of pragmas :
    - Loops unrollings or not.
    - Pipelining
    - Store datas in single port memories
    - Store datas in dual port memories
    - Store datas in registers
  - Implement the exploration of architectures (check results for different combinations of pragmas)
    - Provided in some tools, or using external scripting
    - But unbearably long, because it uses multiple RTL synthesis

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Example : The FFT (Fast Fourier Transform)
- There are already some libraries of optimized hardware IPs.
- In HLS, we start (for example) with a reference C code.
  - Rewrite the original C code (with the good subset of C usable by the tool)
  - Find the right combination of pragmas :
    - Loops unrollings or not.
    - Pipelining
    - Store datas in single port memories
    - Store datas in dual port memories
    - Store datas in registers
  - Implement the exploration of architectures (check results for different combinations of pragmas)
    - Provided in some tools, or using external scripting
    - But unbearably long, because it uses multiple RTL synthesis

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Example : The FFT (Fast Fourier Transform)
- There are already some libraries of optimized hardware IPs.
- In HLS, we start (for example) with a reference C code.
  - Rewrite the original C code (with the good subset of C usable by the tool)
  - Find the right combination of pragmas :
    - Loops unrollings or not.
    - Pipelining
    - Store datas in single port memories
    - Store datas in dual port memories
    - Store datas in registers
  - Implement the exploration of architectures (check results for different combinations of pragmas)
    - Provided in some tools, or using external scripting
    - But unbearably long, because it uses multiple RTL synthesis

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Example : The FFT (Fast Fourier Transform)
- There are already some libraries of optimized hardware IPs.
- In HLS, we start (for example) with a reference C code.
  - Rewrite the original C code (with the good subset of C usable by the tool)
  - Find the right combination of pragmas :
    - Loops unrollings or not.
    - Pipelining
    - Store datas in single port memories
    - Store datas in dual port memories
    - Store datas in registers
  - Implement the exploration of architectures (check results for different combinations of pragmas)
    - Provided in some tools, or using external scripting
    - But unbearably long, because it uses multiple RTL synthesis

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Example : The FFT (Fast Fourier Transform)
- There are already some libraries of optimized hardware IPs.
- In HLS, we start (for example) with a reference C code.
  - Rewrite the original C code (with the good subset of C usable by the tool)
  - Find the right combination of pragmas :
    - Loops unrollings or not.
    - Pipelining
    - Store datas in single port memories
    - Store datas in dual port memories
    - Store datas in registers
  - Implement the exploration of architectures (check results for different combinations of pragmas)
    - Provided in some tools, or using external scripting
    - But unbearably long, because it uses multiple RTL synthesis

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Example : The FFT (Fast Fourier Transform)
- There are already some libraries of optimized hardware IPs.
- In HLS, we start (for example) with a reference C code.
  - Rewrite the original C code (with the good subset of C usable by the tool)
  - Find the right combination of pragmas :
    - Loops unrollings or not.
    - Pipelining
    - Store datas in single port memories
    - Store datas in dual port memories
    - Store datas in registers
  - Implement the exploration of architectures (check results for different combinations of pragmas)
    - Provided in some tools, or using external scripting
    - But unbearably long, because it uses multiple RTL synthesis

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Example : The FFT (Fast Fourier Transform)
- There are already some libraries of optimized hardware IPs.
- In HLS, we start (for example) with a reference C code.
  - Rewrite the original C code (with the good subset of C usable by the tool)
  - Find the right combination of pragmas :
    - Loops unrollings or not.
    - Pipelining
    - Store datas in single port memories
    - Store datas in dual port memories
    - Store datas in registers
  - Implement the exploration of architectures (check results for different combinations of pragmas)
    - Provided in some tools, or using external scripting
    - But unbearably long, because it uses multiple RTL synthesis

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Example : The FFT (Fast Fourier Transform)
- There are already some libraries of optimized hardware IPs.
- In HLS, we start (for example) with a reference C code.
  - Rewrite the original C code (with the good subset of C usable by the tool)
  - Find the right combination of pragmas :
    - Loops unrollings or not.
    - Pipelining
    - Store datas in single port memories
    - Store datas in dual port memories
    - Store datas in registers
  - Implement the exploration of architectures (check results for different combinations of pragmas)
    - Provided in some tools, or using external scripting
    - But unbearably long, because it uses multiple RTL synthesis

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Example : The FFT (Fast Fourier Transform)
- There are already some libraries of optimized hardware IPs.
- In HLS, we start (for example) with a reference C code.
  - Rewrite the original C code (with the good subset of C usable by the tool)
  - Find the right combination of pragmas :
    - Loops unrollings or not.
    - Pipelining
    - Store datas in single port memories
    - Store datas in dual port memories
    - Store datas in registers
  - Implement the exploration of architectures (check results for different combinations of pragmas)
    - Provided in some tools, or using external scripting
    - But unbearably long, because it uses multiple RTL synthesis

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Example : The FFT (Fast Fourier Transform)
- There are already some libraries of optimized hardware IPs.
- In HLS, we start (for example) with a reference C code.
  - Rewrite the original C code (with the good subset of C usable by the tool)
  - Find the right combination of pragmas :
    - Loops unrollings or not.
    - Pipelining
    - Store datas in single port memories
    - Store datas in dual port memories
    - Store datas in registers
  - Implement the exploration of architectures (check results for different combinations of pragmas)
    - Provided in some tools, or using external scripting
    - But unbearably long, because it uses multiple RTL synthesis

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Example : The FFT (Fast Fourier Transform)
- There are already some libraries of optimized hardware IPs.
- In HLS, we start (for example) with a reference C code.
  - Rewrite the original C code (with the good subset of C usable by the tool)
  - Find the right combination of pragmas :
    - Loops unrollings or not.
    - Pipelining
    - Store datas in single port memories
    - Store datas in dual port memories
    - Store datas in registers
  - Implement the exploration of architectures (check results for different combinations of pragmas)
    - Provided in some tools, or using external scripting
    - But unbearably long, because it uses multiple RTL synthesis

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Reduce long and tedious coding times (RTL) :
  - Writing control state machines.
  - Writing communication codes to the outside world (ad-hoc communications, standardized buses)
- The real problem is not "I want to code a FFT"
- The real problem is "I want to encode a FFT ..."
  - Which communicates with a slave bus of the standard XXX
  - With a YYY width data bus
  - With processed data of width ZZZ
  - With internal datas using type TTT (floating point, fixed point, integer...)
  - With packet data transfers for FFT datas

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Reduce long and tedious coding times (RTL) :
  - Writing control state machines.
  - Writing communication codes to the outside world (ad-hoc communications, standardized buses)
- The real problem is not "I want to code a FFT"
- The real problem is "I want to encode a FFT ..."
  - Which communicates with a slave bus of the standard XXX
  - With a YYY width data bus
  - With processed data of width ZZZ
  - With internal datas using type TTT (floating point, fixed point, integer...)
  - With packet data transfers for FFT datas

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Reduce long and tedious coding times (RTL) :
  - Writing control state machines.
  - Writing communication codes to the outside world (ad-hoc communications, standardized buses)
- The real problem is not "I want to code a FFT"
- The real problem is "I want to encode a FFT ..."
  - Which communicates with a slave bus of the standard XXX
  - With a YYY width data bus
  - With processed data of width ZZZ
  - With internal datas using type TTT (floating point, fixed point, integer...)
  - With packet data transfers for FFT datas

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Reduce long and tedious coding times (RTL) :
  - Writing control state machines.
  - Writing communication codes to the outside world (ad-hoc communications, standardized buses)
- The real problem is not "I want to code a FFT"
- The real problem is "I want to encode a FFT ..."
  - Which communicates with a slave bus of the standard XXX
  - With a YYY width data bus
  - With processed data of width ZZZ
  - With internal datas using type TTT (floating point, fixed point, integer...)
  - With packet data transfers for FFT datas

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Reduce long and tedious coding times (RTL) :
  - Writing control state machines.
  - Writing communication codes to the outside world (ad-hoc communications, standardized buses)
- The real problem is not "I want to code a FFT"
- The real problem is "I want to encode a FFT ..."
  - Which communicates with a slave bus of the standard XXX
  - With a YYY width data bus
  - With processed data of width ZZZ
  - With internal datas using type TTT (floating point, fixed point, integer...)
  - With packet data transfers for FFT datas

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Reduce long and tedious coding times (RTL) :
  - Writing control state machines.
  - Writing communication codes to the outside world (ad-hoc communications, standardized buses)
- The real problem is not "I want to code a FFT"
- The real problem is "I want to encode a FFT ..."
  - Which communicates with a slave bus of the standard XXX
  - With a YYY width data bus
  - With processed data of width ZZZ
  - With internal datas using type TTT (floating point, fixed point, integer...)
  - With packet data transfers for FFT datas

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Reduce long and tedious coding times (RTL) :
  - Writing control state machines.
  - Writing communication codes to the outside world (ad-hoc communications, standardized buses)
- The real problem is not "I want to code a FFT"
- The real problem is "I want to encode a FFT ..."
  - Which communicates with a slave bus of the standard XXX
  - With a YYY width data bus
  - With processed data of width ZZZ
  - With internal datas using type TTT (floating point, fixed point, integer...)
  - With packet data transfers for FFT datas

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Reduce long and tedious coding times (RTL) :
  - Writing control state machines.
  - Writing communication codes to the outside world (ad-hoc communications, standardized buses)
- The real problem is not "I want to code a FFT"
- The real problem is "I want to encode a FFT ..."
  - Which communicates with a slave bus of the standard XXX
  - With a YYY width data bus
  - With processed data of width ZZZ
  - With internal datas using type TTT (floating point, fixed point, integer...)
  - With packet data transfers for FFT datas

HLS Methodologies and tools for Systems on Chips

What question do we want to solve ?

- Reduce long and tedious coding times (RTL) :
  - Writing control state machines.
  - Writing communication codes to the outside world (ad-hoc communications, standardized buses)
- The real problem is not "I want to code a FFT"
- The real problem is "I want to encode a FFT ..."
  - Which communicates with a slave bus of the standard XXX
  - With a YYY width data bus
  - With processed data of width ZZZ
  - With internal datas using type TTT (floating point, fixed point, integer...)
  - With packet data transfers for FFT datas

HLS Methodologies and tools for Systems on Chips

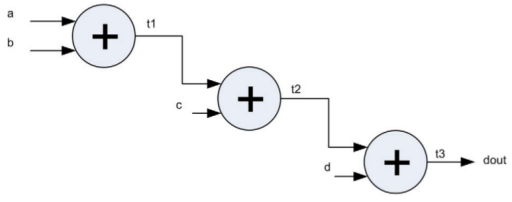
What question do we want to solve ?

- Reduce long and tedious coding times (RTL) :
  - Writing control state machines.
  - Writing communication codes to the outside world (ad-hoc communications, standardized buses)
- The real problem is not "I want to code a FFT"
- The real problem is "I want to encode a FFT ..."
  - Which communicates with a slave bus of the standard XXX
  - With a YYY width data bus
  - With processed data of width ZZZ
  - With internal datas using type TTT (floating point, fixed point, integer...)
  - With packet data transfers for FFT datas

Outline

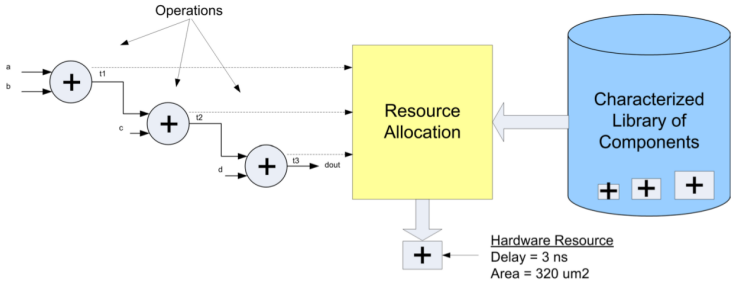
Step1 : Data Flow Graph

```
#include "accum.h"
void accumulate(int a, int b, int c, int d, int
&dout)
{
  int t1,t2;
  t1 = a + b;
  t2 = t1 + c;
  dout = t2 + d;
}
```



0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

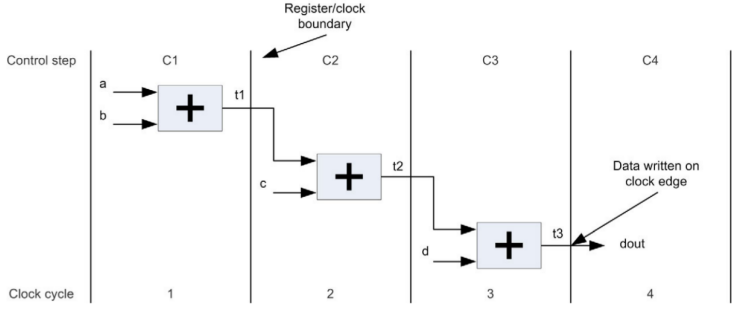
Step2 : Resource Allocation



- Choice of operators, estimation of datapaths sizes
- Area and delay estimation for each operator
- Requires knowledge of technological data

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

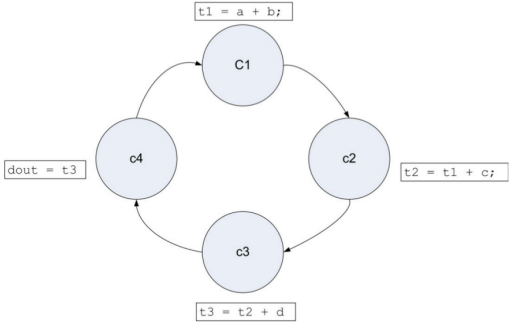
Step 3 : Scheduling



- Assumption : $T_{addition} < T_{clk}$
- Allocation of each operation to a given cycle.
- Storage of intermediate results into registers.

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

Step 4 : Control

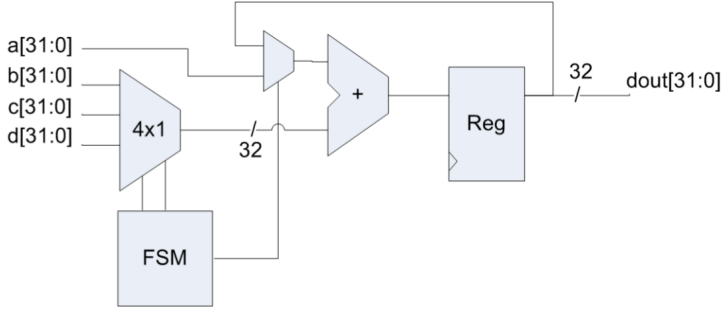


- Finite State Machine associated to the data path.
- Drives the choosen schedule.

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

"Hardware implementation"

A constraint-less design

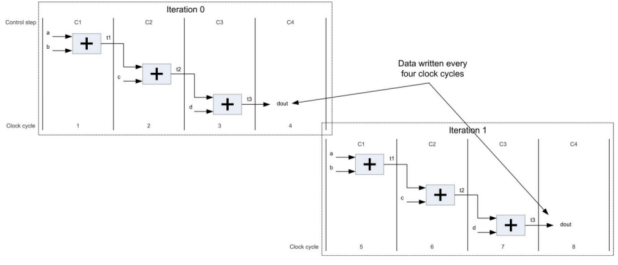


- Resources minimization
- Only one adder and a finite state machine.

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

Loop Pipelining

- **Initiation Interval (II)** : how many cycles between each loop usage.
- Example : II=1 means one new computation at each clock cycle.
- **"Latency" (L)** : How many cycles between the arrival of the first input data and the first output data.

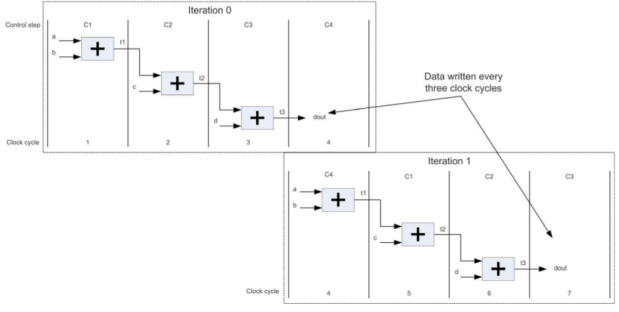


No constraint -> L=3, II=4

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

Loop Pipelining : II=3, L=3

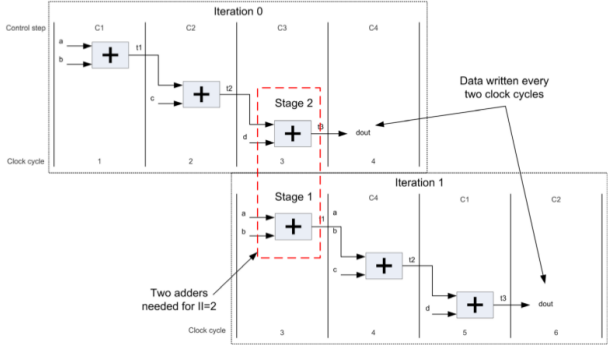
- Synthesis constraints using pragmas or scripts...
- Note : In C4 an output at the same time as an input.
- We assume that there are no constraints on the I/Os



0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

Loop Pipelining : II=2, L=3

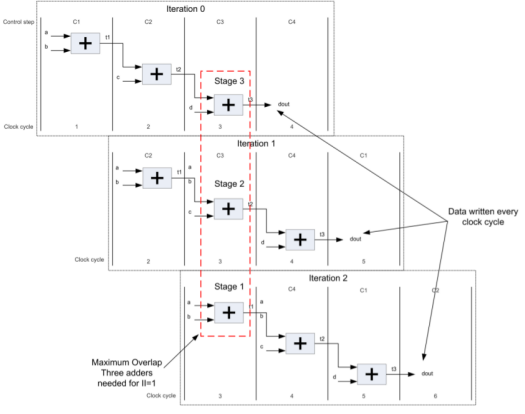
- Synthesis constraints using pragmas or scripts...
- You need 2 adders in parallel.
- We assume that there are no constraints on the I/Os



0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

Loop Pipelining : II=1, L=3

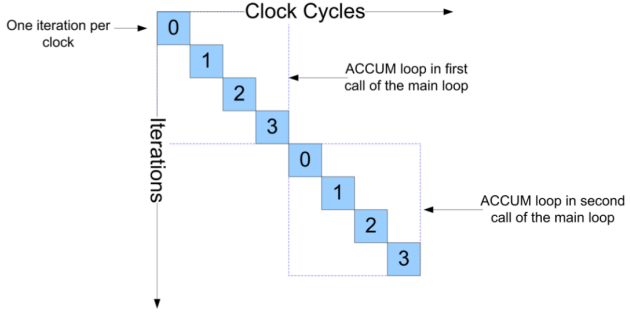
- You need 3 adders in parallel : Maximum performance is achieved.



0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

Loop Unrolling

- We do not play with the iterations of the loop.
- We play with parallelism inside the loop.

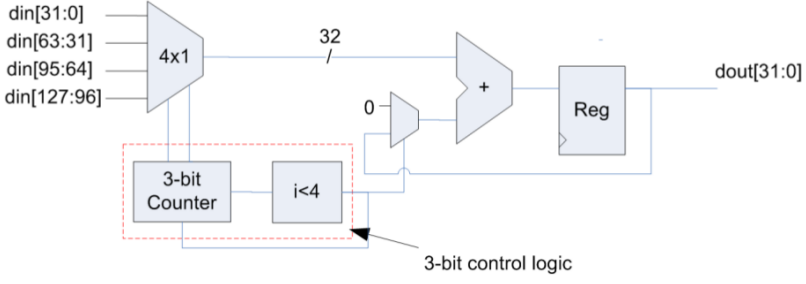


Initial scheduling showing 2 successive calls to the loop.

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

Loop Unrolling : Initial implementation, II=4

- Loading of "din[31 :0]+0" at first cycle.
- Computing accumulation at other 3 cycles.

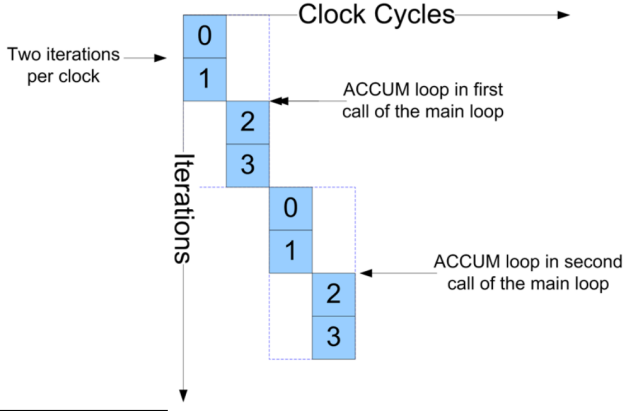


Hardware Implementation

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

Loop Unrolling : Partial unrolling, II=2

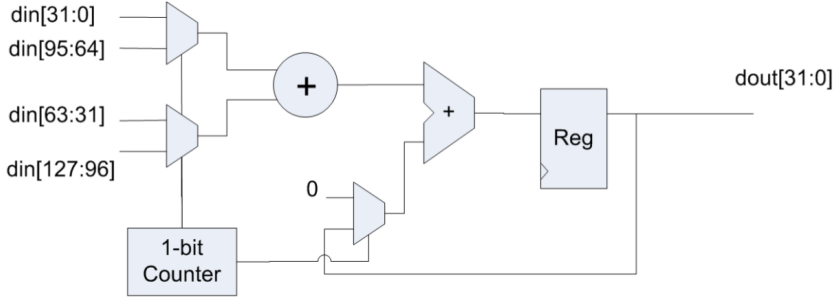
- New scheduling using an unrolling factor equal to 2
- Two input datas received during the same clock cycle.



0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

Loop Unrolling : Partial unrolling, II=2

- Still Only one adder needed.
- Smaller counter needed.

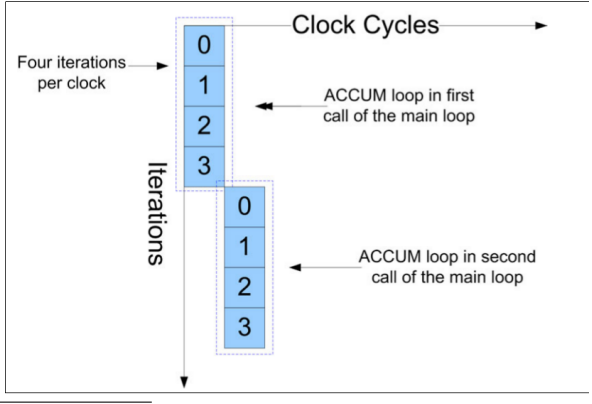


Hardware Implementation

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

Loop Unrolling : Full unrolling, II=1

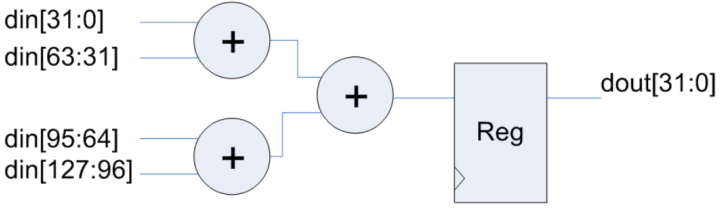
- New scheduling using an unrolling factor equal to 4
- Four input datas received during the same clock cycle.



0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

Loop Unrolling : Full unrolling

- Three adders, no counter.



Hardware Implementation

0. Source : High Level Synthesis Blue Book (Michael Fingerhoff/Mentor Graphics)

Loop pipelining versus Loop unrolling

- Loop pipelining
  - Uses parallelism at a global level.
  - Increase area as **II** is decreased.
  - At a given cycle, inputs and outputs are related to several calls of the loop
- Loop unrolling
  - Uses parallelism at a local level.
  - Increase clock frequency constraints as **II** is decreased.
  - At a given cycle, inputs and outputs are related to only one call of the loop

Loop pipelining versus Loop unrolling

- Loop pipelining
  - Uses parallelism at a global level.
  - Increase area as **II** is decreased.
  - At a given cycle, inputs and outputs are related to several calls of the loop
- Loop unrolling
  - Uses parallelism at a local level.
  - Increase clock frequency constraints as **II** is decreased.
  - At a given cycle, inputs and outputs are related to only one call of the loop

"Loops : the core of the HLS tools"

- Simple loops with well known number of iterations.
- Loop Pipelining and Loop Unrolling may be used together.
- Automatic synthesis of the finite state machines.
  - RTL : Long and tedious coding of the start and end of pipeline behavior.
  - RTL : Long and tedious debugging.
  - RTL : Very often, coding of only one alternative
- Bonus : Easy exploration of different alternatives.
- Warning : The optimal result can be counterintuitive.
- Standard in all HLS tools but not necessarily with the same vocabulary

"Loops : the core of the HLS tools"

- Simple loops with well known number of iterations.
- Loop Pipelining and Loop Unrolling may be used together.
- Automatic synthesis of the finite state machines.
  - RTL : Long and tedious coding of the start and end of pipeline behavior.
  - RTL : Long and tedious debugging.
  - RTL : Very often, coding of only one alternative
- Bonus : Easy exploration of different alternatives.
- Warning : The optimal result can be counterintuitive.
- Standard in all HLS tools but not necessarily with the same vocabulary

"Loops : the core of the HLS tools"

- Simple loops with well known number of iterations.
- Loop Pipelining and Loop Unrolling may be used together.
- Automatic synthesis of the finite state machines.
  - RTL : Long and tedious coding of the start and end of pipeline behavior.
  - RTL : Long and tedious debugging.
  - RTL : Very often, coding of only one alternative
- Bonus : Easy exploration of different alternatives.
- Warning : The optimal result can be counterintuitive.
- Standard in all HLS tools but not necessarily with the same vocabulary

"Loops : the core of the HLS tools"

- Simple loops with well known number of iterations.
- Loop Pipelining and Loop Unrolling may be used together.
- Automatic synthesis of the finite state machines.
  - RTL : Long and tedious coding of the start and end of pipeline behavior.
  - RTL : Long and tedious debugging.
  - RTL : Very often, coding of only one alternative
- Bonus : Easy exploration of different alternatives.
- Warning : The optimal result can be counterintuitive.
- Standard in all HLS tools but not necessarily with the same vocabulary

"Loops : the core of the HLS tools"

- Simple loops with well known number of iterations.
- Loop Pipelining and Loop Unrolling may be used together.
- Automatic synthesis of the finite state machines.
  - RTL : Long and tedious coding of the start and end of pipeline behavior.
  - RTL : Long and tedious debugging.
  - RTL : Very often, coding of only one alternative
- Bonus : Easy exploration of different alternatives.
- Warning : The optimal result can be counterintuitive.
- Standard in all HLS tools but not necessarily with the same vocabulary

"Loops : the core of the HLS tools"

- Simple loops with well known number of iterations.
- Loop Pipelining and Loop Unrolling may be used together.
- Automatic synthesis of the finite state machines.
  - RTL : Long and tedious coding of the start and end of pipeline behavior.
  - RTL : Long and tedious debugging.
  - RTL : Very often, coding of only one alternative
- Bonus : Easy exploration of different alternatives.
- Warning : The optimal result can be counterintuitive.
- Standard in all HLS tools but not necessarily with the same vocabulary

"Loops : the core of the HLS tools"

- Simple loops with well known number of iterations.
- Loop Pipelining and Loop Unrolling may be used together.
- Automatic synthesis of the finite state machines.
  - RTL : Long and tedious coding of the start and end of pipeline behavior.
  - RTL : Long and tedious debugging.
  - RTL : Very often, coding of only one alternative
- Bonus : Easy exploration of different alternatives.
- Warning : The optimal result can be counterintuitive.
- Standard in all HLS tools but not necessarily with the same vocabulary

"Loops : the core of the HLS tools"

- Simple loops with well known number of iterations.
- Loop Pipelining and Loop Unrolling may be used together.
- Automatic synthesis of the finite state machines.
  - RTL : Long and tedious coding of the start and end of pipeline behavior.
  - RTL : Long and tedious debugging.
  - RTL : Very often, coding of only one alternative
- Bonus : Easy exploration of different alternatives.
- Warning : The optimal result can be counterintuitive.
- Standard in all HLS tools but not necessarily with the same vocabulary

Outline

C/C++ : Communications ports

- C/C++ language : hardware seen as a function.
- Function I/Os are seen as communication ports.
- The design should be able to choose the communication protocol.
- Solution : Using C++ pragmas.

```
void example(int A[50], int B[50]) {
//Set the HLS native interface types
#pragma HLS INTERFACE axis port=A
#pragma HLS INTERFACE axis port=B
  int i;
  for(i = 0; i < 50; i++){
     B[i] = A[i] + 5;
  }
}
```

Xilinx Vitis-HLS : Two separated AXI4-stream ports

C/C++ : Communications ports

- C/C++ language : hardware seen as a function.
- Function I/Os are seen as communication ports.
- The design should be able to choose the communication protocol.
- Solution : Using C++ pragmas.

```
void example(int A[50], int B[50]) {
//Set the HLS native interface types
#pragma HLS INTERFACE axis port=A
#pragma HLS INTERFACE axis port=B
  int i;
  for(i = 0; i < 50; i++){
     B[i] = A[i] + 5;
  }
}
```

Xilinx Vitis-HLS : Two separated AXI4-stream ports

C/C++ : Communications ports

- C/C++ language : hardware seen as a function.
- Function I/Os are seen as communication ports.
- The design should be able to choose the communication protocol.
- Solution : Using C++ pragmas.

```
void example(int A[50], int B[50]) {
//Set the HLS native interface types
#pragma HLS INTERFACE axis port=A
#pragma HLS INTERFACE axis port=B
  int i;
  for(i = 0; i < 50; i++){
     B[i] = A[i] + 5;
  }
}
```

Xilinx Vitis-HLS : Two separated AXI4-stream ports

C/C++ : Communications ports

- C/C++ language : hardware seen as a function.
- Function I/Os are seen as communication ports.
- The design should be able to choose the communication protocol.
- Solution : Using C++ pragmas.

```
void example(int A[50], int B[50]) {
//Set the HLS native interface types
#pragma HLS INTERFACE axis port=A
#pragma HLS INTERFACE axis port=B
  int i;
  for(i = 0; i < 50; i++){
     B[i] = A[i] + 5;
  }
}
```

Xilinx Vitis-HLS : Two separated AXI4-stream ports

C/C++ : Communication ports

C/C++

- Sharing the same bus for several inputs/outputs
- Defining base addresses for port c
- Force usage of "valid handshake" for port b

```
void example(char *a, char *b, char *c) {
#pragma HLS INTERFACE s_axilite port=return bundle=BUS_A
#pragma HLS INTERFACE s_axilite port=a bundle=BUS_A
#pragma HLS INTERFACE s_axilite port=b bundle=BUS_A
#pragma HLS INTERFACE s_axilite port=c bundle=BUS_A offset=0x0400
#pragma HLS INTERFACE ap_vld port=b
  *c += *a + *b;
}
```

Xilinx Vitis-HLS : Only one AXI4-lite port for all I/Os...

C/C++ : Communication ports

C/C++

- Sharing the same bus for several inputs/outputs
- Defining base addresses for port c
- Force usage of "valid handshake" for port b

```
void example(char *a, char *b, char *c) {
#pragma HLS INTERFACE s_axilite port=return bundle=BUS_A
#pragma HLS INTERFACE s_axilite port=a bundle=BUS_A
#pragma HLS INTERFACE s_axilite port=b bundle=BUS_A
#pragma HLS INTERFACE s_axilite port=c bundle=BUS_A offset=0x0400
#pragma HLS INTERFACE ap_vld port=b
  *c += *a + *b;
}
```

Xilinx Vitis-HLS : Only one AXI4-lite port for all I/Os...

C/C++ : Communication ports

C/C++

- Sharing the same bus for several inputs/outputs
- Defining base addresses for port c
- Force usage of "valid handshake" for port b

```
void example(char *a, char *b, char *c) {
#pragma HLS INTERFACE s_axilite port=return bundle=BUS_A
#pragma HLS INTERFACE s_axilite port=a bundle=BUS_A
#pragma HLS INTERFACE s_axilite port=b bundle=BUS_A
#pragma HLS INTERFACE s_axilite port=c bundle=BUS_A offset=0x0400
#pragma HLS INTERFACE ap_vld port=b
  *c += *a + *b;
}
```

Xilinx Vitis-HLS : Only one AXI4-lite port for all I/Os...

SystemC : Communication ports

- Recognition of specific SystemC constructs like **sc_in**, **sc_out** or **sc_fifo**

```
SC_MODULE(sc_sequ_cthread){
    sc_fifo_out<int> dout;
    sc_fifo_in<int> din;
    ...
}

void sc_FIFO_port::Prc2() {
#pragma HLS resource core=AXI4Stream variable=din
#pragma HLS resource core=AXI4Stream variable=dout
  ...
}
```

Xilinx Vitis-HLS : SystemC specific constructs

C/C++ Mathematical support

C/C++

- DSP processing needs handling of the precision of mathematical datas :
  - Integer/Floating-Point/Fixed Point datatypes
  - Arbitrary precision data-types.
- VHDL/Verilog RTL supports arbitrary precision integer datatypes but no floating point, and hardly fixed-point.
- SystemC supports arbitrary precision integer or fixed-point datatypes and standard floating point.
- C/C++ doesn't support natively arbitrary precision datatypes.
- Solution : **Algorithm C (AC) Datatypes**
  - Specific templated classes for C++
  - Integers with arbitrary width
  - Fixed Point with arbitrary width and arbitrary integer part width
  - Floating Point with arbitrary mantissa width and arbitrary exponant width

C/C++ Mathematical support

C/C++

- DSP processing needs handling of the precision of mathematical datas :
  - Integer/Floating-Point/Fixed Point datatypes
  - Arbitrary precision data-types.
- VHDL/Verilog RTL supports arbitrary precision integer datatypes but no floating point, and hardly fixed-point.
- SystemC supports arbitrary precision integer or fixed-point datatypes and standard floating point.
- C/C++ doesn't support natively arbitrary precision datatypes.
- Solution : **Algorithm C (AC) Datatypes**
  - Specific templated classes for C++
  - Integers with arbitrary width
  - Fixed Point with arbitrary width and arbitrary integer part width
  - Floating Point with arbitrary mantissa width and arbitrary exponant width

C/C++ Mathematical support

C/C++

- DSP processing needs handling of the precision of mathematical datas :
  - Integer/Floating-Point/Fixed Point datatypes
  - Arbitrary precision data-types.
- VHDL/Verilog RTL supports arbitrary precision integer datatypes but no floating point, and hardly fixed-point.
- SystemC supports arbitrary precision integer or fixed-point datatypes and standard floating point.
- C/C++ doesn't support natively arbitrary precision datatypes.
- Solution : **Algorithm C (AC) Datatypes**
  - Specific templated classes for C++
  - Integers with arbitrary width
  - Fixed Point with arbitrary width and arbitrary integer part width
  - Floating Point with arbitrary mantissa width and arbitrary exponant width

C/C++ Mathematical support

C/C++

- DSP processing needs handling of the precision of mathematical datas :
  - Integer/Floating-Point/Fixed Point datatypes
  - Arbitrary precision data-types.
- VHDL/Verilog RTL supports arbitrary precision integer datatypes but no floating point, and hardly fixed-point.
- SystemC supports arbitrary precision integer or fixed-point datatypes and standard floating point.
- C/C++ doesn't support natively arbitrary precision datatypes.
- Solution : **Algorithm C (AC) Datatypes**
  - Specific templated classes for C++
  - Integers with arbitrary width
  - Fixed Point with arbitrary width and arbitrary integer part width
  - Floating Point with arbitrary mantissa width and arbitrary exponant width

C/C++ Mathematical support

C/C++

- DSP processing needs handling of the precision of mathematical datas :
  - Integer/Floating-Point/Fixed Point datatypes
  - Arbitrary precision data-types.
- VHDL/Verilog RTL supports arbitrary precision integer datatypes but no floating point, and hardly fixed-point.
- SystemC supports arbitrary precision integer or fixed-point datatypes and standard floating point.
- C/C++ doesn't support natively arbitrary precision datatypes.
- Solution : **Algorithm C (AC) Datatypes**
  - Specific templated classes for C++
  - Integers with arbitrary width
  - Fixed Point with arbitrary width and arbitrary integer part width
  - Floating Point with arbitrary mantissa width and arbitrary exponant width

C/C++ Mathematical support

C/C++

- DSP processing needs handling of the precision of mathematical datas :
  - Integer/Floating-Point/Fixed Point datatypes
  - Arbitrary precision data-types.
- VHDL/Verilog RTL supports arbitrary precision integer datatypes but no floating point, and hardly fixed-point.
- SystemC supports arbitrary precision integer or fixed-point datatypes and standard floating point.
- C/C++ doesn't support natively arbitrary precision datatypes.
- Solution : **Algorithm C (AC) Datatypes**
  - Specific templated classes for C++
  - Integers with arbitrary width
  - Fixed Point with arbitrary width and arbitrary integer part width
  - Floating Point with arbitrary mantissa width and arbitrary exponant width

C/C++ Mathematical support

C/C++

- DSP processing needs handling of the precision of mathematical datas :
  - Integer/Floating-Point/Fixed Point datatypes
  - Arbitrary precision data-types.
- VHDL/Verilog RTL supports arbitrary precision integer datatypes but no floating point, and hardly fixed-point.
- SystemC supports arbitrary precision integer or fixed-point datatypes and standard floating point.
- C/C++ doesn't support natively arbitrary precision datatypes.
- Solution : **Algorithm C (AC) Datatypes**
  - Specific templated classes for C++
  - Integers with arbitrary width
  - Fixed Point with arbitrary width and arbitrary integer part width
  - Floating Point with arbitrary mantissa width and arbitrary exponant width

C/C++ Mathematical support

C/C++

- DSP processing needs handling of the precision of mathematical datas :
  - Integer/Floating-Point/Fixed Point datatypes
  - Arbitrary precision data-types.
- VHDL/Verilog RTL supports arbitrary precision integer datatypes but no floating point, and hardly fixed-point.
- SystemC supports arbitrary precision integer or fixed-point datatypes and standard floating point.
- C/C++ doesn't support natively arbitrary precision datatypes.
- Solution : **Algorithm C (AC) Datatypes**
  - Specific templated classes for C++
  - Integers with arbitrary width
  - Fixed Point with arbitrary width and arbitrary integer part width
  - Floating Point with arbitrary mantissa width and arbitrary exponant width

C/C++ Mathematical support

C/C++

- DSP processing needs handling of the precision of mathematical datas :
  - Integer/Floating-Point/Fixed Point datatypes
  - Arbitrary precision data-types.
- VHDL/Verilog RTL supports arbitrary precision integer datatypes but no floating point, and hardly fixed-point.
- SystemC supports arbitrary precision integer or fixed-point datatypes and standard floating point.
- C/C++ doesn't support natively arbitrary precision datatypes.
- Solution : **Algorithm C (AC) Datatypes**
  - Specific templated classes for C++
  - Integers with arbitrary width
  - Fixed Point with arbitrary width and arbitrary integer part width
  - Floating Point with arbitrary mantissa width and arbitrary exponant width

Outline

Some known HLS tools

Tool	Owner	Input language	Target	Comment
Stratus HLS	Cadence	c/c++ SystemC	ASIC/FPGA	RTL/HLS
Symphony C	Synopsys	c/c++	ASIC/FPGA	
Catapult	Mentor	c/c++ SystemC	ASIC/FPGA	
Intel HLS	Intel FPGA	c/c++	FPGA	Intel FPGAs
Vitis HLS	Xilinx	c/c++ SystemC	FPGA	Xilinx FPGAs
Legup HLS	Microchip Tech.	c/c++	FPGA	Microchip FPGAs

Outline

The FPGA case

heterogeneous architectures

- Language : OpenCL parallel programming
- originaly targeted for CPU/GPU
- SPMD parallelism (Single Program Multiple Data)
- Extended to CPU/FPGA associations
- SocFPGA (CPU and FPGA integrated in the same circuit)
- FPGA based hadware accelerators in the cloud (data centers).
- In general : one tool for specific destination platforms

The FPGA case

heterogeneous architectures

- Language : OpenCL parallel programming
- originaly targeted for CPU/GPU
- SPMD parallelism (Single Program Multiple Data)
- Extended to CPU/FPGA associations
- SocFPGA (CPU and FPGA integrated in the same circuit)
- FPGA based hadware accelerators in the cloud (data centers).
- In general : one tool for specific destination platforms

The FPGA case

heterogeneous architectures

- Language : OpenCL parallel programming
- originaly targeted for CPU/GPU
- SPMD parallelism (Single Program Multiple Data)
- Extended to CPU/FPGA associations
- SocFPGA (CPU and FPGA integrated in the same circuit)
- FPGA based hadware accelerators in the cloud (data centers).
- In general : one tool for specific destination platforms

The FPGA case

heterogeneous architectures

- Language : OpenCL parallel programming
- originaly targeted for CPU/GPU
- SPMD parallelism (Single Program Multiple Data)
- Extended to CPU/FPGA associations
- SocFPGA (CPU and FPGA integrated in the same circuit)
- FPGA based hadware accelerators in the cloud (data centers).
- In general : one tool for specific destination platforms

The FPGA case

heterogeneous architectures

- Language : OpenCL parallel programming
- originaly targeted for CPU/GPU
- SPMD parallelism (Single Program Multiple Data)
- Extended to CPU/FPGA associations
- SocFPGA (CPU and FPGA integrated in the same circuit)
- FPGA based hadware accelerators in the cloud (data centers).
- In general : one tool for specific destination platforms

The FPGA case

heterogeneous architectures

- Language : OpenCL parallel programming
- originaly targeted for CPU/GPU
- SPMD parallelism (Single Program Multiple Data)
- Extended to CPU/FPGA associations
- SocFPGA (CPU and FPGA integrated in the same circuit)
- FPGA based hadware accelerators in the cloud (data centers).
- In general : one tool for specific destination platforms

Intel "OneAPI"