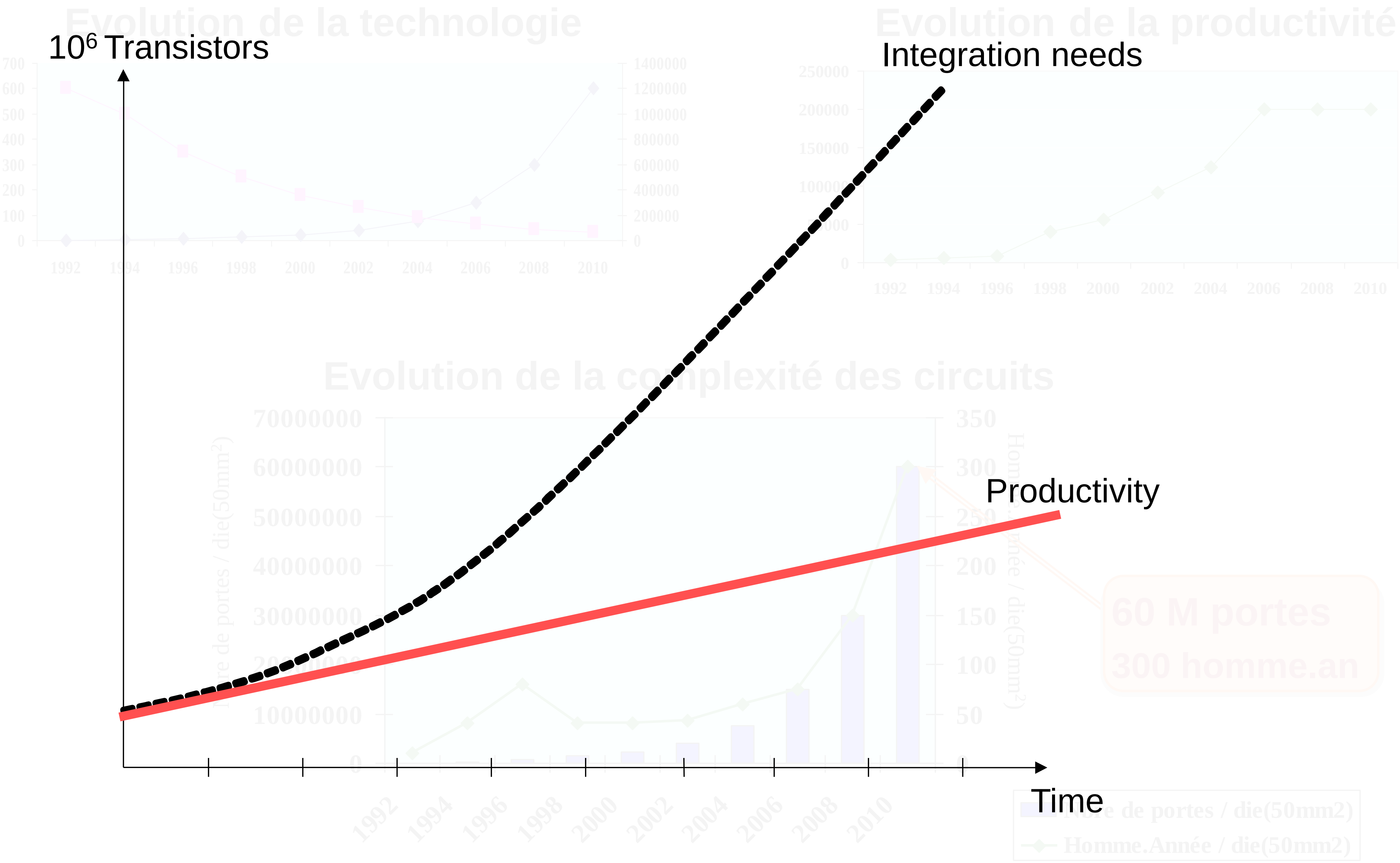
Une introduction à la synthèse de haut-niveau

(ou comment générer des architectures matérielles à partir du langage C)

Université de Bretagne-Sud
Lab-STICC

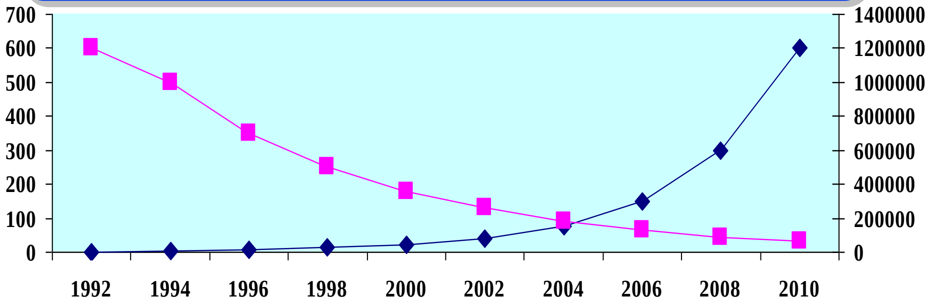
Philippe COUSSY
philippe.coussy@univ-ubs.fr

Productivity gap

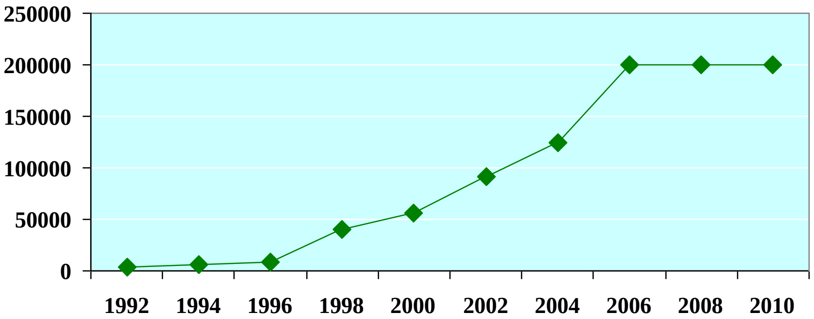


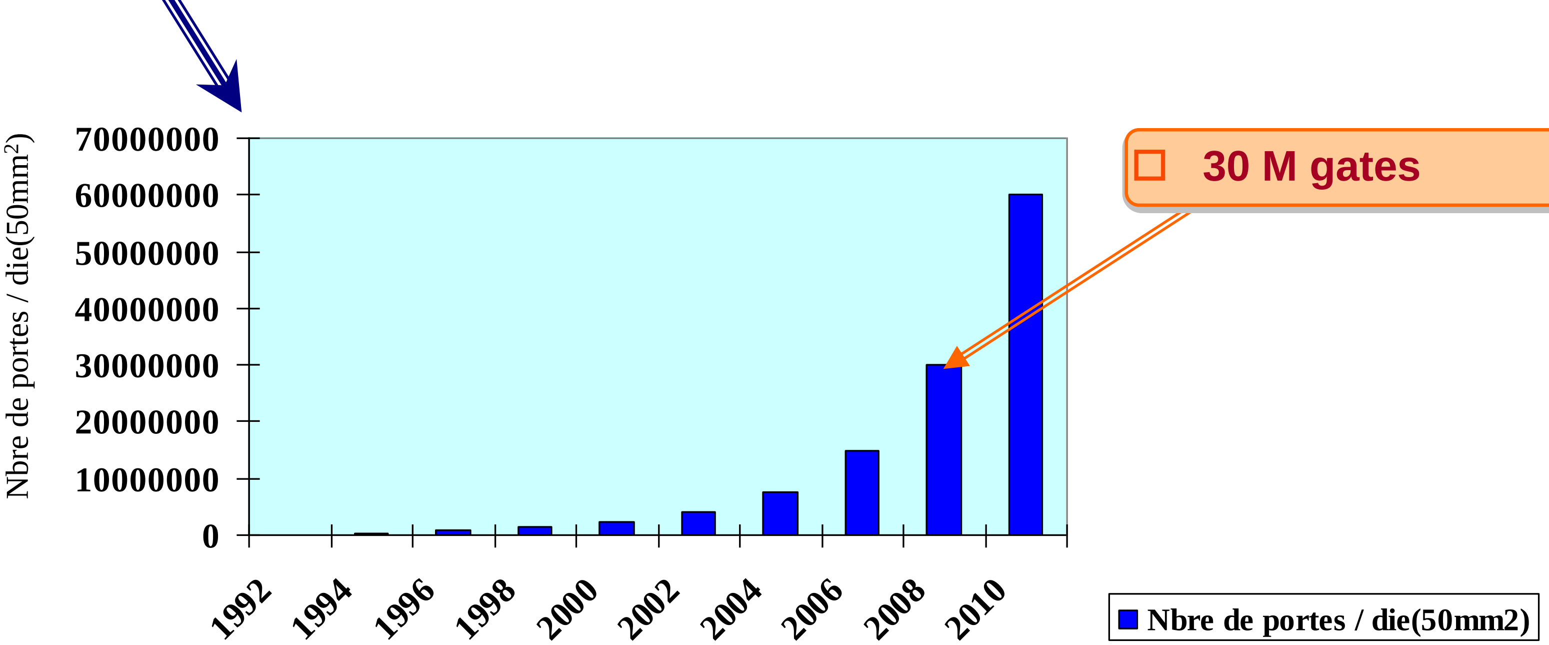
Context

- □ Decreasing technology scale
- □ Increasing transistor density



- □ Designer productivity



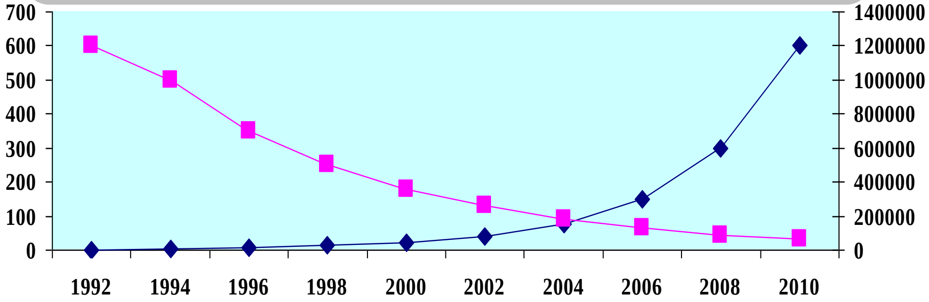


- □ 30 M gates

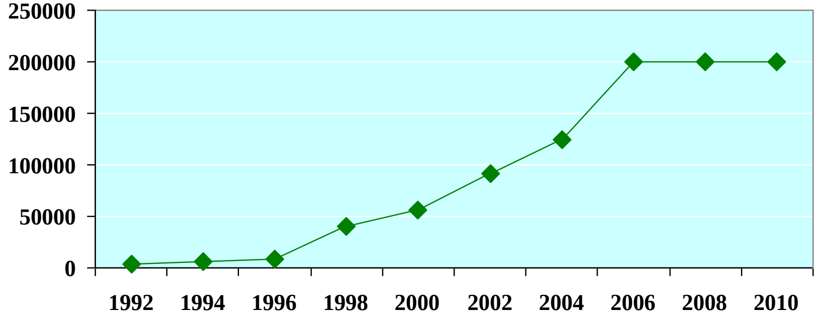
Context

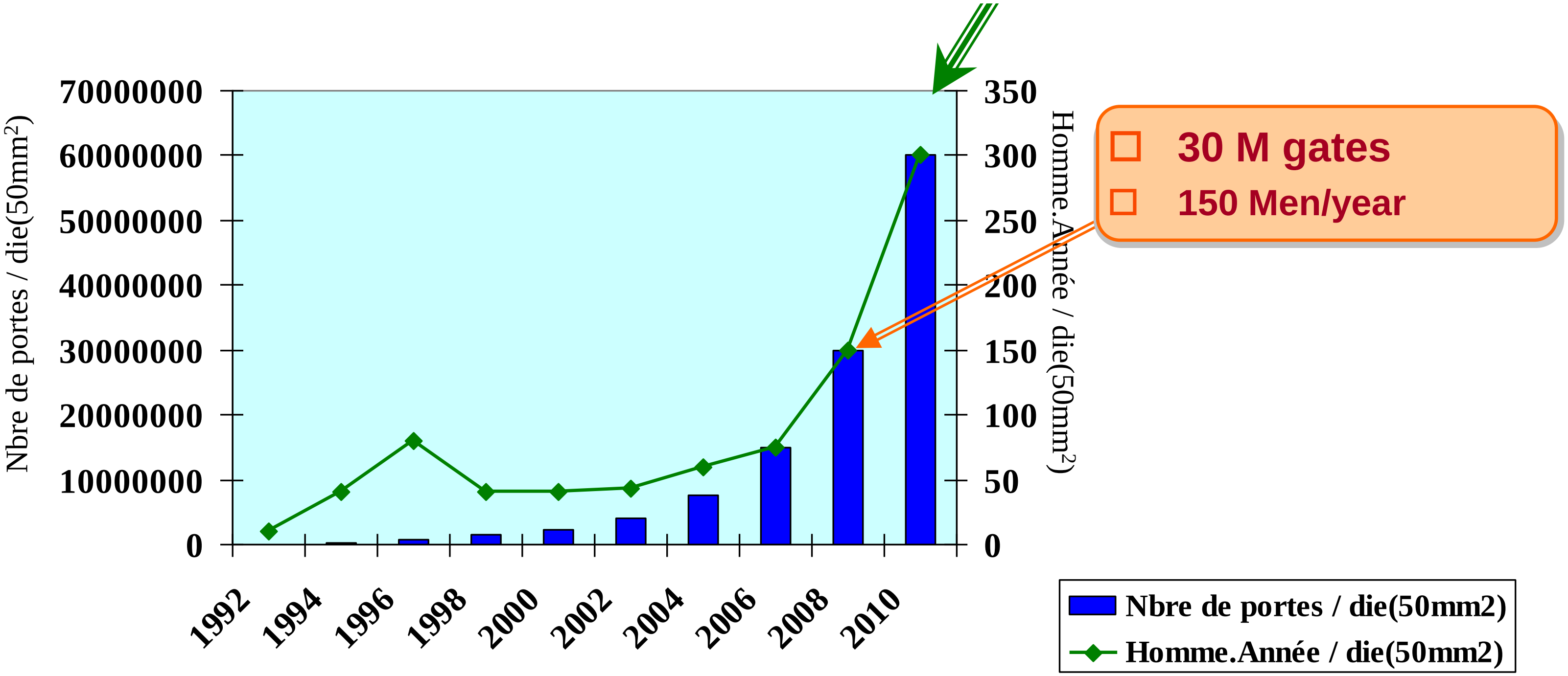
□ Decreasing technology scale

□ Increasing transistor density



□ Designer productivity





□ 30 M gates

□ 150 Men/year

Design methodologies

- **Synthesis and verification automation has always been key factors in the evolution of the design process**
  - Allow to explore the design space efficiently and rapidly
  - Correct by construction design

Design methodologies

Software domain

- Machine code (binary sequence)
- 1950s: concept of assembly language (and assembler)
  - ***based on mnemonics***
  - ***Maurice V. Wilkes de l'université de Cambridge***
- Later: High-level languages and compilers
  - ***1951: First compiler***
    - (A-0 system) par Grace Hopper
  - ***Fortran 1954-1957: First high-level language***
    - FORmula TRANslator
  - ***Cobol 1959, Basic 1964, C 1972, C++ 1983…***

High-level language

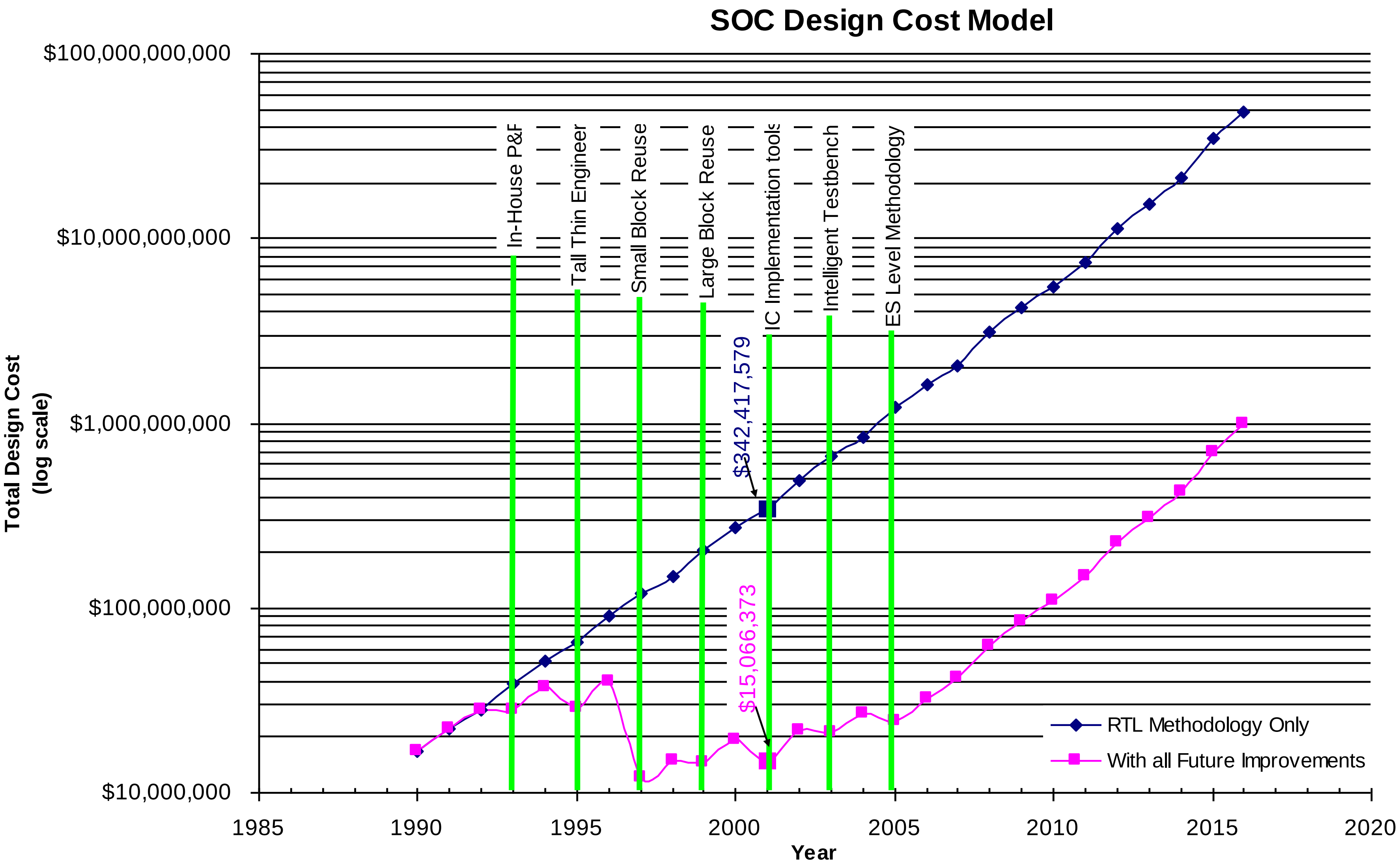
- Platform independent
- Follow the rules of human language
  - ***with a grammar, a syntax and a semantic***
- Provide flexibility and portability
  - ***by hiding details of the computer architecture***

Design methodologies

Hardware domain

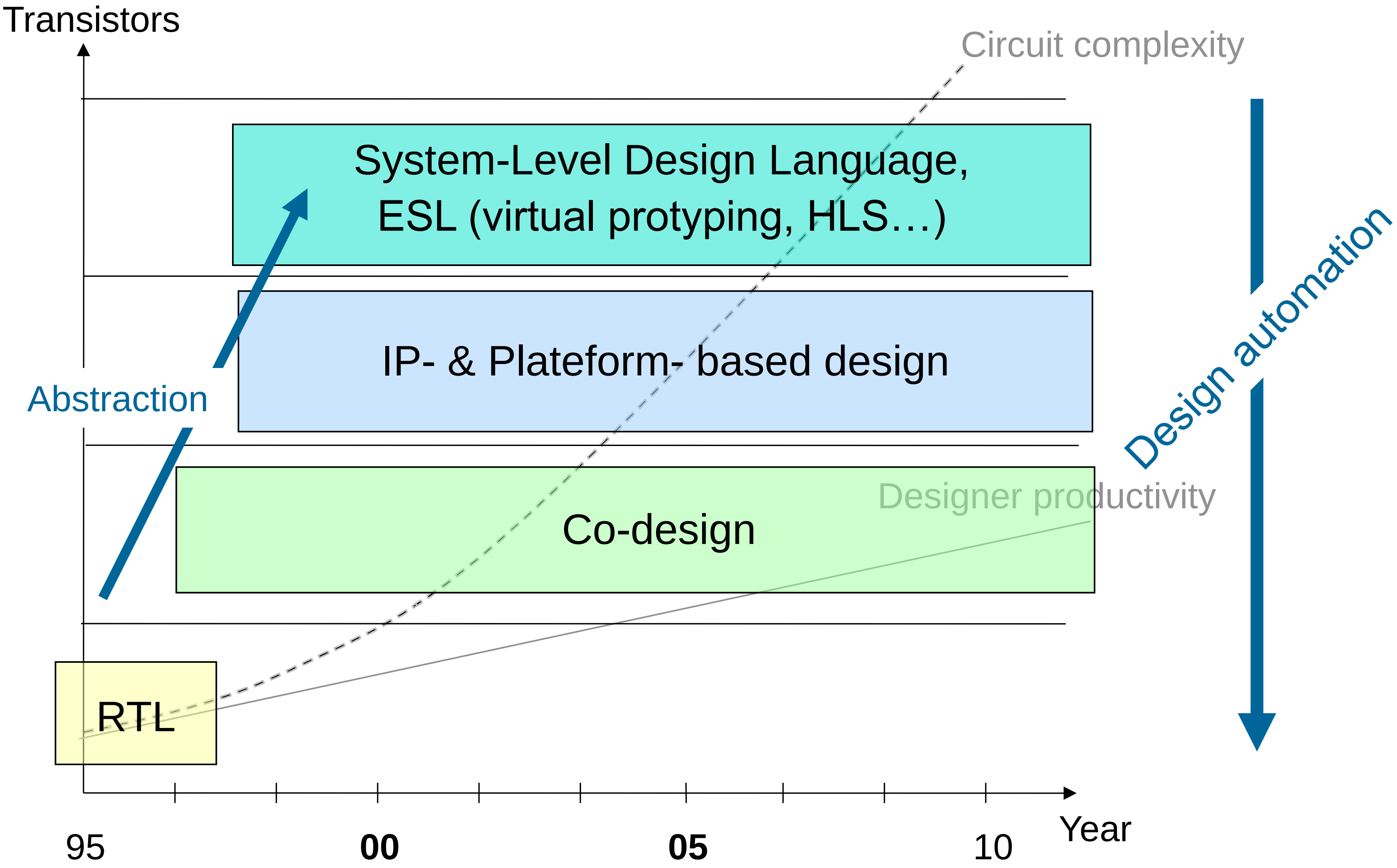
- 1960: IC were done by hand
  - ***designed, optimized and laid out***
- 1970: Gate-level simulation
- end of 70: Cycle-based simulation
- 1980: Wide automation
  - ***place & route, schematic circuit capture, formal verification and static timing analysis***
- Mid 1980: Hardware description language
  - ***1986 Verilog, 1987 VHDL***
- 1990: logic synthesis
  - ***VHDL and Verilog synthesizable subsets***
- Mid 1990:
  - ***High-level synthesis (First gen),***
  - ***Co-design, IP-core reuse…***
- 2000 : Electronic System Level ESL
  - ***System level language***
    - SystemC, SystemVerilog…,
    - Virtual prototyping, Transaction Level Modellin TLM ...

Design gap

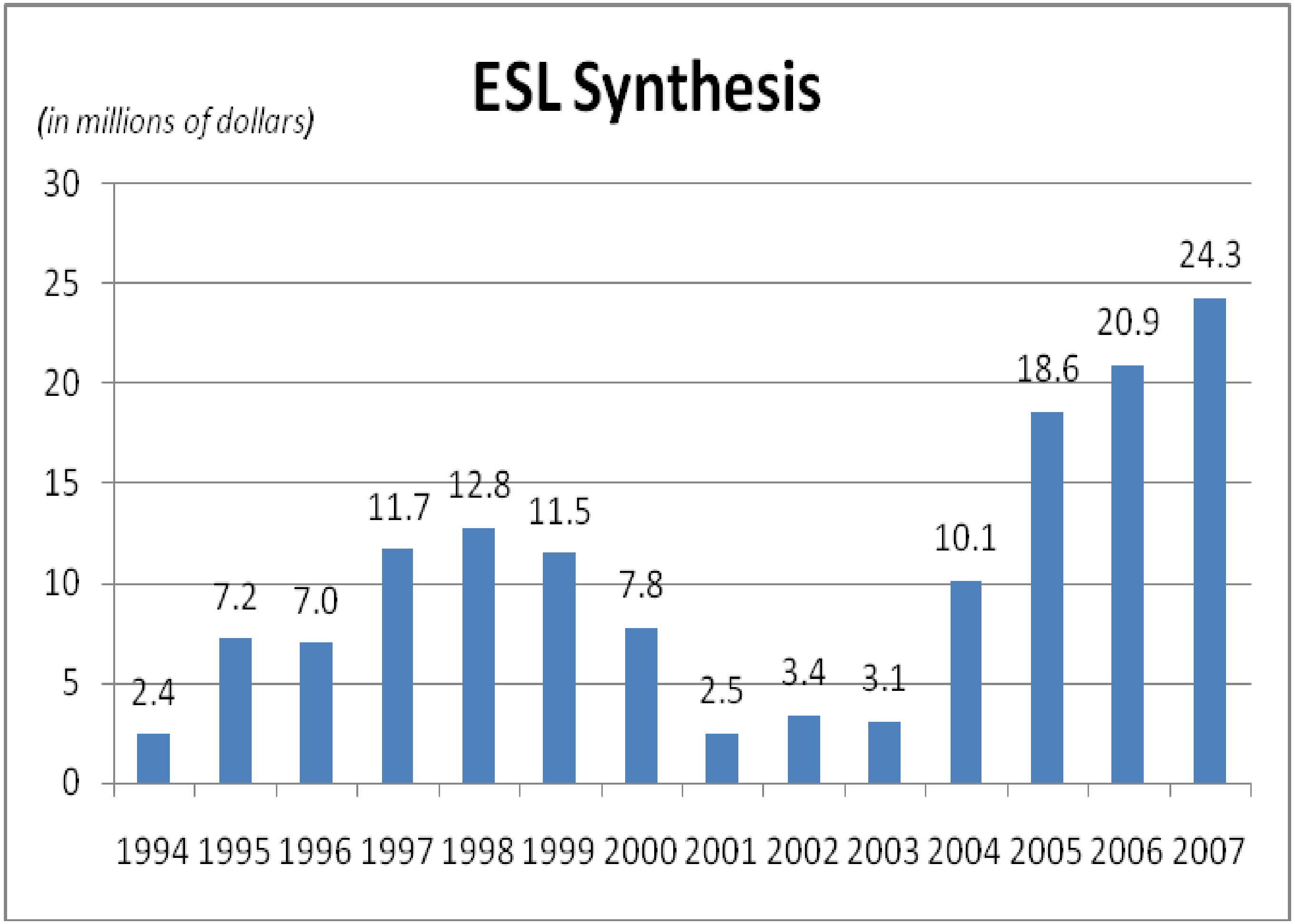


(A. B. Kahng, G. Smith, “A New Design Cost Model”, For the 2001 ITRS)

Electronic System Level Design (ESLD)



ESL Market

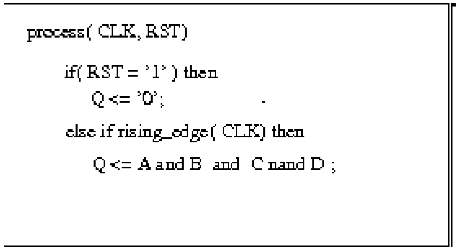

ESL Synthesis
(in millions of dollars)
30
25
20
15
10
5
0
2.4
7.2
7.0
11.7
12.8
11.5
7.8
2.5
3.4
3.1
10.1
18.6
20.9
24.3
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007

Outline

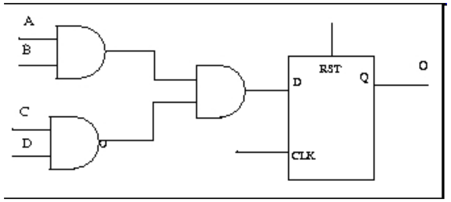
- □ Lab-STICC
- □ General context
- □ **High-Level Synthesis**
  - ■ Brief introduction
  - ■ “In details”
- □ **GAUT**
  - ■ Overview
  - ■ Results
- □ **Conclusion**
- □ **References**

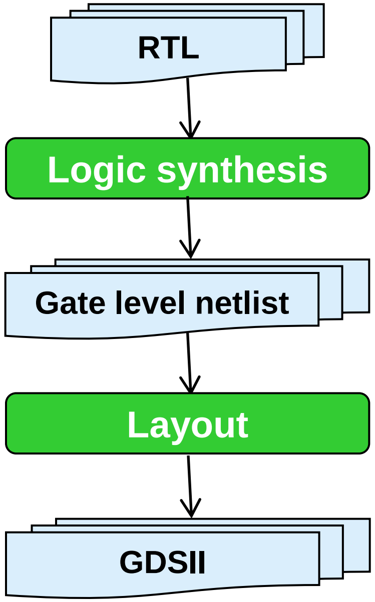
Typical HW design flow

- □ Starting from a Register Transfer Level description, generate an IC layout

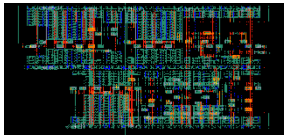


```
process( CLK, RST)
      if( RST = '1' ) then
          Q <= '0';
      else if rising_edge( CLK) then
          Q <= A and B  and  C nand D ;
```



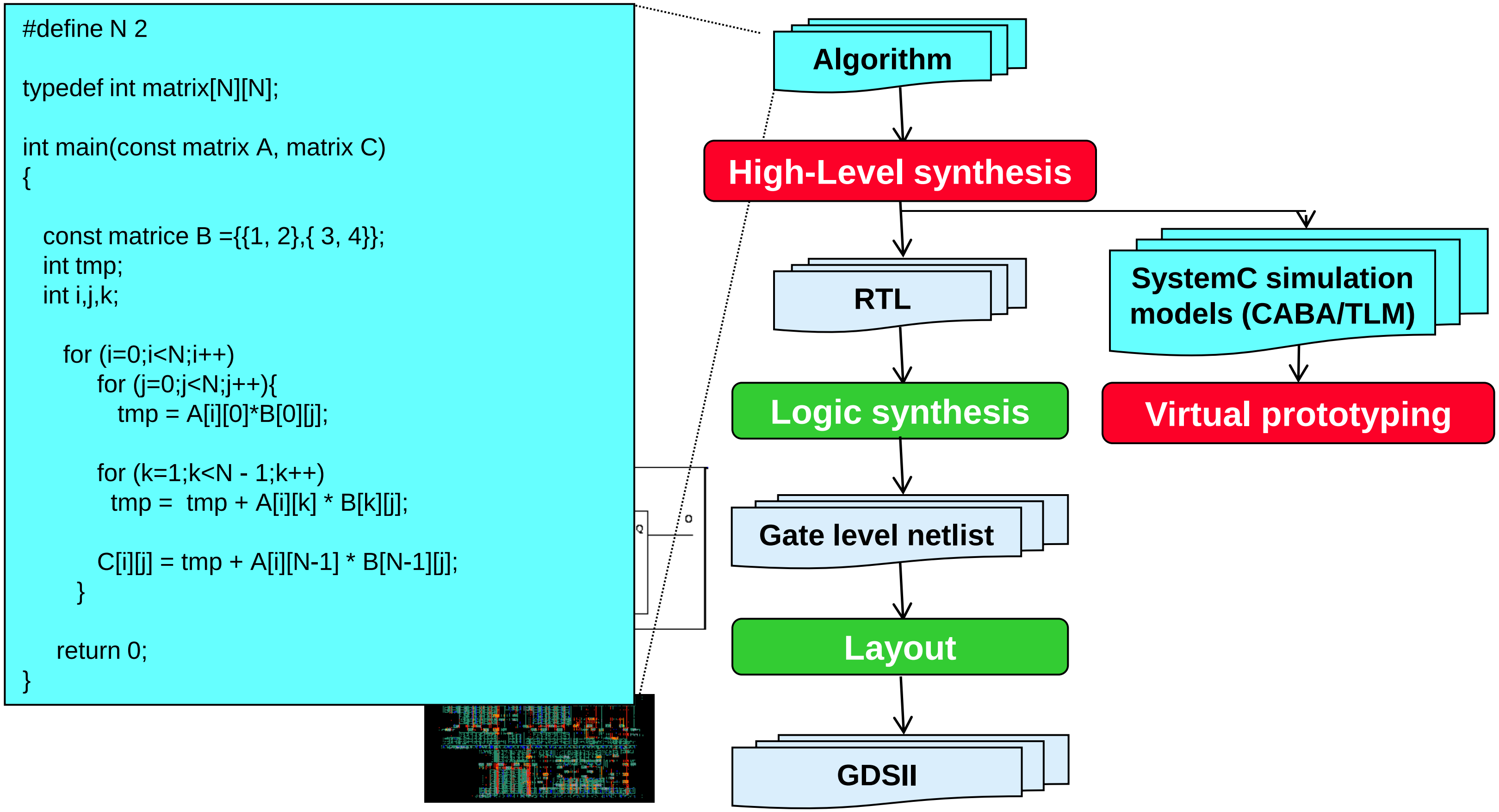


RTL → Logic synthesis → Gate level netlist → Layout → GDSII



Typical HW design flow

- Starting from a functional description, automatically generate an RTL architecture



```
#define N 2

typedef int matrix[N][N];

int main(const matrix A, matrix C)
{

   const matrice B ={{1, 2},{ 3, 4}};
   int tmp;
   int i,j,k;

      for (i=0;i<N;i++)
           for (j=0;j<N;j++){
              tmp = A[i][0]*B[0][j];

           for (k=1;k<N - 1;k++)
             tmp =  tmp + A[i][k] * B[k][j];

           C[i][j] = tmp + A[i][N-1] * B[N-1][j];
        }

     return 0;
}
```

High-level synthesis

- □ **Starting from a functional description, automatically generate an RTL architecture**

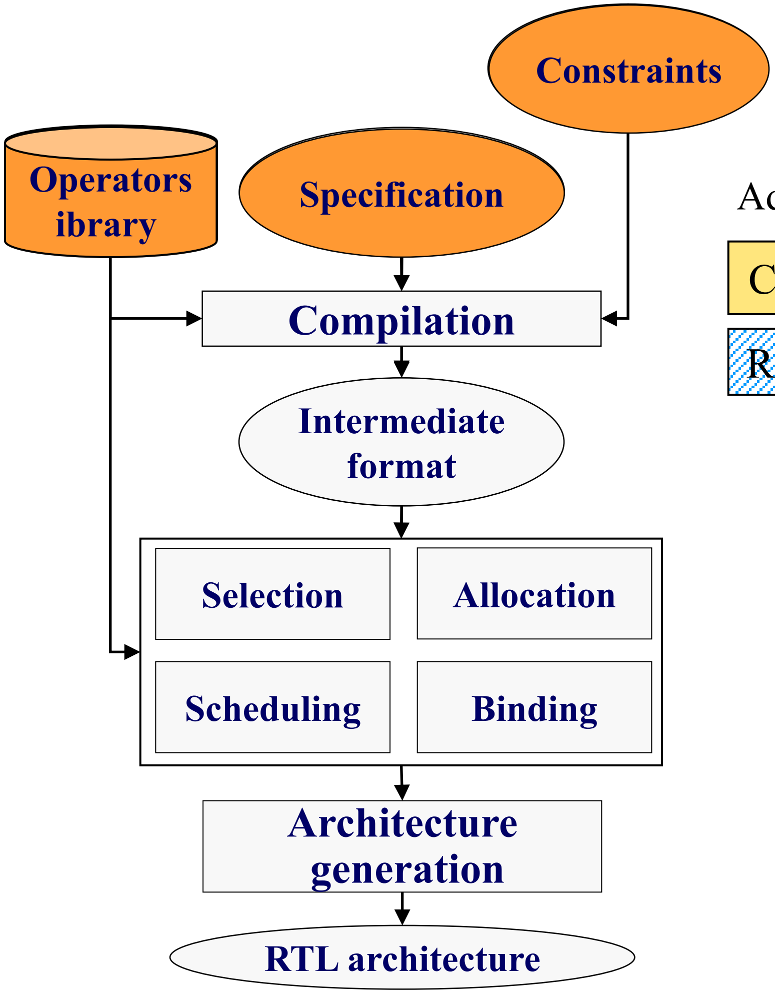
- □ **Constraints**
  - Timing constraints: latency and/or throughput
  - Resource constraints: #Operators and/or #Registers and/or #Memory, #Slices...

- □ **Objectives**
  - Minimization: area i.e. resources, latency, power consumption…
  - Maximization: throughput

Synthesis steps

- □ **Compilation**
  - Generates a formal modeling of the specification
- □ **Selection**
  - Chooses the architecture of the operators
- □ **Allocation**
  - Defines the number of operators for each selected type
- □ **Scheduling**
  - Defines the execution date of each operation
- □ **Binding (or Assignment)**
  - Defines which operator will execute a given operation
  - Defines which memory element will store a data
- □ **Architecture generation**
  - Writes out the RTL source code in the target language e.g. VHDL

HLS steps: inputs



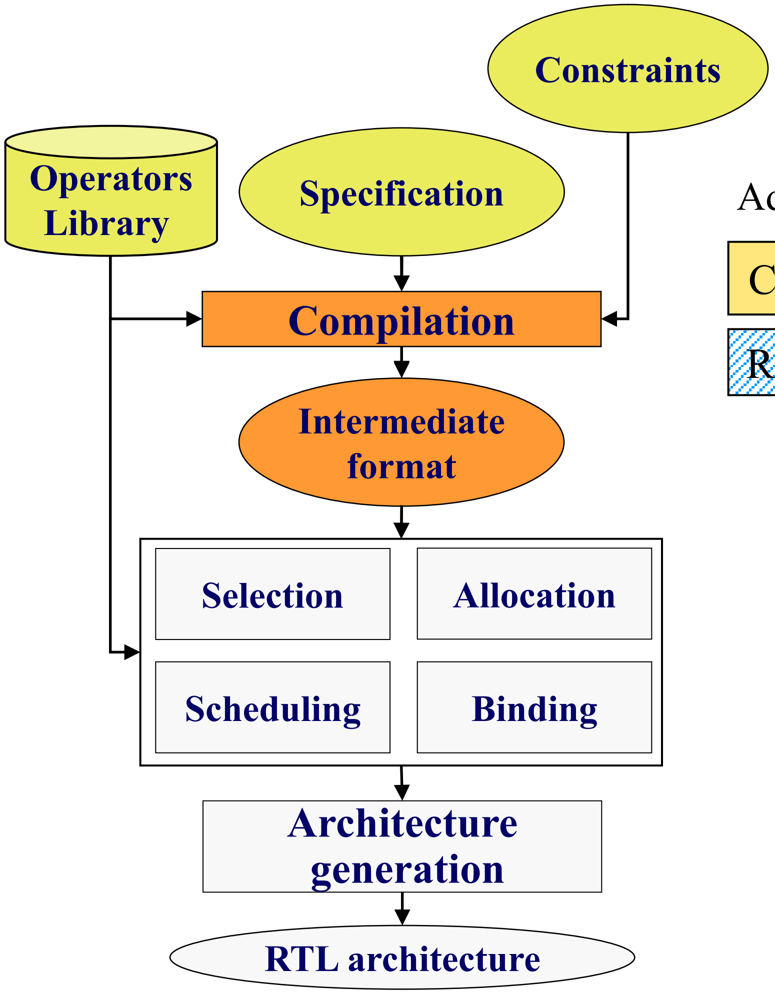
Operators library

Adders	multipliers	subtractors
CLA	Booth	CLA
RCA	Wallace	RCA

Specification

$$O = ((n_{01}+n_{02})*n_{12})-(n_{21}+n_{22})$$

HLS steps: Compilation



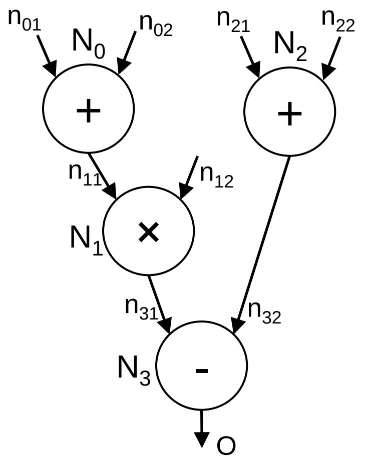
Operators library

Adders	multipliers	subtractors
CLA	Booth	CLA
RCA	Wallace	RCA

Specification

$O = ((n_{01}+n_{02})*n_{12})-(n_{21}+n_{22})$

Intermediate representation



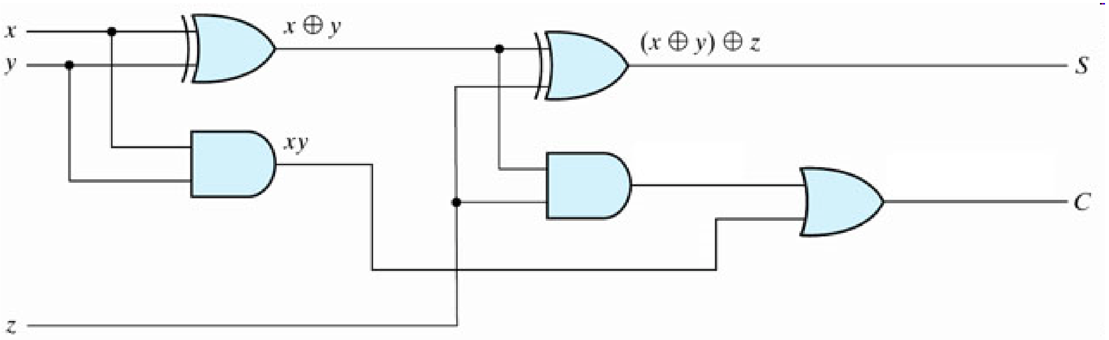
Synthesis steps

- □ **Compilation**
  - Generates a formal modeling of the specification
- □ **Selection**
  - Chooses the architecture of the operators
- □ **Allocation**
  - Defines the number of operators for each selected type
- □ **Scheduling**
  - Defines the execution date of each operation
- □ **Binding (or Assignment)**
  - Defines which operator will execute a given operation
  - Defines which memory element will store a data
- □ **Architecture generation**
  - Writes out the RTL source code in the target language e.g. VHDL

Operator architecture

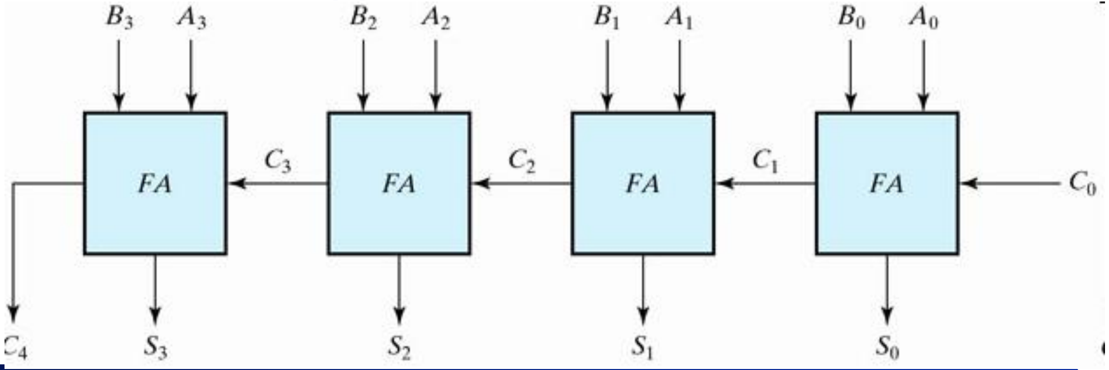
□ Full 1-bit adder : X + Y + Z

- X, Y are the operands
- Z is the input carry



□ Ripple Carry Adder

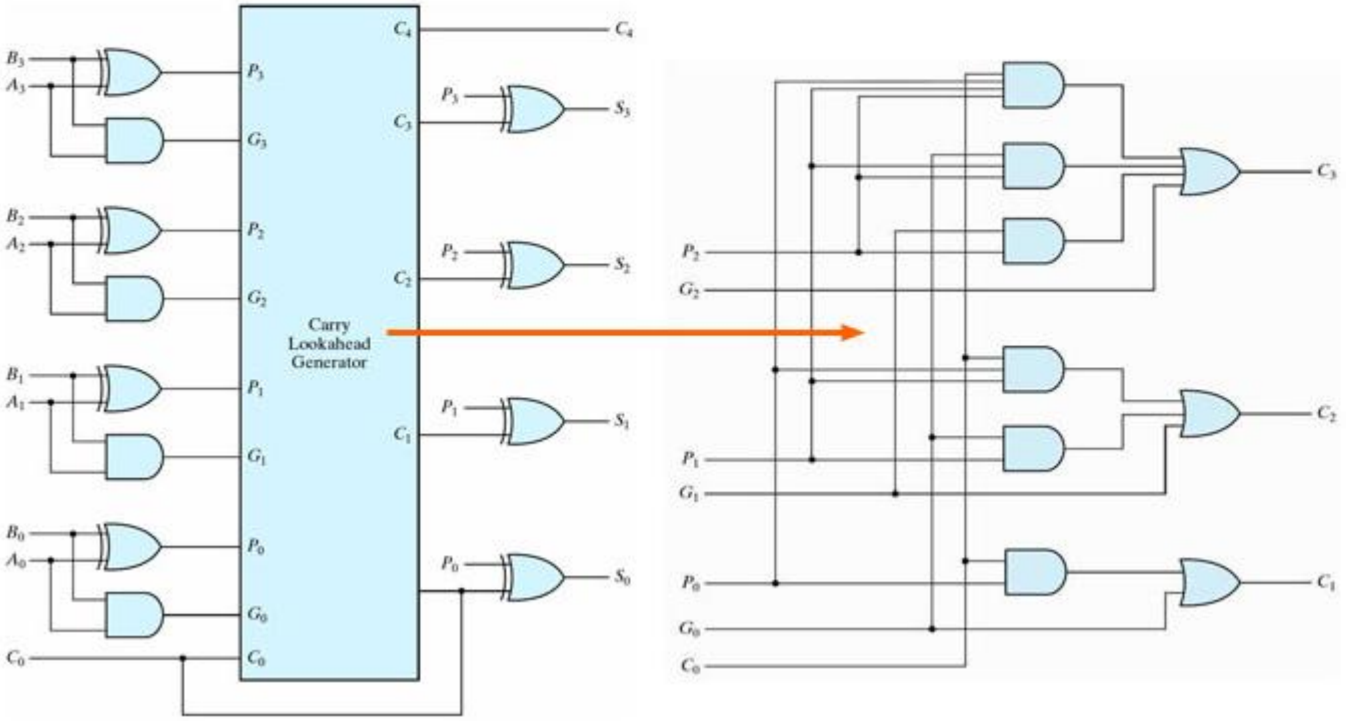
- Add two integers A and B
- Cascade of 1-bit adders => Ripple-Carry Adder



Operator architecture

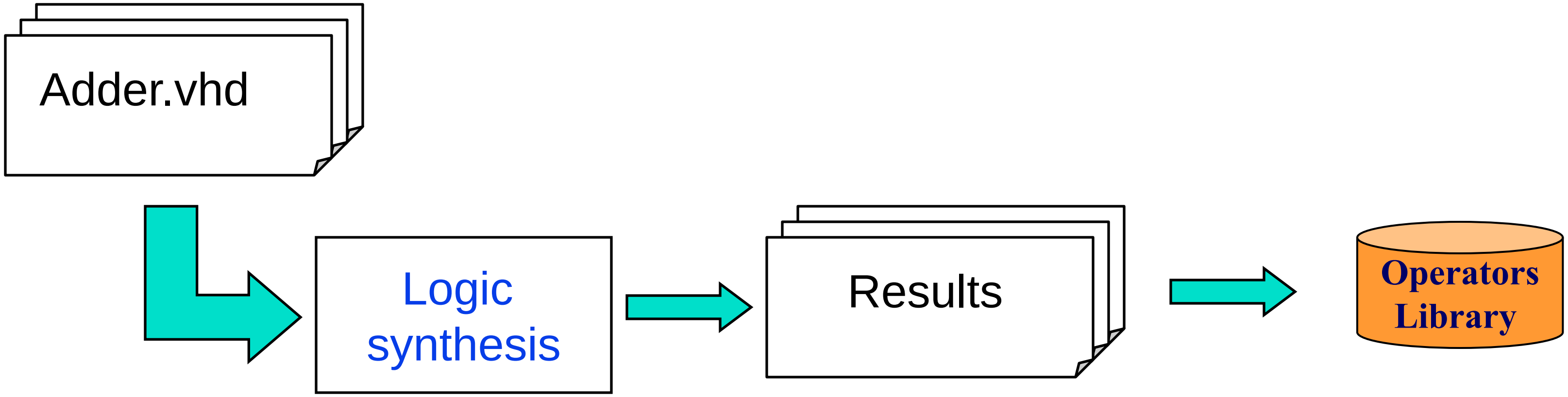
□ Carry Look-ahead adder CLA

- Uses a carry generator to compute all the carries concurrently
  - ❑ *faster but also larger than the RCA*

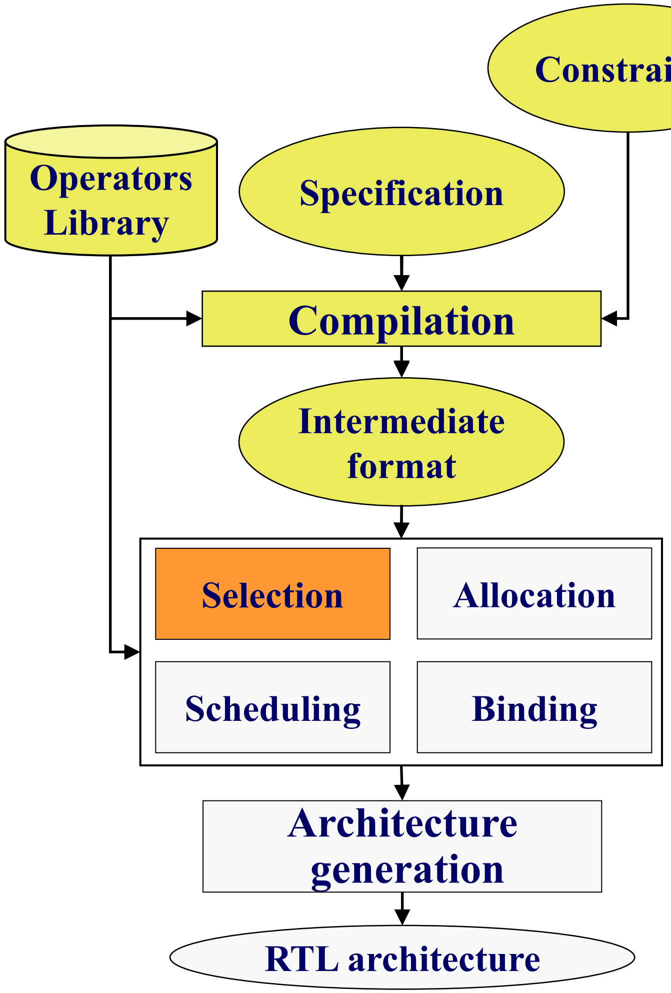


Library characterization

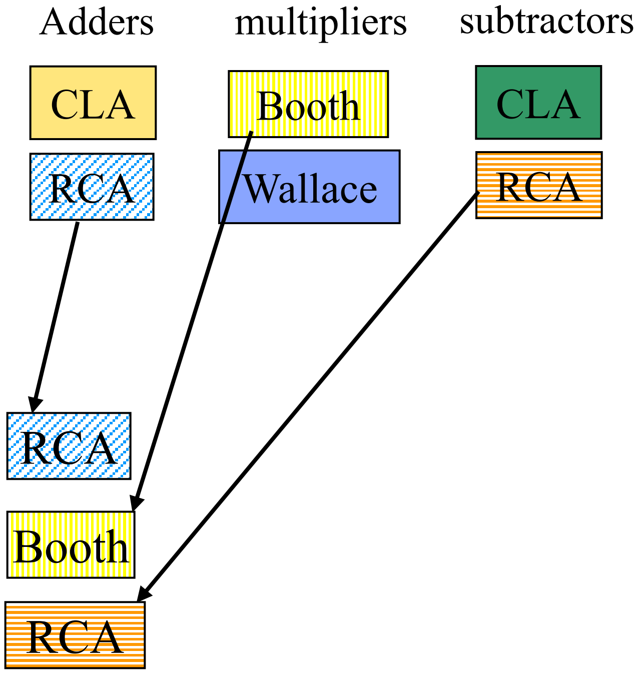
- □ RTL architecture produced by HLS depends on the capabilities and characteristics of the operators
- □ Library processing reads the available libraries and determines the functional, timing, and area characteristics of the available parts.



HLS steps: Selection



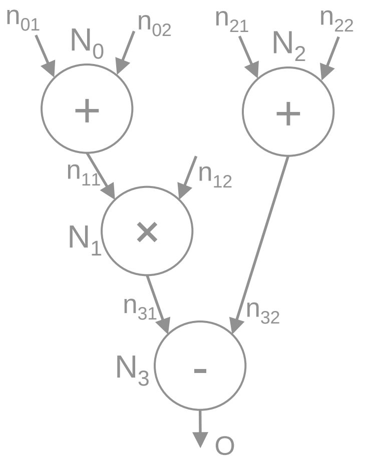
Operators library



Specification

$O = ((n_{01}+n_{02})*n_{12})-(n_{21}+n_{22})$

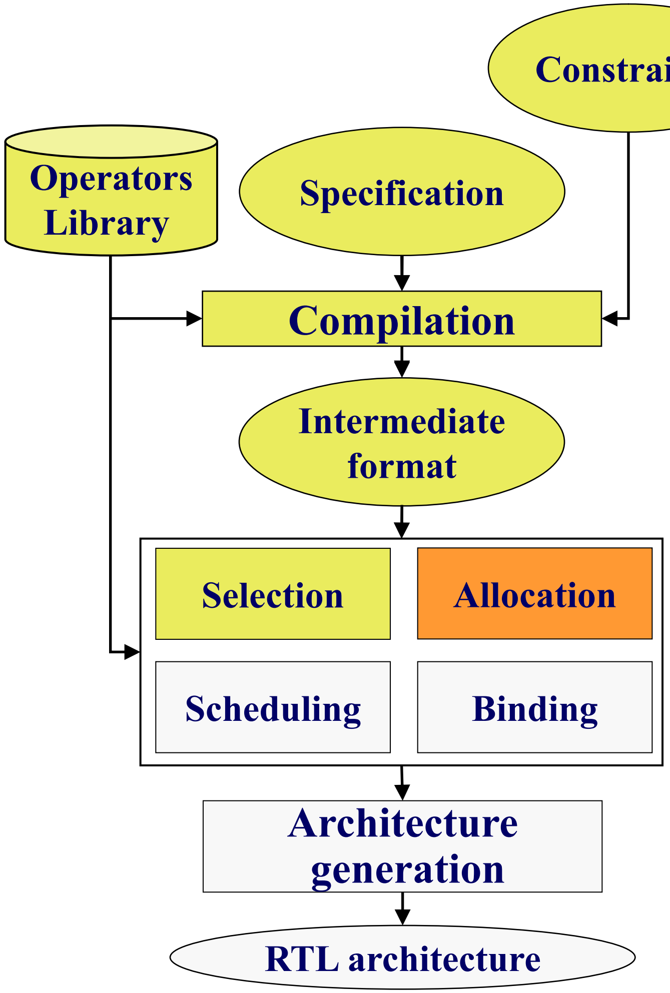
Intermediate representation



Synthesis steps

- □ **Compilation**
  - Generates a formal modeling of the specification
- □ **Selection**
  - Chooses the architecture of the operators
- □ **Allocation**
  - Defines the number of operators for each selected type
- □ **Scheduling**
  - Defines the execution date of each operation
- □ **Binding (or Assignment)**
  - Defines which operator will execute a given operation
  - Defines which memory element will store a data
- □ **Architecture generation**
  - Writes out the RTL source code in the target language e.g. VHDL

HLS steps: allocation



Operators library

Adders	multipliers	subtractors
CLA	Booth	CL
RCA	Wallace	RCA

RCA *1

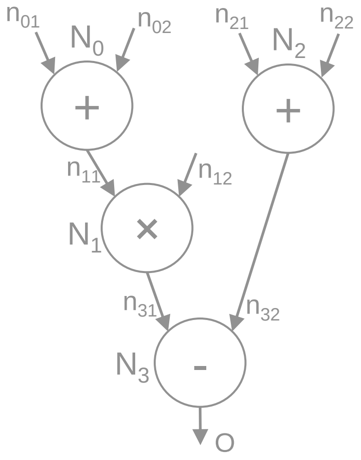
Booth *1

RCA *1

Specification

$$O = ((n_{01}+n_{02})*n_{12})-(n_{21}+n_{22})$$

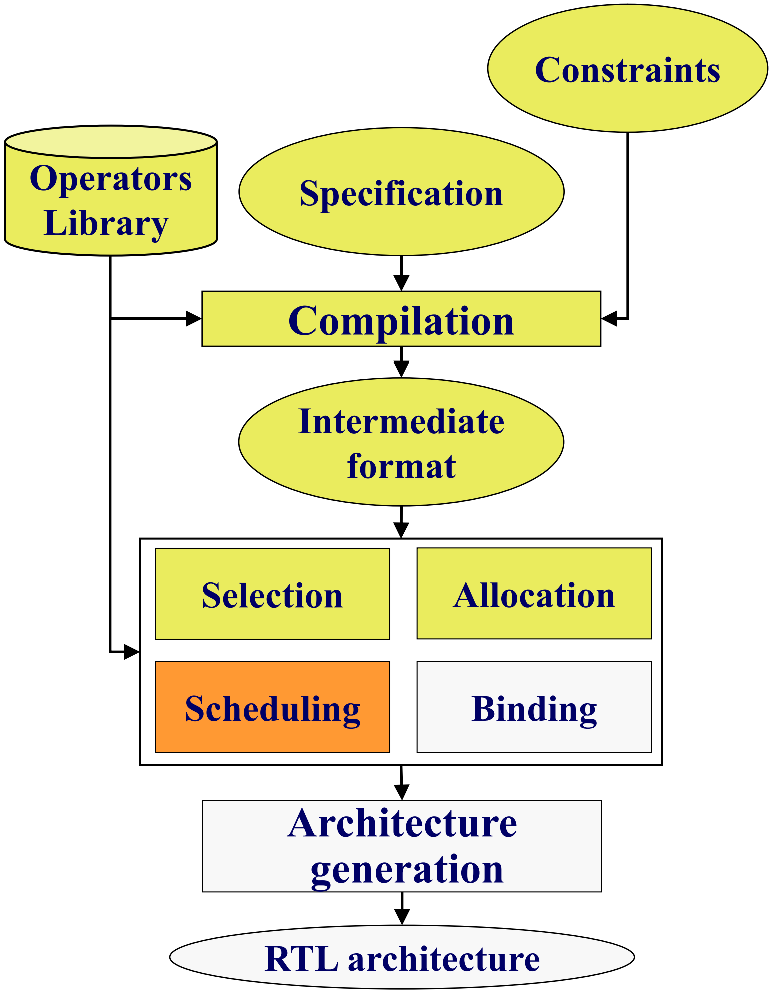
Intermediate representation

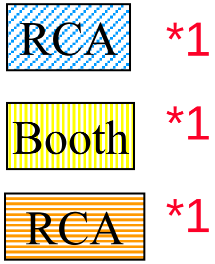


Synthesis steps

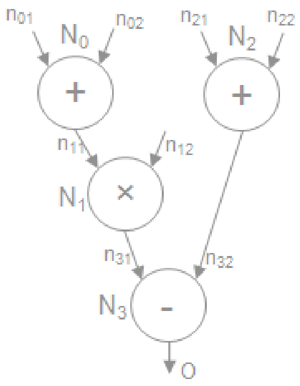
- □ **Compilation**
  - ■ Generates a formal modeling of the specification
- □ **Selection**
  - ■ Chooses the architecture of the operators
- □ **Allocation**
  - ■ Defines the number of operators for each selected type
- □ **Scheduling**
  - ■ Defines the execution date of each operation
- □ **Binding (or Assignment)**
  - ■ Defines which operator will execute a given operation
  - ■ Defines which memory element will store a data
- □ **Architecture generation**
  - ■ Writes out the RTL source code in the target language e.g. VHDL

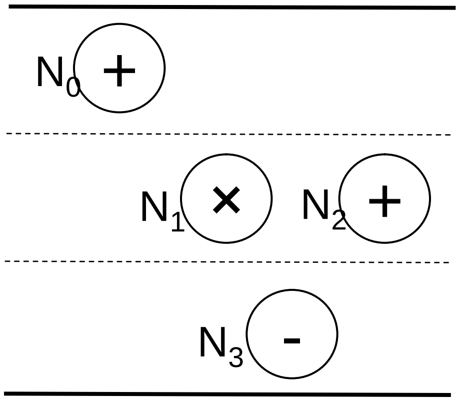
HLS steps: scheduling





Intermediate representation

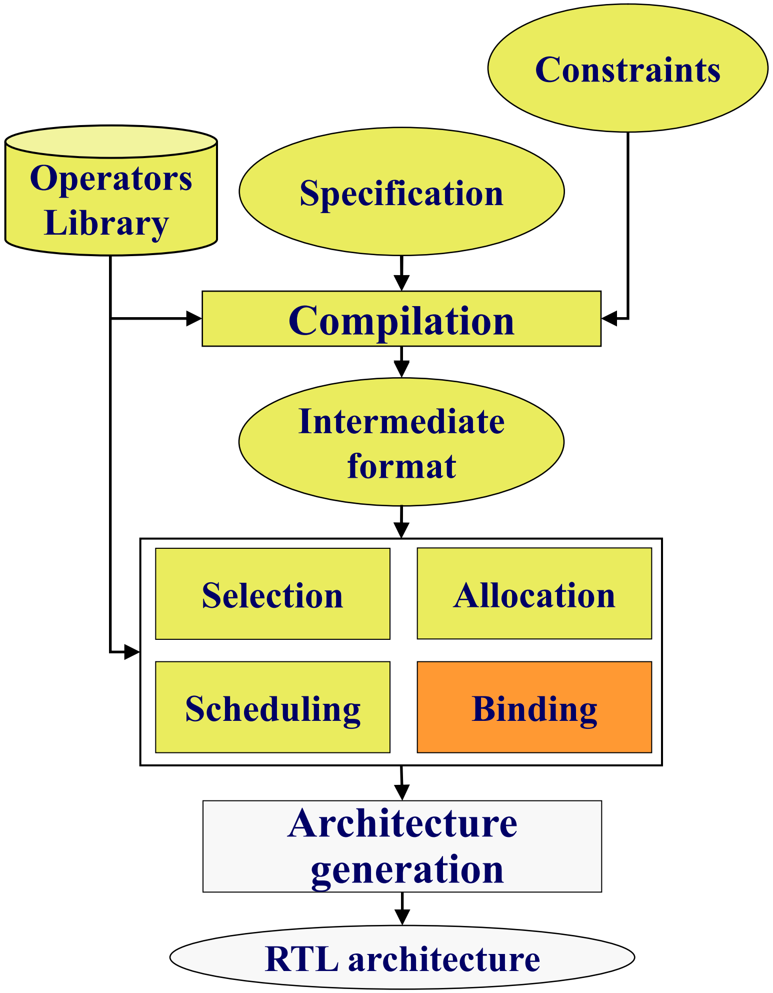




Synthesis steps

- □ **Compilation**
  - Generates a formal modeling of the specification
- □ **Selection**
  - Chooses the architecture of the operators
- □ **Allocation**
  - Defines the number of operators for each selected type
- □ **Scheduling**
  - Defines the execution date of each operation
- □ **Binding (or Assignment)**
  - Defines which operator will execute a given operation
  - Defines which memory element will store a data
- □ **Architecture generation**
  - Writes out the RTL source code in the target language e.g. VHDL

HLS steps: binding


Constraints

Operators Library

Specification

Compilation

Intermediate format

Selection

Allocation

Scheduling

Binding

Architecture generation

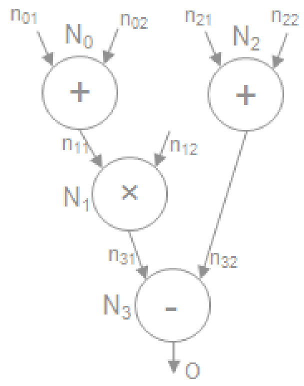
RTL architecture

RCA *1

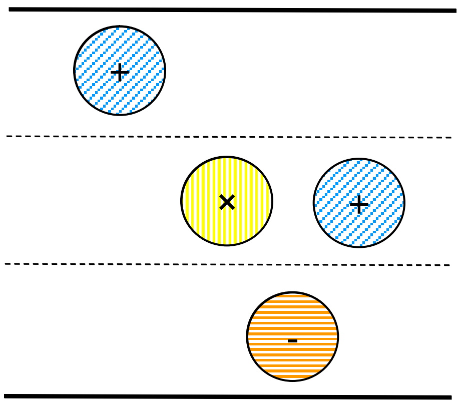
Booth *1

RCA *1

Intermediate representation



Operation binding



Data Binding

$n_{01} \rightarrow R_1$

$n_{02} \rightarrow R_2$

$n_{21}, n_{11} \rightarrow R_3$

$n_{22}, n_{12} \rightarrow R_4$

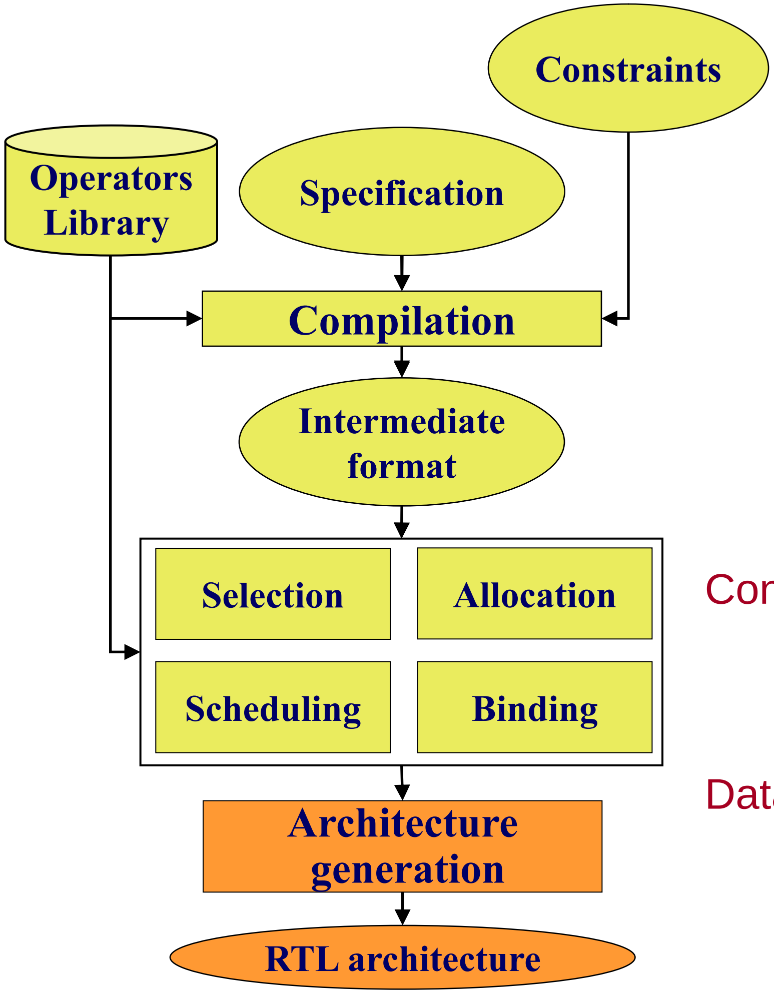
$n_{31} \rightarrow R_5$

$n_{32} \rightarrow R_6$

Synthesis steps

- □ **Compilation**
  - Generates a formal modeling of the specification
- □ **Selection**
  - Chooses the architecture of the operators
- □ **Allocation**
  - Defines the number of operators for each selected type
- □ **Scheduling**
  - Defines the execution date of each operation
- □ **Binding (or Assignment)**
  - Defines which operator will execute a given operation
  - Defines which memory element will store a data
- □ **Architecture generation**
  - Writes out the RTL source code in the target language e.g. VHDL

HLS steps: output


Constraints

Operators Library

Specification

Compilation

Intermediate format

Selection

Allocation

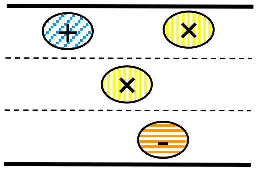
Scheduling

Binding

Architecture generation

RTL architecture

Operation binding



Data binding

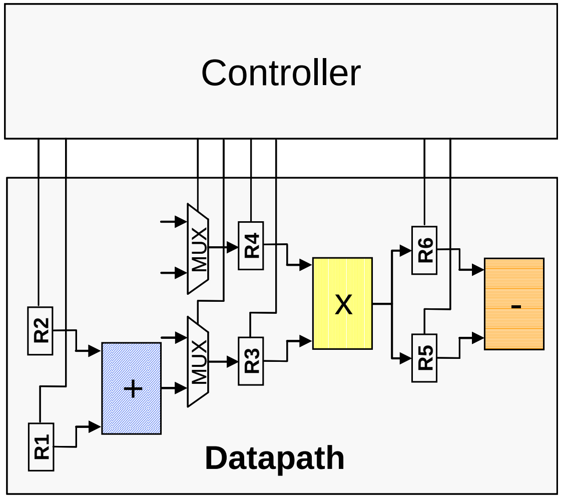
n_{01}	→ R_1
n_{02}	→ R_2
n_{21}, n_{11}	→ R_3
n_{22}, n_{12}	→ R_4
n_{31}	→ R_5
n_{32}	→ R_6

Controller

- FSM controller
- Programmable controller

Datapath components

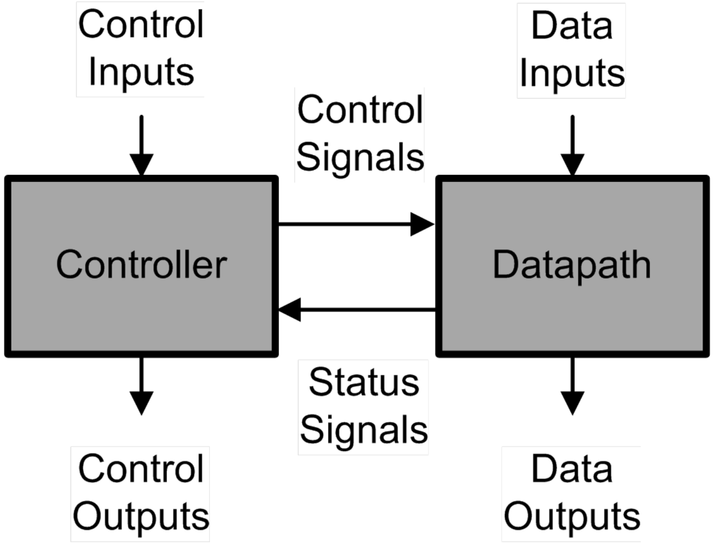
- Storage components
- Functional units
- Connection components


Controller

Datapath

RTL Architecture

- Controller
  - FSM controller
  - Programmable controller
- Datapath components
  - Storage components
  - Functional units
  - Connection components

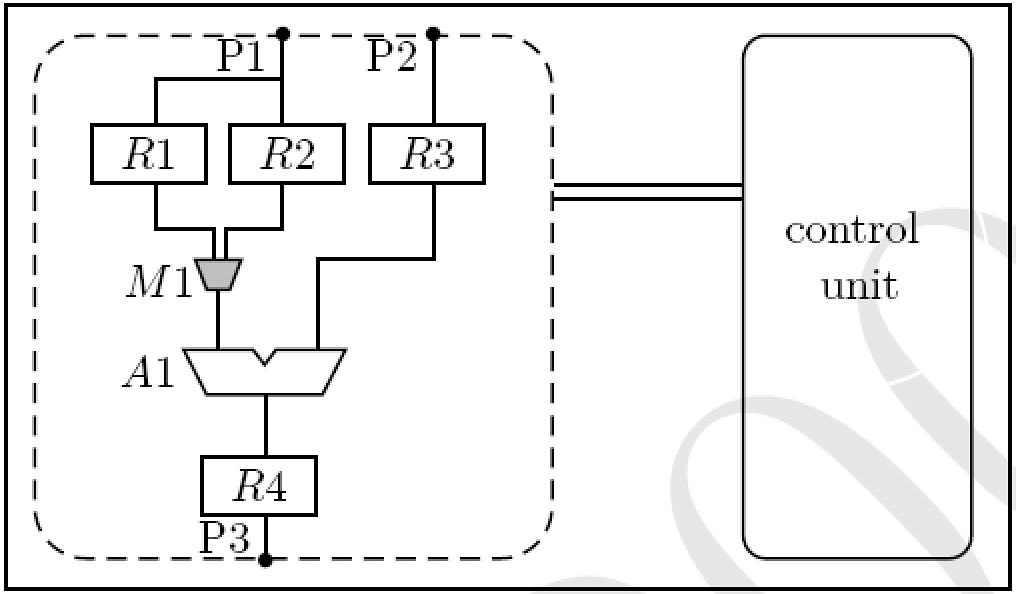


Source :
Embedded System Design, © 2009, Gajski, Abdi, Gerstlauer, Schirner

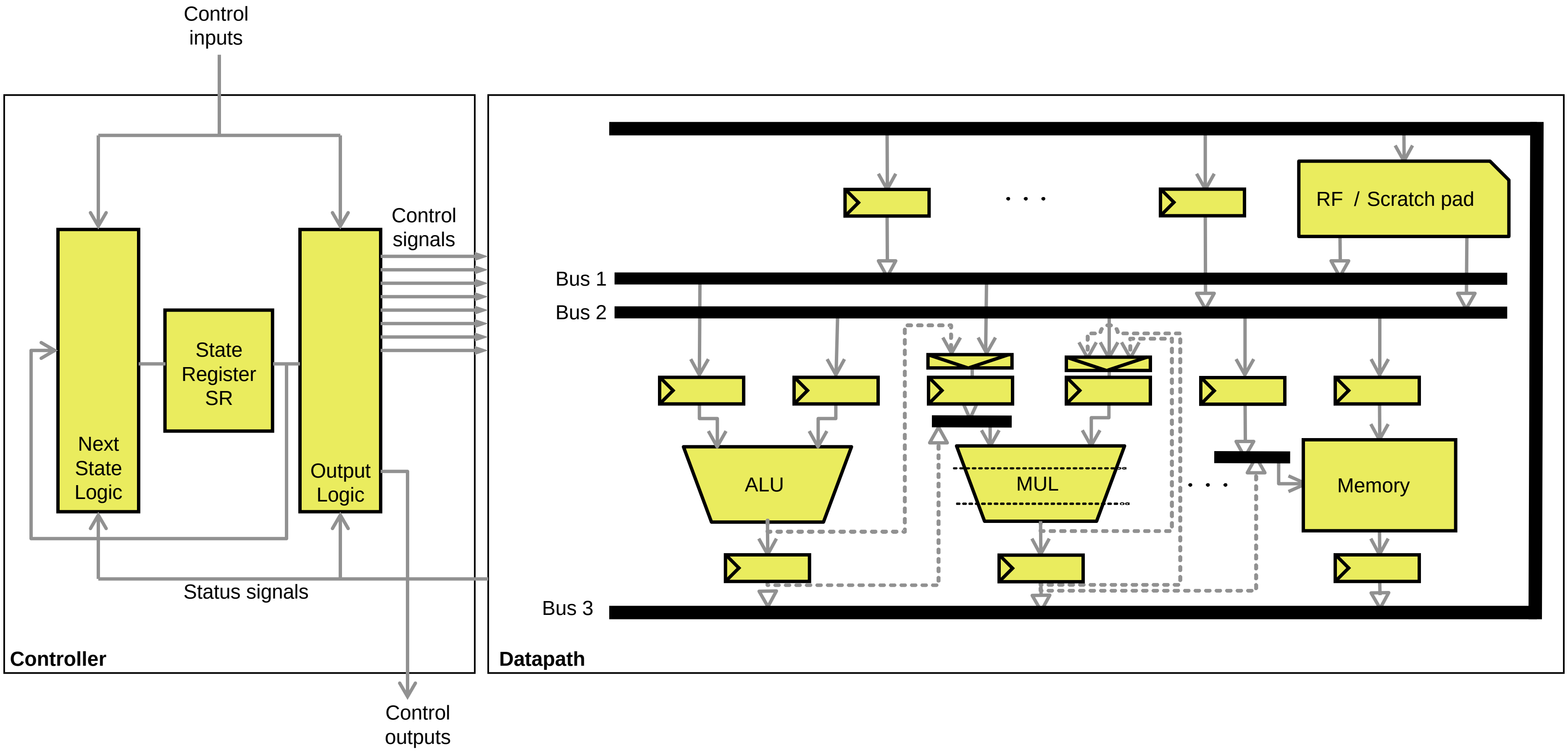
Example

☐ This architecture performs the following operations:

- store two variables coming from the port P1 in R1 and R2
- store one variable coming from the port P2 in R3
- add the variables stored in R1 and R3 and put the result in R4
- add the variables stored in R2 and R3 and put the result in R4
- connect either R1 or R2 to A1
  - ❑ ***the control unit manages this connection through M1***



RTL architecture



Source :
Embedded System Design, © 2009, Gajski, Abdi, Gerstlauer, Schirner

Problem examples and design flow

Resource constrained HLS

- **Limited number of resources**
  - e.g.: 2 multipliers, 3 adders
  - Pseudo architecture
- **Schedule operations according to the available operators in the current control step**
- **Objectives**
  - Minimize the latency or maximize the throughput
    - ***based on operations mobility i.e. operations urgency***

Resource constrained HLS

- □ **Limited number of resources**
  - ■ e.g.: 2 multipliers, 3 adders
  - ■ Pseudo architecture
- □ **Schedule operations according to the available operators in the current control step**
- □ **Objectives**
  - ■ Minimize the latency or maximize the throughput
    - ❑ ***based on operations mobility i.e. operations urgency***

⟹ **Allocation and then Scheduling**

Time constrained HLS

- □ **Latency constraint**
  - e.g. 5 clock cycles to process all the data
- □ **Throughput constraint**
  - Cadency, initiation interval…
  - e.g. process each 5 cycles a new set of input data
- □ **Schedule operations by using operators as much as needed**
- □ **Objective**
  - Minimize the circuit area

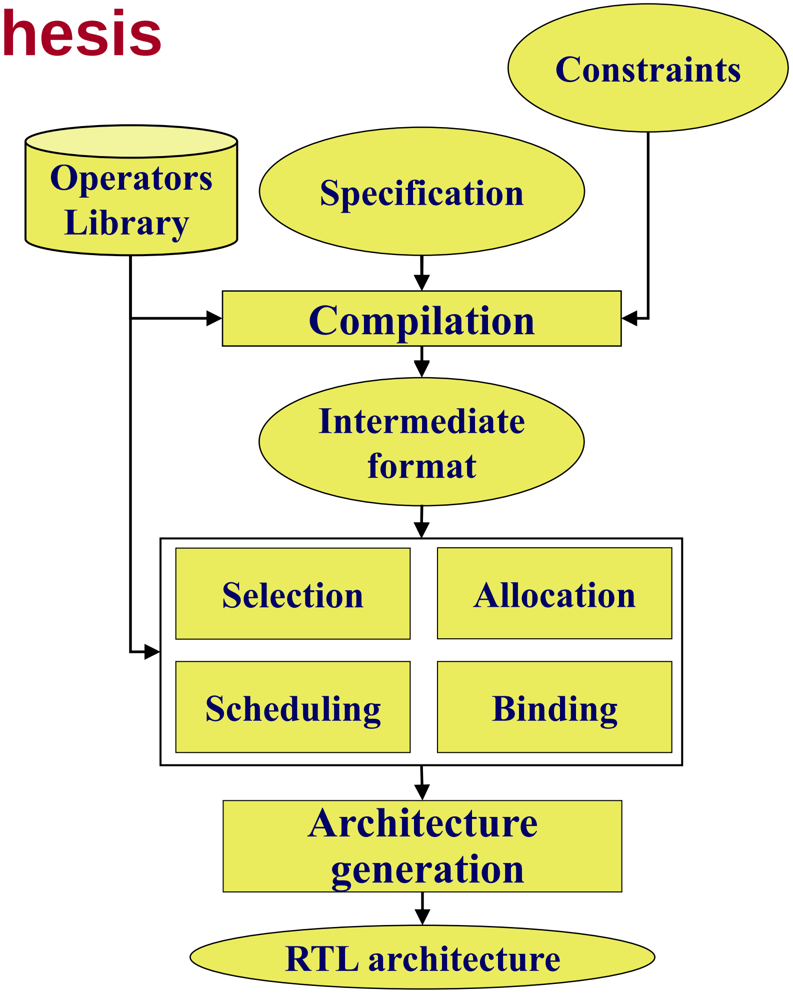
Time constrained HLS

- □ **Latency constraint**
  - e.g. 5 clock cycles to process all the data
- □ **Throughput constraint**
  - Cadency, initiation interval…
  - e.g. process each 5 cycles a new set of input data
- □ **Schedule operations by using operators as much as needed**
- □ **Objective**
  - Minimize the circuit area

⇒ **Scheduling and then Allocation**

Design flows

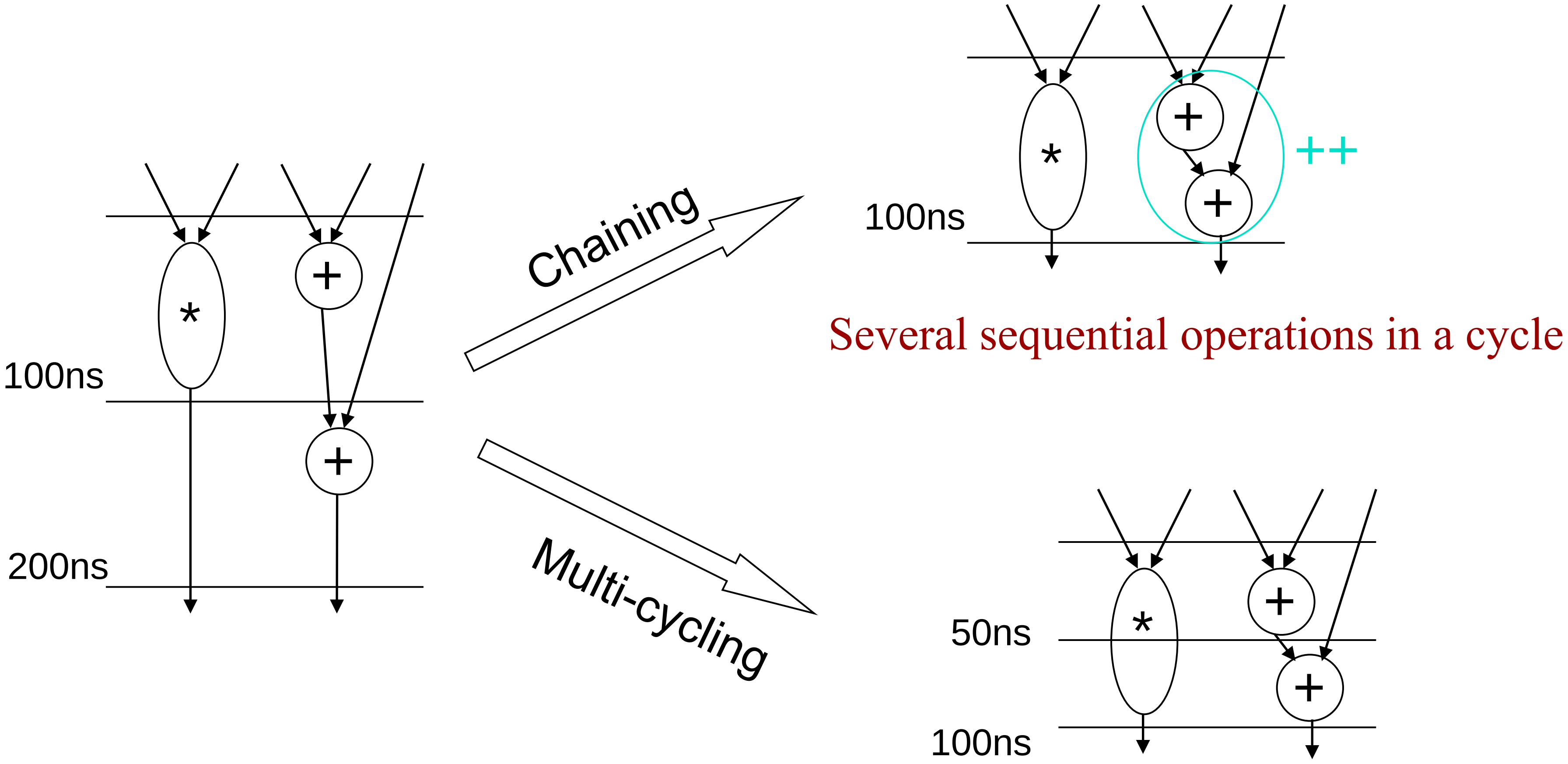
- □ **No unique design flow i.e. synthesis steps order**
  - Allocation → Scheduling
    - □ ***resource constrained***
  - Scheduling → Allocation
    - □ ***Time constrained***
  - Scheduling → Binding
  - Binding → Scheduling
  - Scheduling & Binding
  - ...



And a lot of other problems...

- Variable merging Storage Sharing
- Operation merging Operator sharing
- **Connection merging**
  - Bus sharing
- **Register merging**
  - Register file...
- **Chaining**
  - Several sequential operations in a cycle
- **Multi-cycling**
  - One operation takes more than one clock cycle to execute
- **Pipelining**
  - Pipelined Datapath, pipelined operator, pipelined controller
- **...**

Chaining, multi-cycling



Several sequential operations in a cycle

One operation takes more than one clock cycle to execute

Outline

- □ Lab-STICC
- □ General context
- □ **High-Level Synthesis**
  - Brief introduction
  - “In details”
- □ **GAUT**
  - Overview
  - Results
- □ **Conclusion**
- □ **References**

Synthesis steps

- □ **Compilation**
  - Generates a formal modeling of the specification
- □ **Selection**
  - Chooses the architecture of the operators
- □ **Allocation**
  - Defines the number of operators for each selected type
- □ **Scheduling**
  - Defines the execution date of each operation
- □ **Binding (or Assignment)**
  - Defines which operator will execute a given operation
  - Defines which memory element will store a data
- □ **Architecture generation**
  - Writes out the RTL source code in the target language e.g. VHDL

High-level synthesis goal

- **Starting from a functional description, automatically generate an RTL architecture**
  - Mathematic formula
  - Matlab/Simulink
  - C/C++/SystemC
  - …

Synthesizable models

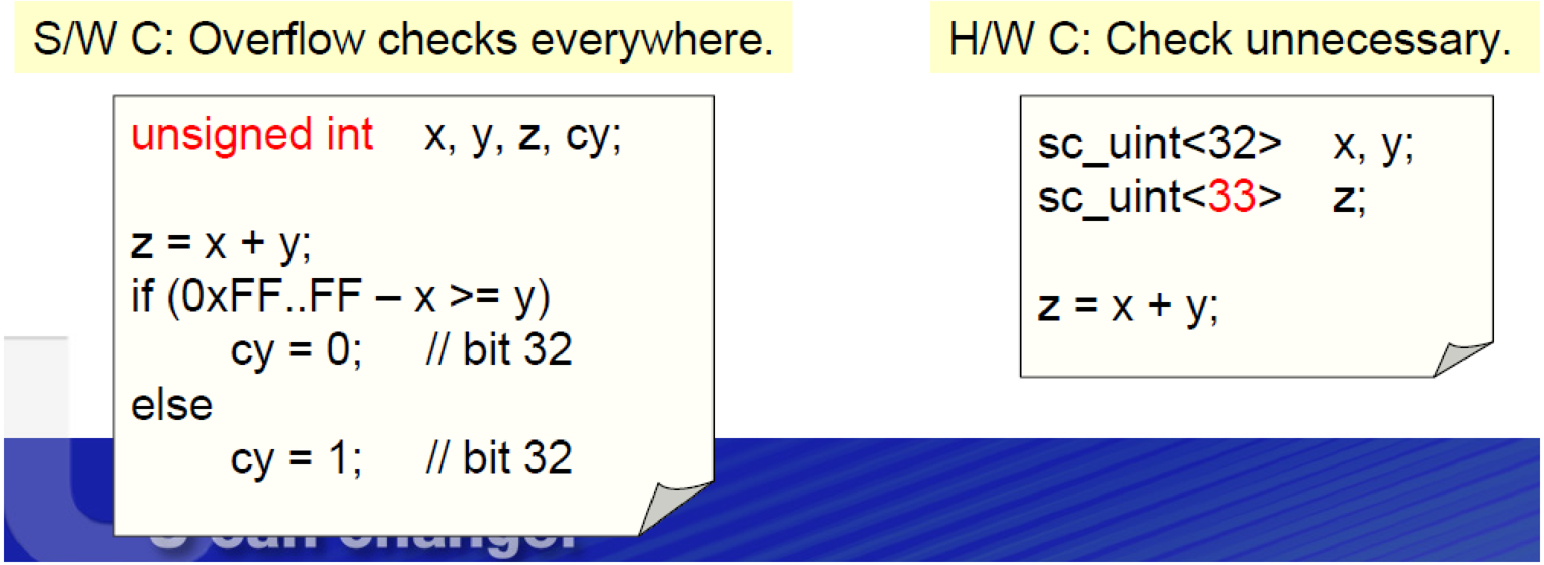
C for the synthesis:

- No pointer
  - ***Statically unresolved***
  - ***Arrays are allowed!***
- No standard function call
  - ***printf, scanf, fopen, malloc…***
- Function calls are allowed
  - ***Can be in-lined or not***
- Finite precision
  - ***Bit accurate integers, fixed point, signed, unsigned…***
  - ***Based on SystemC or Mentor Graphics data types***
    - sc_int, sc_fixed
    - ac_int, ac_fixed

Synthesizable models

□ C for the synthesis:

- Finite precision
  - ❑ ***bit accurate integer, fixed point, signed, unsigned…***



S/W C: Overflow checks everywhere.

```
unsigned int    x, y, z, cy;

z = x + y;
if (0xFF..FF – x >= y)
        cy = 0;     // bit 32
else
        cy = 1;     // bit 32
```

H/W C: Check unnecessary.

```
sc_uint<32>    x, y;
sc_uint<33>    z;

z = x + y;
```

Purely functional Example #1: a simple C code

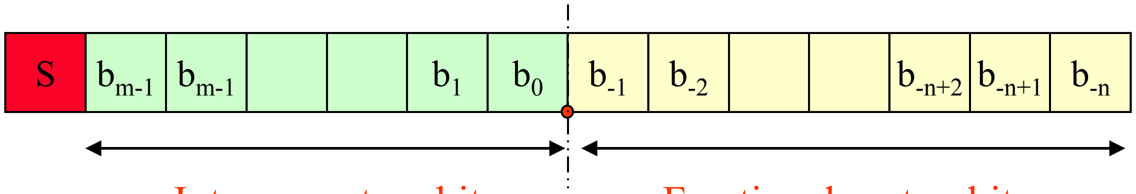
```
#define N 16

int main(int data_in, int *data_out)
{ static const int Coeffs [N] = {98,-39,-327,439,950,-2097,-1674,9883,9883,-1674,-2097,950,439,-327,-39,98};

  int Values[N];
  int temp;
  int sample,i,j;

   sample = data_in;
   temp = sample * Coeffs[N-1];

   for(i = 1; i<=(N-1); i++){
              temp += Values[i] * Coeffs[N-i-1];
    }

   for(j=(N-1); j>=2; j-=1 ){
       Values[j] = Values[j-1];
   }

   Values[1] = sample;
   *data_out=temp;

   return 0;
}
```

Purely functional example #2: bit accurate C++ code

```
#include "ac_fixed.h"  // From Mentor Graphics
#define PORT_SIZE ac_fixed<16, 12, true, AC_RND,AC_SAT>
// 16 bits, 12 bits after the point, quantization = rounding, overflow = saturation
#define N 16
int main(PORT_SIZE data_in, PORT_SIZE &data_out)
{
  static const PORT_SIZE Coeffs [N]={1.1, 1.5, 1.0, 1.0, 1.7, 1.8, 1.2, 1.0, 1.6, 1.0, 1.5, 1.1, 1.9, 1.3, 1.4, 1.7};
  PORT_SIZE Values[N];
  PORT_SIZE temp;
  PORT_SIZE sample;

  sample= data_in;
  temp = sample * Coeffs[N-1];
  for(int i = 1; i<=(N-1); i++){
     temp = Values [i] * Coeffs[N-i-1] + temp;
  }

  for(int j=(N-1); j>=2; j-=1 ){
     Values[j] = Values [j-1];
  }
  Values[1] = sample;


  data_out=temp;
  return 0;
}
```

Fixed-point

□Fixed point:



Integer part: *m* bits

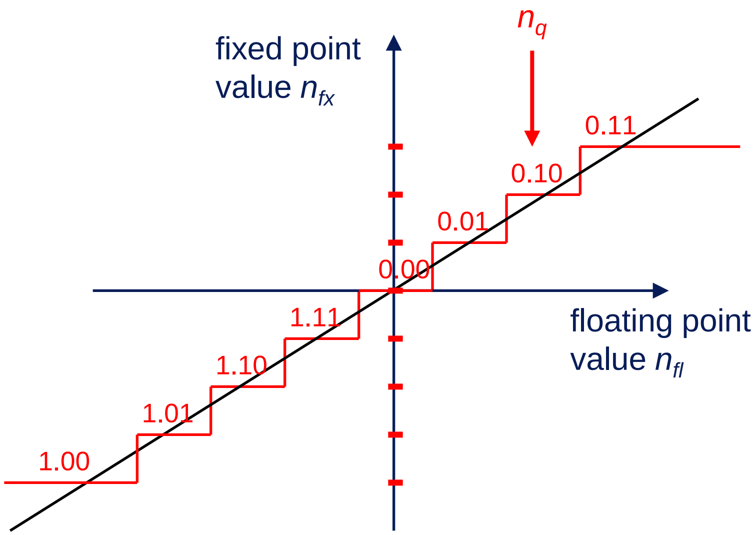
Fractional part: *n* bits

$b = m + n + 1$ bits

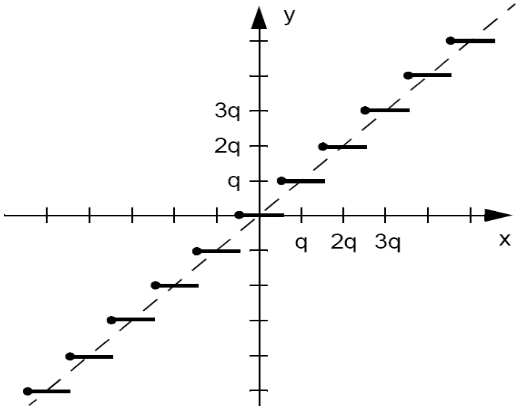
format*: (b,m,n)*

□Ac_fixed<W,I,S,Q,O>

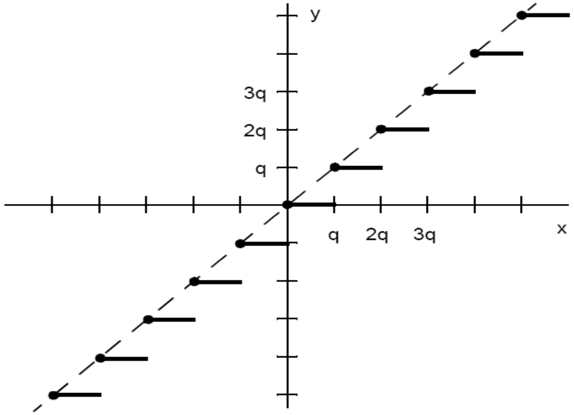
- ■W : word size (m+n+1)
- ■I : integer part size (m+1)
- ■S : signed or unsigned
- ■Q : rounding mode
- ■O : overflow mode
- ■Equivalent to SystemC data type sc_fixed



Fixed point: rounding mode

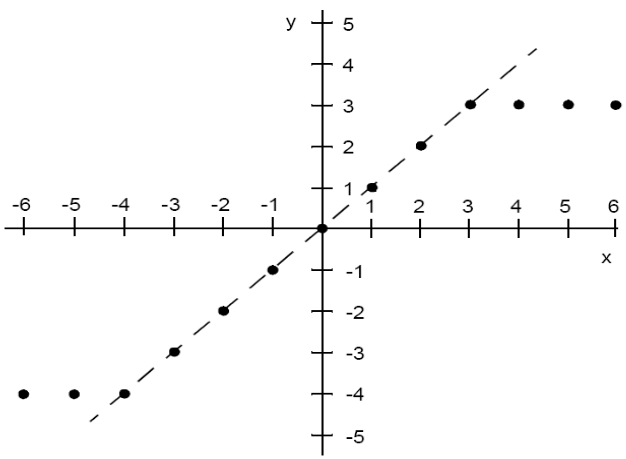


SC_RND

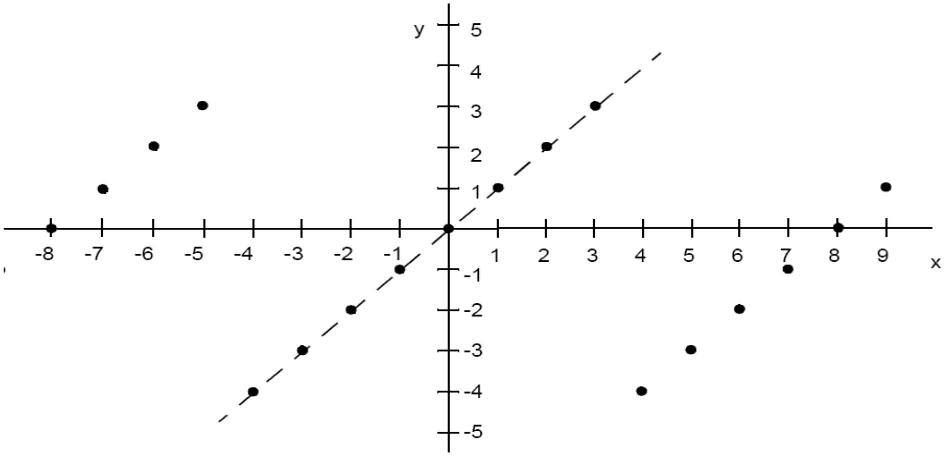


SC_TRN

Fixed point: overflow mode



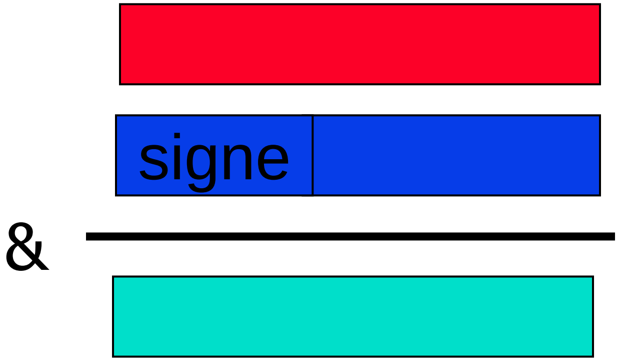
SC_SAT



SC_WRAP

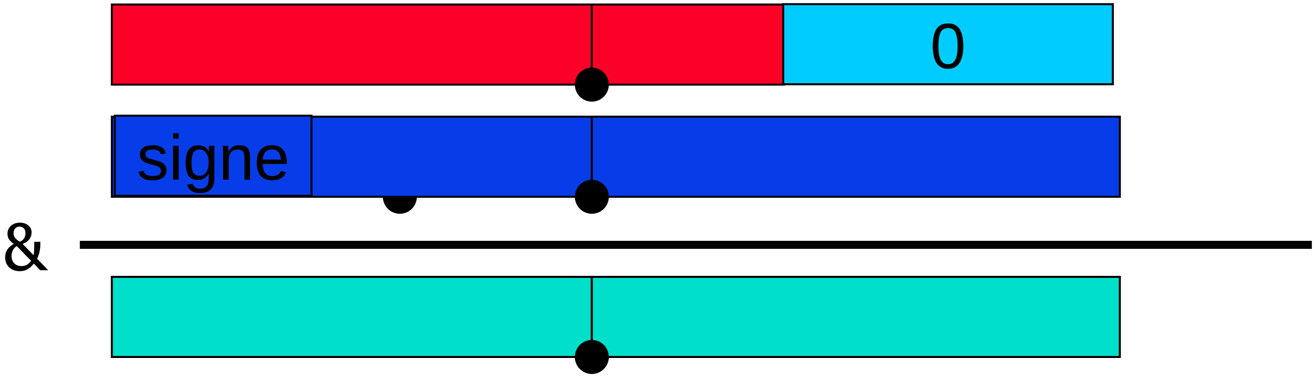
Bit accurate operation

- □ **Sign extension before the computation**



Fixed point operation

- □ **Alignment before the computation**



High-level synthesis goal

- **Starting from a functional description, automatically generate an RTL architecture**
  - Algorithmic description
    - ***no timing notion in the source code***
  - Behavioral description
    - ***Notion of step / local timing constraints in the source code***
      - by using the wait statements of SystemC for example
  - The description can be
    - ***“RTL oriented”***
    - ***“Function oriented”***

High-level synthesis

- **Starting from a functional description, automatically generate an RTL architecture**
  - Algorithmic description
    - ***No timing notion in the source code***
    - ***Mainly oriented toward data dominated application***
      - Highly processing algorithm like filters…
    - ***Initial description can be***
      - “RTL oriented”
      - “Function oriented”
  - Behavioral description
    - ***Notion of step / local timing constraints in the source code***
      - by using the wait statements of SystemC for example
    - ***Can be used for both data and control dominated application***
      - Interface controller, DMA…
      - Filters…

High-level synthesis

- **Starting from a functional description, automatically generate an RTL architecture**

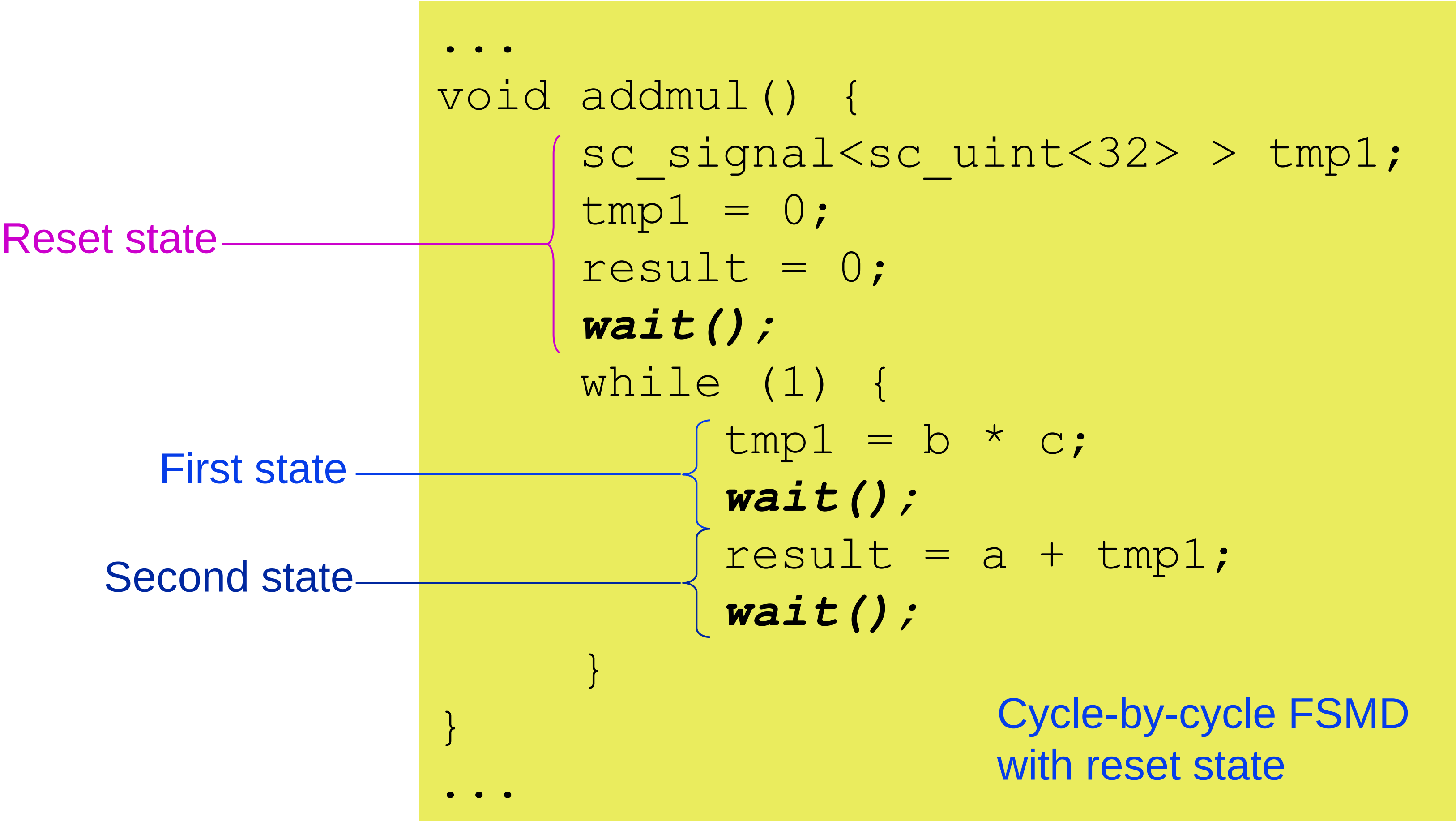
  - Algorithmic description
    - ***No timing notion in the source code***
    - ***Mainly oriented toward data dominated application***
      - Highly processing algorithm like filters…
    - ***Initial description can be***
      - “RTL oriented”
      - “Function oriented”

  - Behavioral description
    - ***Notion of step / local timing constraints in the source code***
      - by using the wait statements of SystemC for example
    - ***Can be used for both data and control dominated application***
      - Interface controller, DMA…
      - Filters…

Behavioral description

Behavioral description

- ❑ **Notion of step / local timing constraints in the source code**
  - by using the wait statements of SystemC for example



```
...
void addmul() {
        sc_signal<sc_uint<32> > tmp1;
        tmp1 = 0;
        result = 0;
        wait();
        while (1) {
                tmp1 = b * c;
                wait();
                result = a + tmp1;
                wait();
        }
}
...
```

Cycle-by-cycle FSMD with reset state

Function v.s. RTL description

```
int OnesCounter(int Data){
 int Ocount = 0;
 int Temp, Mask = 1;
 while (Data > 0) {
  Temp = Data & Mask;
  Ocount = Data + Temp;
  Data >>= 1;
 }
 return Ocount;
}
```

Function-based C code

```
while(1) {
  while (Start == 0);
  Done = 0;
  Data = Input;
  Ocount = 0;
  Mask = 1;
  while (Data>0) {
    Temp = Data & Mask;
    Ocount = Ocount + Temp;
    Data >>= 1;
  }
  Output = Ocount;
  Done = 1;
}
```

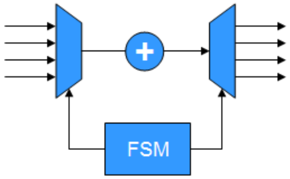
RTL-based C code

Source :
Embedded System Design, © 2009, Gajski, Abdi, Gerstlauer, Schirner

High-level transformations

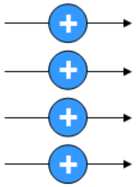
- **Loops**
  - loop unrolling
    - ***None, partially, completely***
  - Loop merging
  - Loop tiling
  - …

```
for (i = 0; i<4; i++)
{
   r[i] = a[i] + b[i];
}
```



No Unrolling
1 Adder shared for 4 additions
Latency = 4 cycles

```
r[0] = a[0] + b[0];
r[1] = a[1] + b[1];
r[2] = a[2] + b[2];
r[3] = a[3] + b[3];
```



Unrolling = 4 (Full)
4 Adders in parallel
Latency = 1 cycle

```
for (i = 0; i<32; i++)
{
   a[i] = b[i] * c[i];
}
for (i = 0; i<16; i++)
{
   z[i] = a[i] + x[i];
}
```

No Merging
Loops execute sequentially
Latency = 48 cycles

```
for (i = 0; i<32; i++)
{
   atmp = b[i] * c[i];

   if (i<16)
     z[i] = atmp + x[i];
}
```

Merging Enabled
Loops execute in parallel
Latency = 32 cycles

Figures from Mentor Graphics

High-level transformations

- **Loops**
  - Loop pipelining,
  - loop unrolling
    - ***None, partially, completely***
  - Loop merging
  - Loop tiling
  - …

- **Arrays**
  - Arrays can be mapped on memory banks
  - Arrays can be synthesized as registers
  - Constant arrays can be synthesized as logic
  - …

- **Functions**
  - Function calls can be in-lined
  - Function is synthesized as an operator
    - ***Sequential, pipelined, functional unit…***
  - Single function instantiation
  - …

Compilation

- **Optimization**
  - Constant folding
  - Dead code elimination
  - Common sub-expression elimination
    - ***Eliminate redundant operations***
  - ...

- **Formal model**
  - Inputs, outputs, and operations of the algorithm are identified
  - Data and/or control dependencies are determined
  - Intermediate representation is generated

Control-Flow Graph CFG

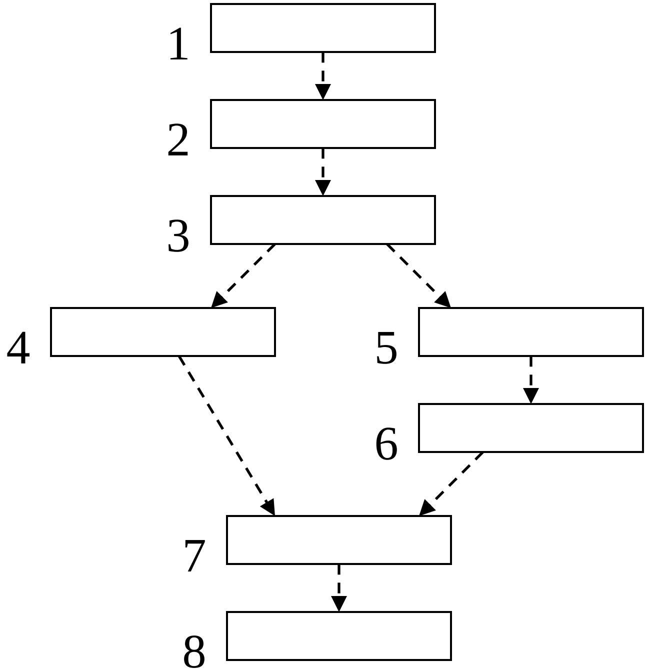
- **Exhibits operation sequences**
  - Through control dependencies

- **The sequence of operations comes directly from the source code**
  - The sequence is kept unchanged
    - ***This limits the parallelism which should be limited if this representation is used to model control-oriented application***

Example

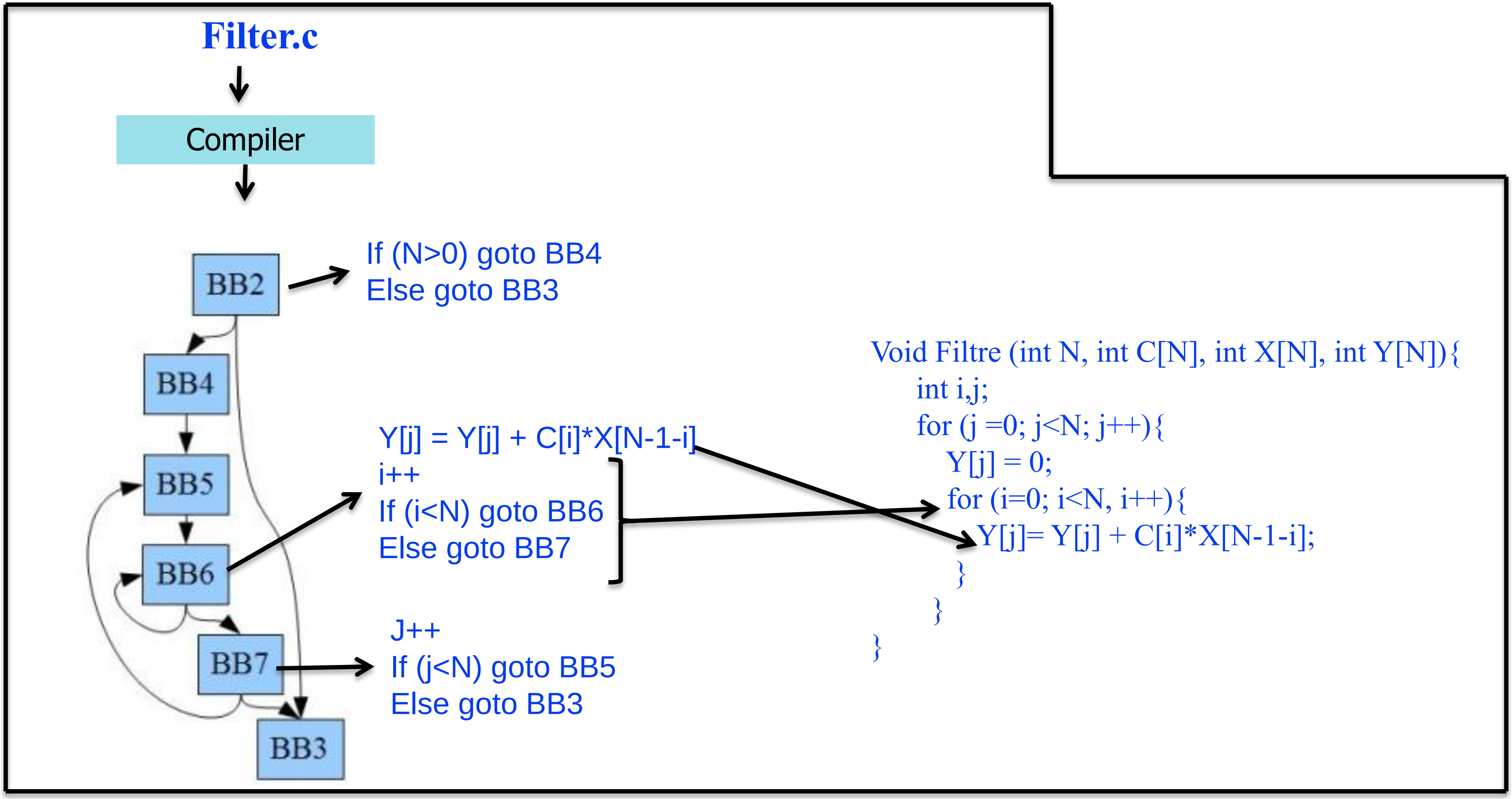
```
1: t = a+b;
2: u = a’-b’;
3: if (a<b)
4:     v = t+c;
else
{
5:     w = u+c’;
6:     v = w-d;
}
7: x = v+e;
8: y = v-e;
```

Source code



Graphical representation

Example (2)



Filter.c

Compiler

If (N>0) goto BB4
Else goto BB3

Y[j] = Y[j] + C[i]*X[N-1-i]
i++
If (i<N) goto BB6
Else goto BB7

J++
If (j<N) goto BB5
Else goto BB3

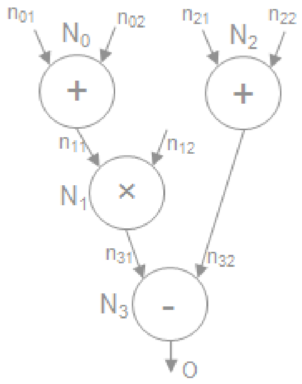
```
Void Filtre (int N, int C[N], int X[N], int Y[N]){
     int i,j;
     for (j =0; j<N; j++){
        Y[j] = 0;
        for (i=0; i<N, i++){
          Y[j]= Y[j] + C[i]*X[N-1-i];
        }
     }
}
```

Data Flow Graph DFG

- □ **Exhibits the parallelism between operations**
    - Through data dependencies
        - ❑ ***Variable node, operation node***

$O = ((n_{01}+n_{02})*n_{12})-(n_{21}+n_{22})$

Intermediate representation



CDFG => DFG

- □ **Exhibits the parallelism between operations**
  - Through data dependencies
    - ❑ ***Variable node, operation node***

- □ **Loops are completely unrolled**

```
for i : 0 → 2
    c[i] = a[i] + b[i]
```

➡

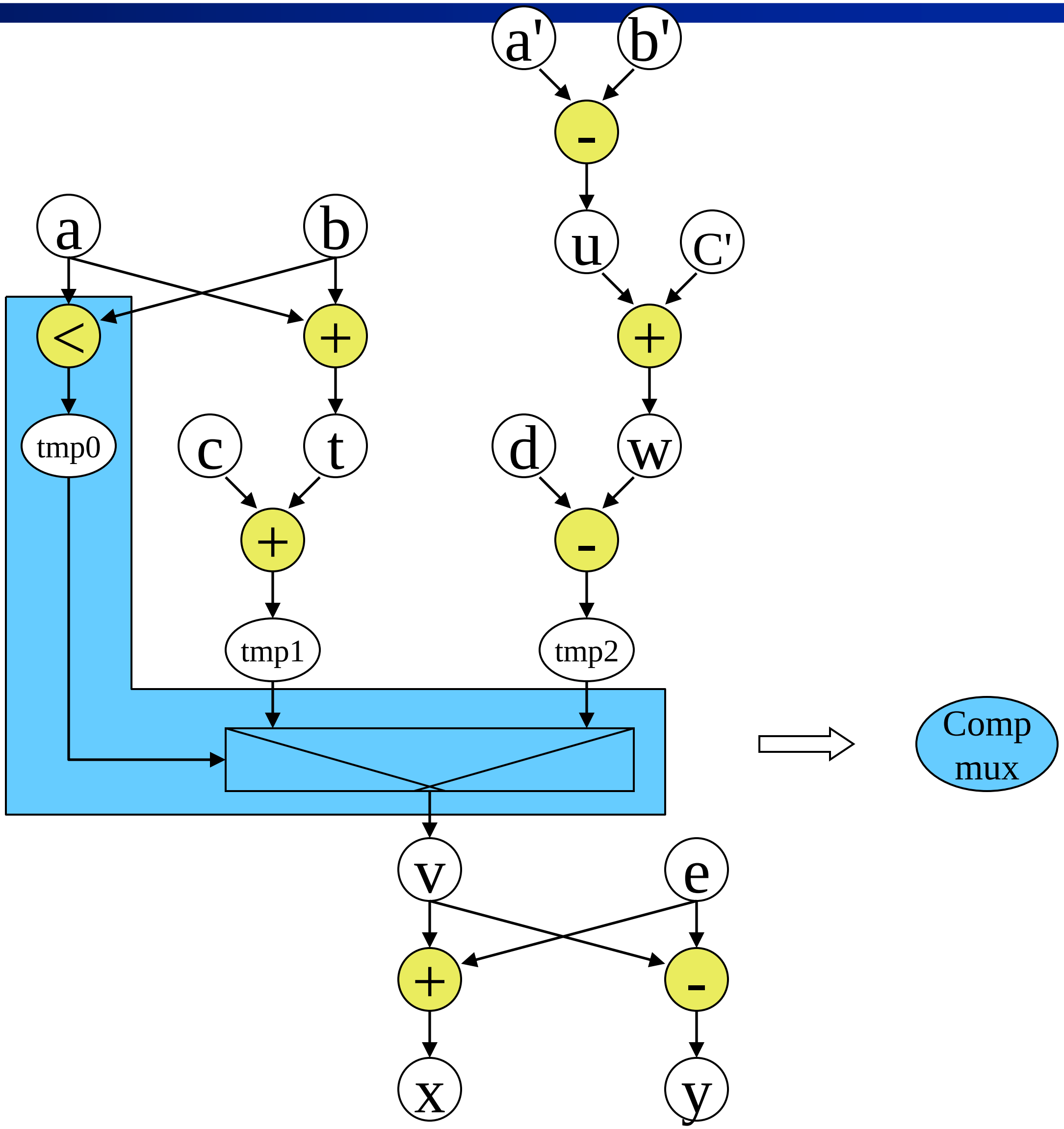
```
c[0] = a[0] + b[0]
c[1] = a[1] + b[1]
c[2] = a[2] + b[2]
```

- □ **Conditional assignments are transformed**
  - i.e. *if/switch* constructs, are resolved by creating multiplexed values

Example

```
1: t = a+b;
2: u = a’-b’;
3: if (a<b)
4:     v = t+c;
else
{
5:     w = u+c’;
6:     v = w-d;
}
7: x = v+e;
8: y = v-e;
```

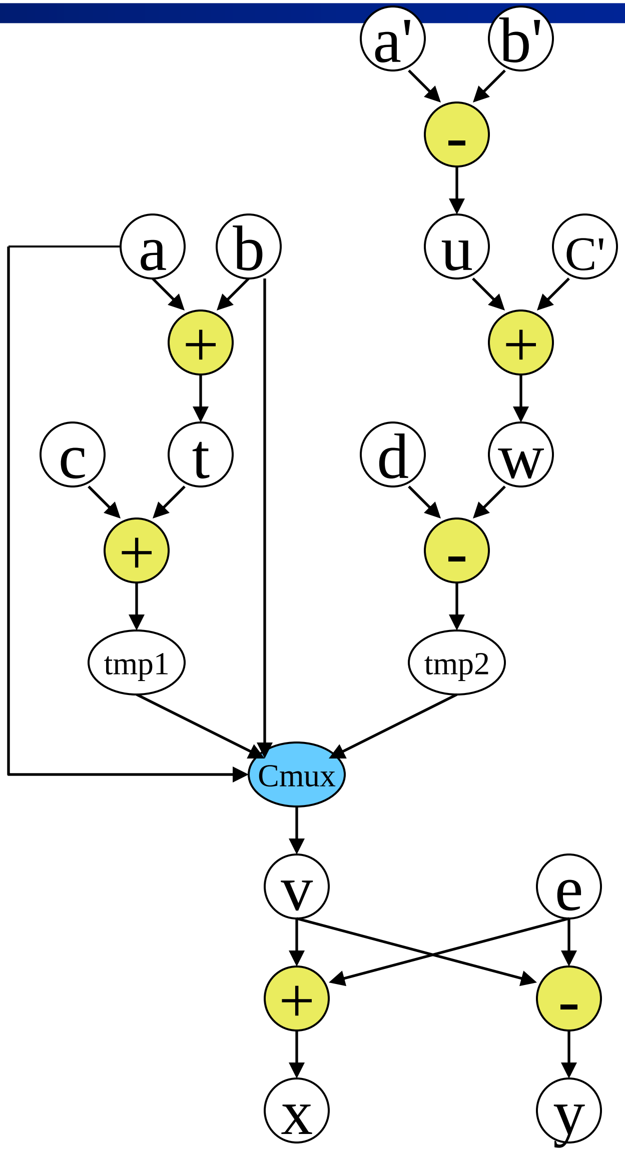
Source code



Example

```
1: t = a+b;
2: u = a’-b’;
3: if (a<b)
4:     v = t+c;
else
{
5:     w = u+c’;
6:     v = w+d;
}
7: x = v+e;
8: y = v-e;
```

Source code



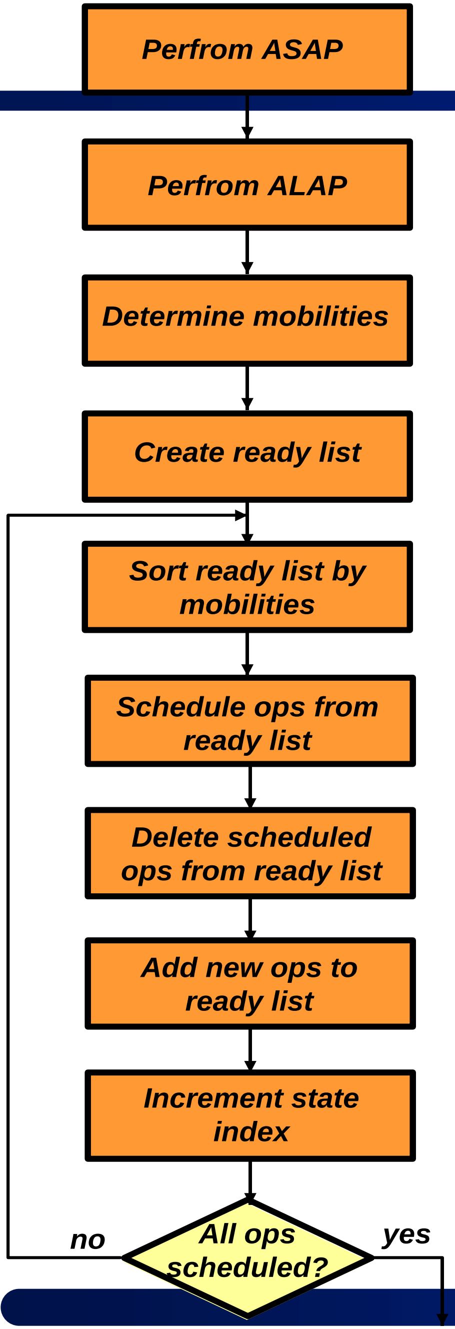
Data Flow Graph DFG

- **Scheduling**
  - Resource constrained
    - ***Latency minimization***
      - *List-Scheduling…*
    - ***Throughput maximization***
      - *Modulo scheduling (IMS, SMS…)*
  - Time constrained
    - ***Resource minimization***
      - *Force-directed scheduling, ILP...*
- **Linear FSM controller**
  - Worst execution time for the conditional assignments

Synthesis steps

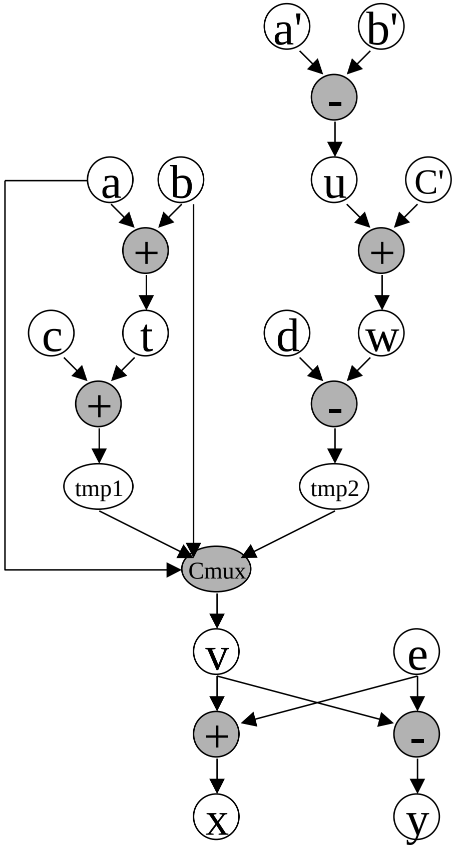
- □ **Compilation**
  - Generates a formal modeling of the specification
- □ **Selection**
  - Chooses the architecture of the operators
- □ **Allocation**
  - Defines the number of operators for each selected type
- □ **Scheduling**
  - Defines the execution date of each operation
- □ **Binding (or Assignment)**
  - Defines which operator will execute a given operation
  - Defines which memory element will store a data
- □ **Architecture generation**
  - Writes out the RTL source code in the target language e.g. VHDL

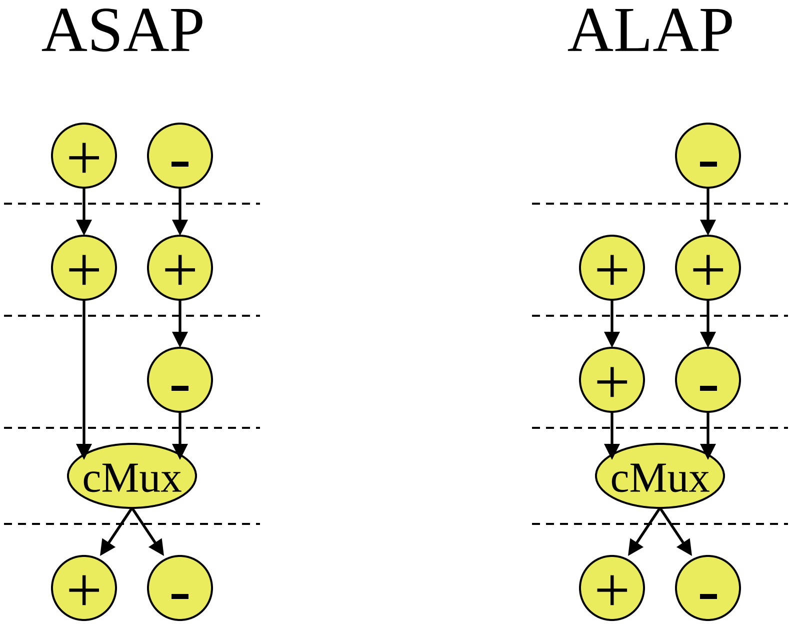
Resource constrained scheduling


Perfrom ASAP
Perfrom ALAP
Determine mobilities
Create ready list
Sort ready list by mobilities
Schedule ops from ready list
Delete scheduled ops from ready list
Add new ops to ready list
Increment state index
All ops scheduled?
no
yes

RC algorithm

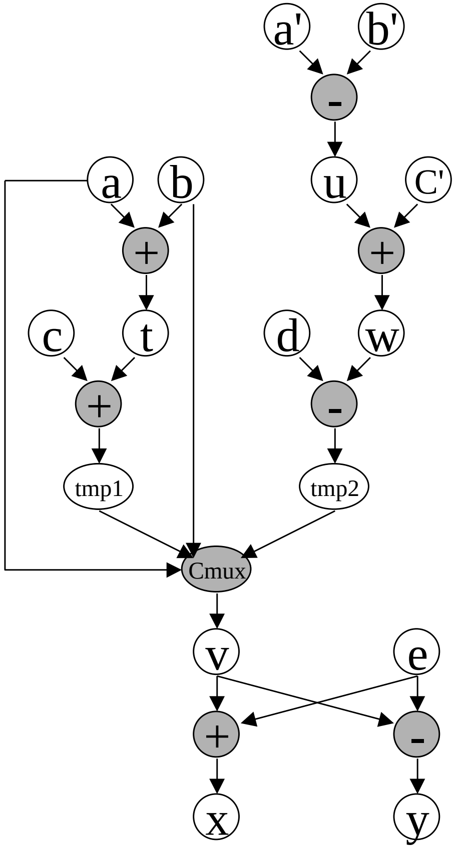
List-scheduling

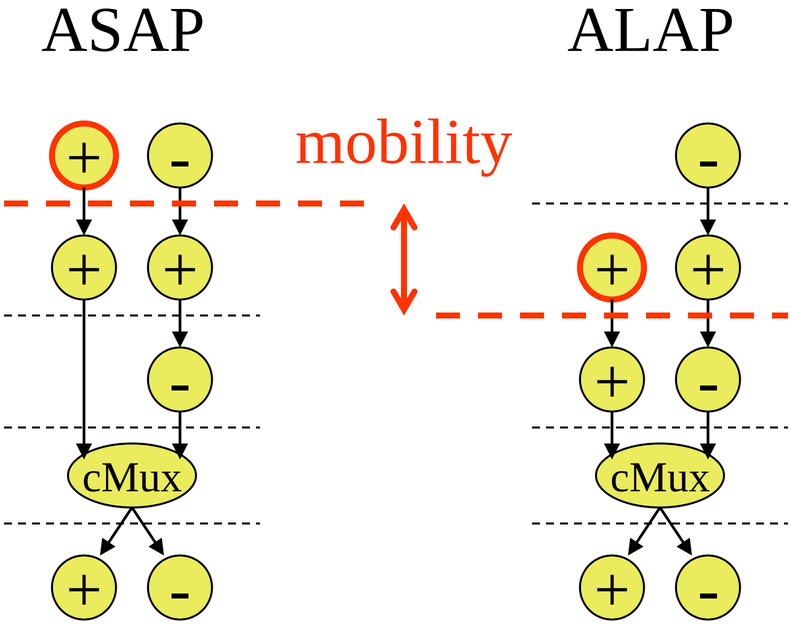




- **Constraints**
  - 1 adder (1 cycle)
  - 1 subtractor (1 cycle)
  - 1 comparing component (1 cycle)
  - No chaining

List-scheduling

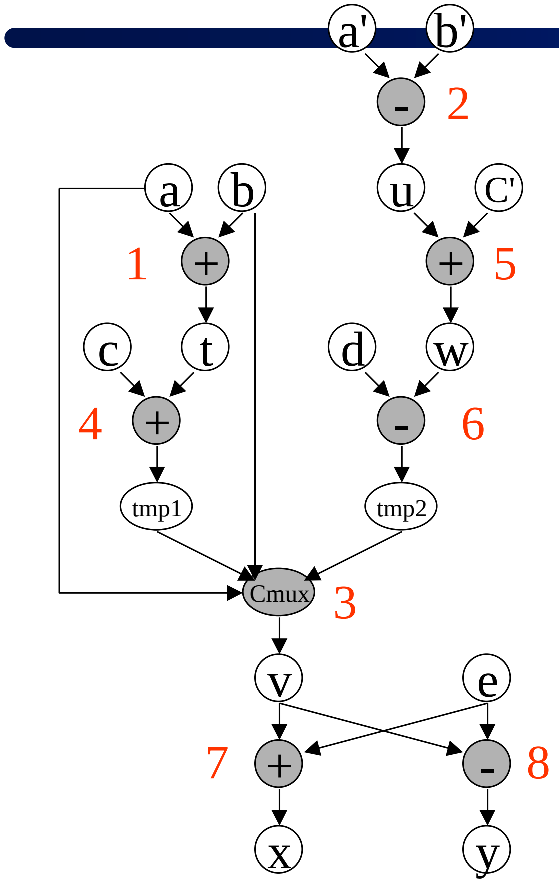


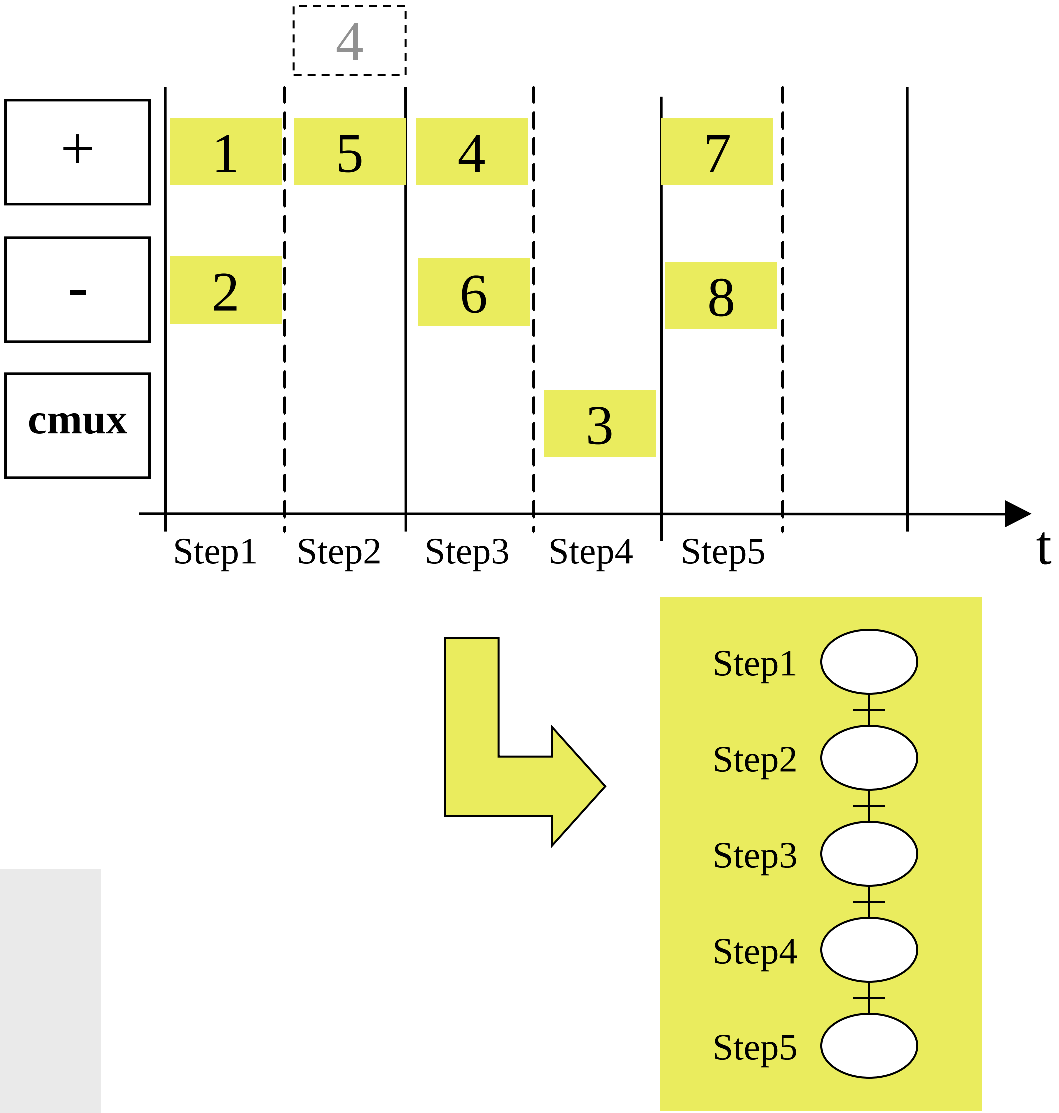


- **Constraints**
  - 1 adder (1 cycle)
  - 1 subtractor (1 cycle)
  - 1 comparing component (1 cycle)
  - No chaining

Priority = 1/Mobility

List-scheduling





□ Constraints

- 1 adder (1 cycle)
- 1 subtractor (1 cycle)
- 1 comparing component (1 cycle)
- No chaining

List-based scheduling

Scheduling under throughput constraint (cadency)

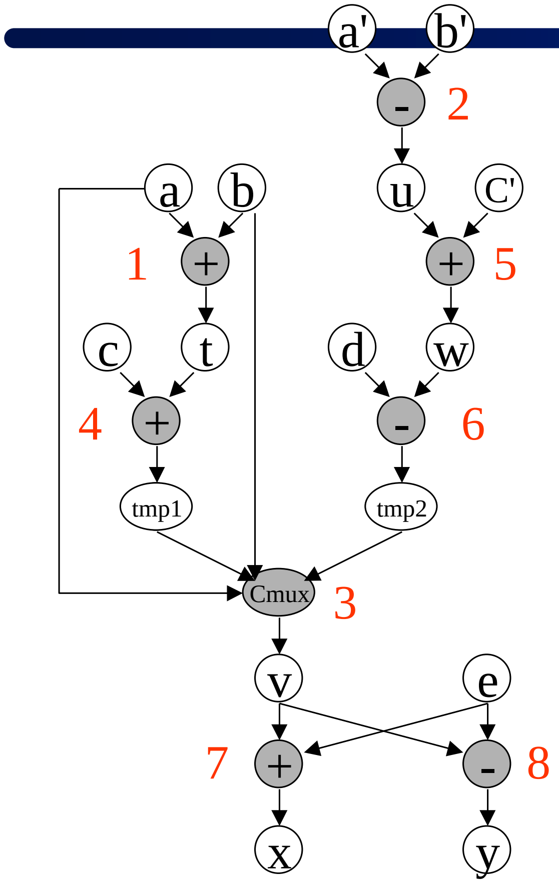
- First operator allocation that a priori support the required parallelism
  - ***In many HLS approach, an initial resource allocation is performed and subsequently modified during scheduling and/or binding => it is a lower bound***

- The average parallelism is calculated separately for each *type* of operation of the DGF

$$avr\_opr(type) = \left\lceil \frac{nb\_ops(type)}{\left\lfloor \frac{II}{T(opr)} \right\rfloor} \right\rceil$$

With *II* the Initiation Interval (cadency)
nb_ops(type) the number of operators of type *type*
T(opr) the propagation time of the operator (in cycles)

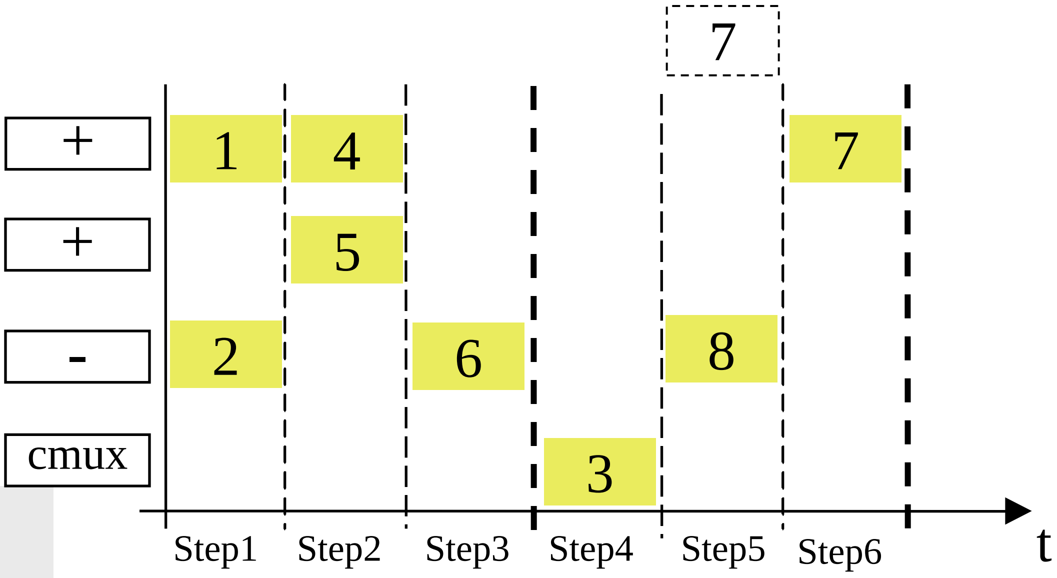
List-based scheduling



$$\#adder = \lceil 4 * 1/3 \rceil = 2$$

$$\#sub = \lceil 3 * 1/3 \rceil = 1$$

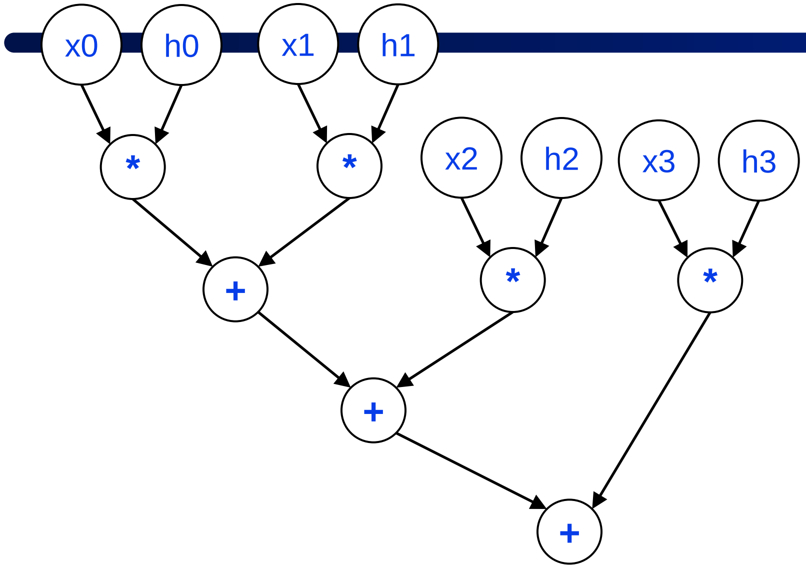
$$\#cmux = \lceil 1 * 1/3 \rceil = 1$$



	Step1	Step2	Step3	Step4	Step5	Step6
+	1	4				7
+		5				
-	2		6		8	
cmux				3		

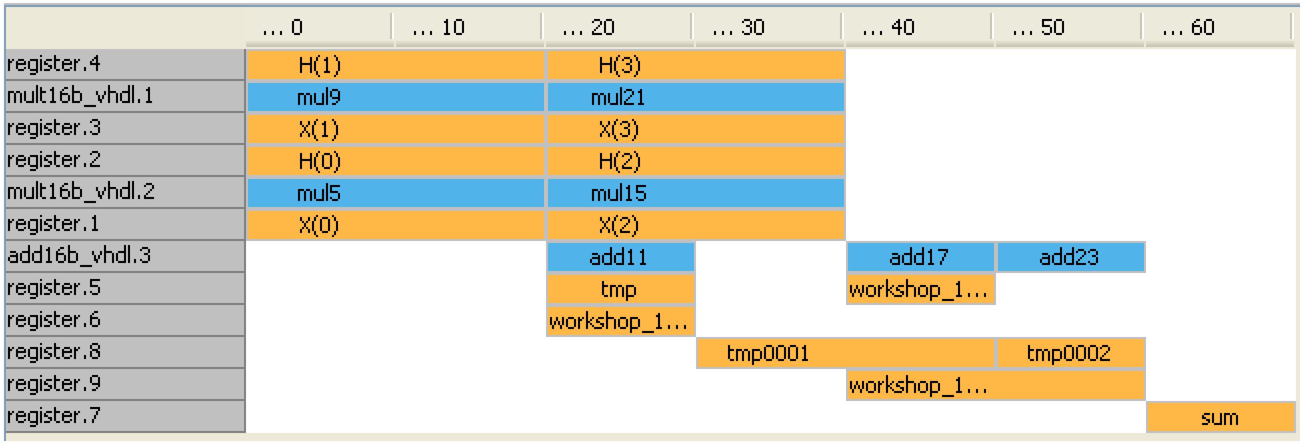
- □ **Constraint**
  - Throughput : one iteration each 3 cycles

Impact on the memory



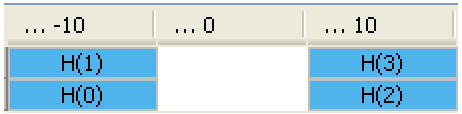
Input : X[N]
Constant : H[N] // in memory

Without memory constraints
Iteration_period = 60ns
Nb_opr(*) = 2
Nb_opr(+) = 1



	… 0	… 10	… 20	… 30	… 40	… 50	… 60
register.4	H(1)		H(3)				
mult16b_vhdl.1	mul9		mul21				
register.3	X(1)		X(3)				
register.2	H(0)		H(2)				
mult16b_vhdl.2	mul5		mul15				
register.1	X(0)		X(2)				
add16b_vhdl.3			add11		add17	add23	
register.5			tmp		workshop_1…		
register.6			workshop_1…				
register.8				tmp0001		tmp0002	
register.9					workshop_1…		
register.7							sum

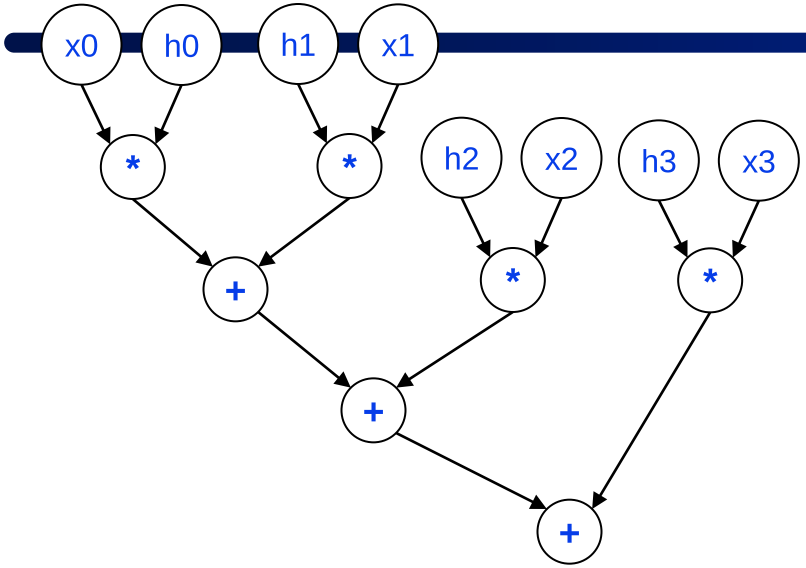
=> 2 memory banks
Latency(arch) = 60 ns



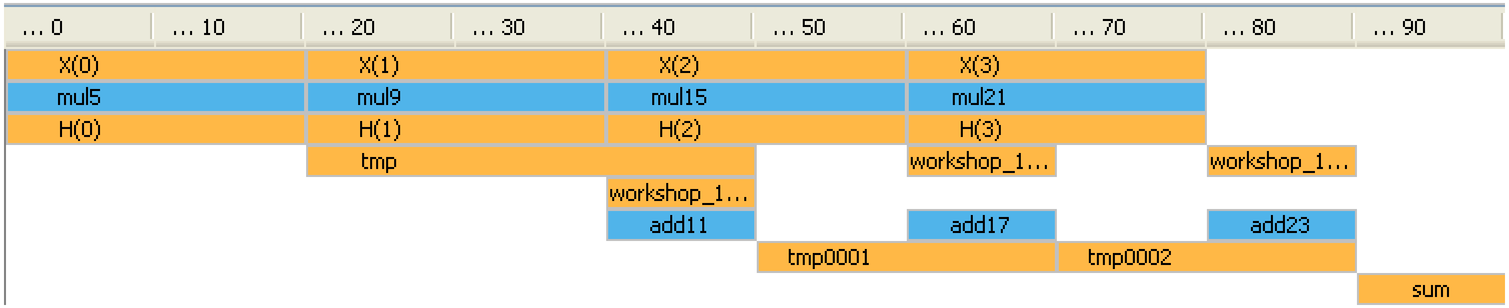
… -10	… 0	… 10
H(1)		H(3)
H(0)		H(2)

The memory mapping has to be done by the user

Memory constraints



With memory constraints
=> 1 memory bank
Iteration_period = 100ns
Nb_opr(*) = 1
Nb_opr(+) = 1

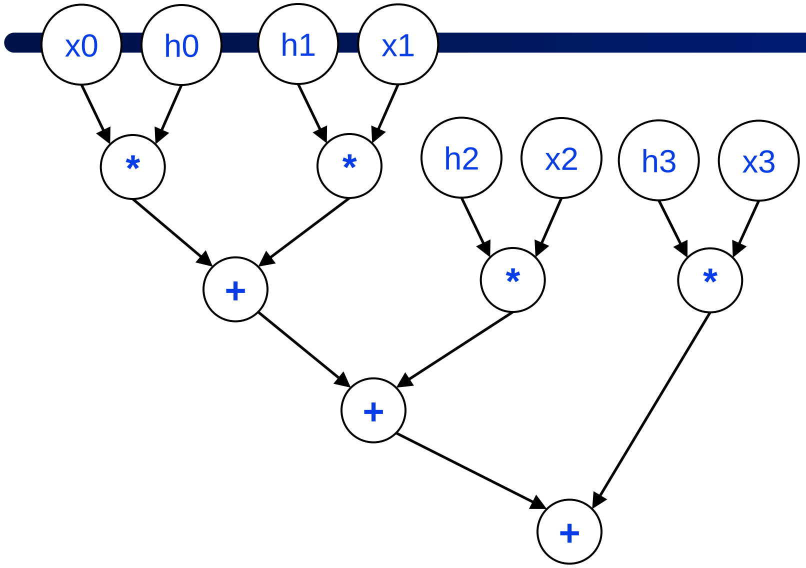


… 0	… 10	… 20	… 30	… 40	… 50	… 60	… 70	… 80	… 90
X(0)		X(1)		X(2)		X(3)			
mul5		mul9		mul15		mul21			
H(0)		H(1)		H(2)		H(3)			
		tmp				workshop_1…		workshop_1…	
				workshop_1…					
				add11		add17		add23	
					tmp0001		tmp0002		
									sum

Latency(arch) = 90 ns

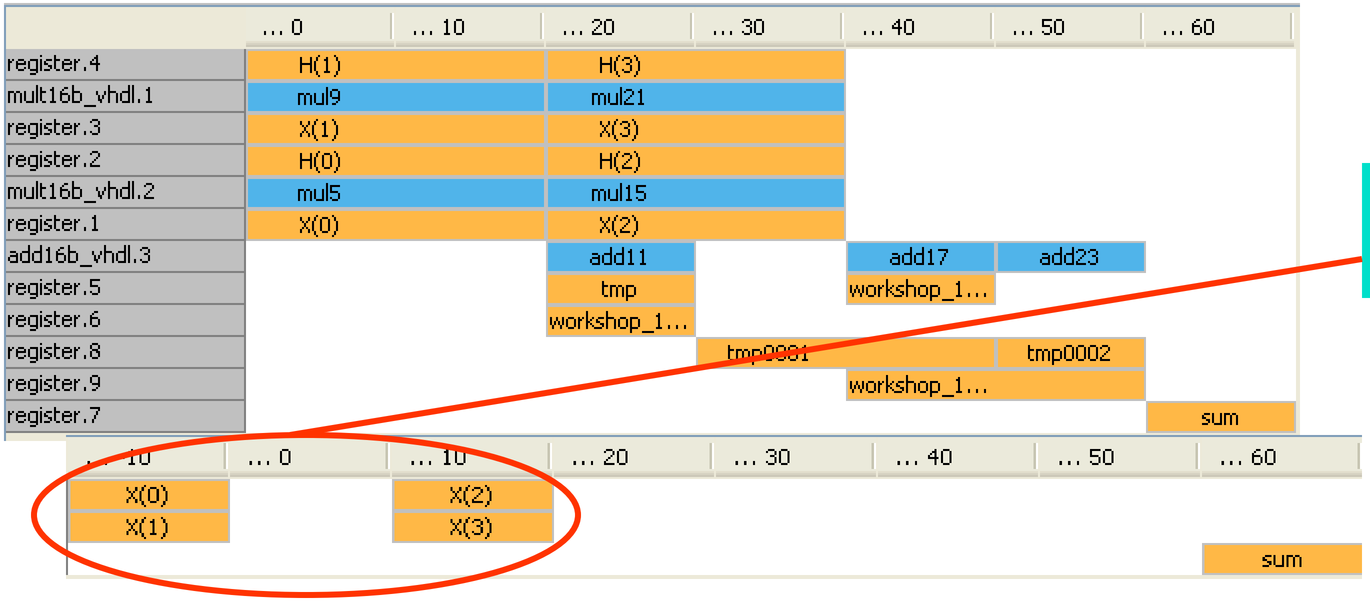
The memory access is the bottleneck !

Impact on the I/O interface



Input : X[N]
Constant : H[N] // in memory

Without I/O constraints
Iteration_period = 60ns
Nb_opr(*) = 2
Nb_opr(+) = 1

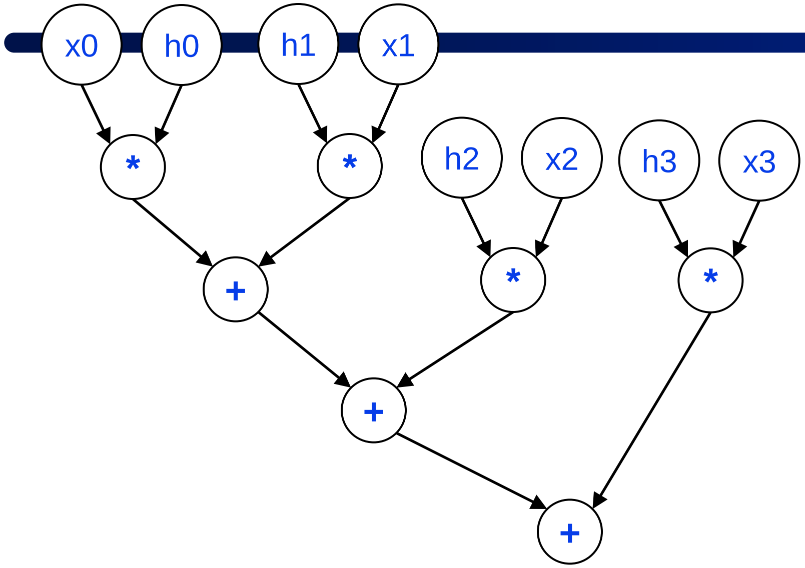


	... 0	... 10	... 20	... 30	... 40	... 50	... 60
register.4	H(1)		H(3)				
mult16b_vhdl.1	mul9		mul21				
register.3	X(1)		X(3)				
register.2	H(0)		H(2)				
mult16b_vhdl.2	mul5		mul15				
register.1	X(0)		X(2)				
add16b_vhdl.3			add11		add17	add23	
register.5			tmp		workshop_1...		
register.6			workshop_1...				
register.8				tmp0001		tmp0002	
register.9					workshop_1...		
register.7							sum

... 10	... 0	... 10	... 20	... 30	... 40	... 50	... 60
X(0)		X(2)					
X(1)		X(3)					
							sum

=> 2 input ports
Latency(arch) = 60 ns

I/O timing constraints



With I/O constraints

4 input data in parallel

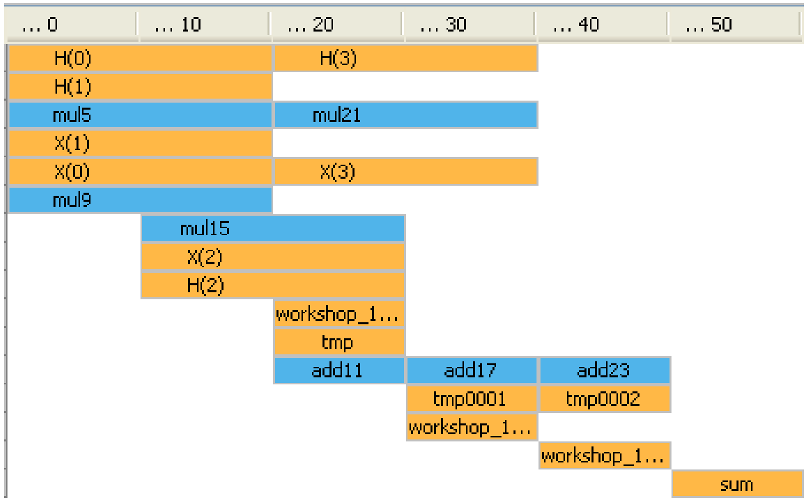
Latency = 50ns

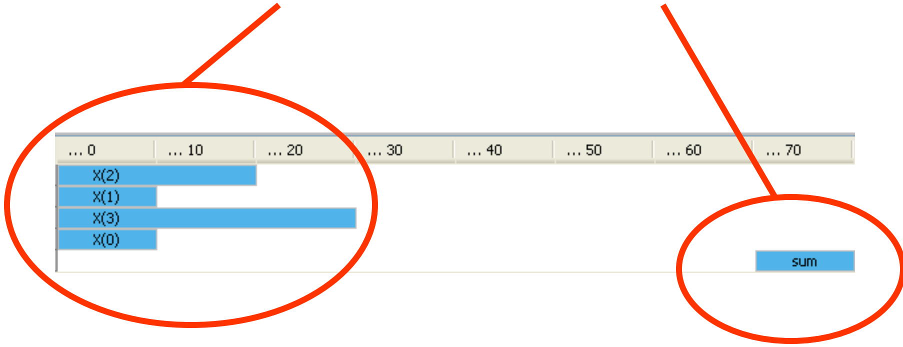
Iteration_period = 60ns

Nb_opr(*) = 3 (and not 4)

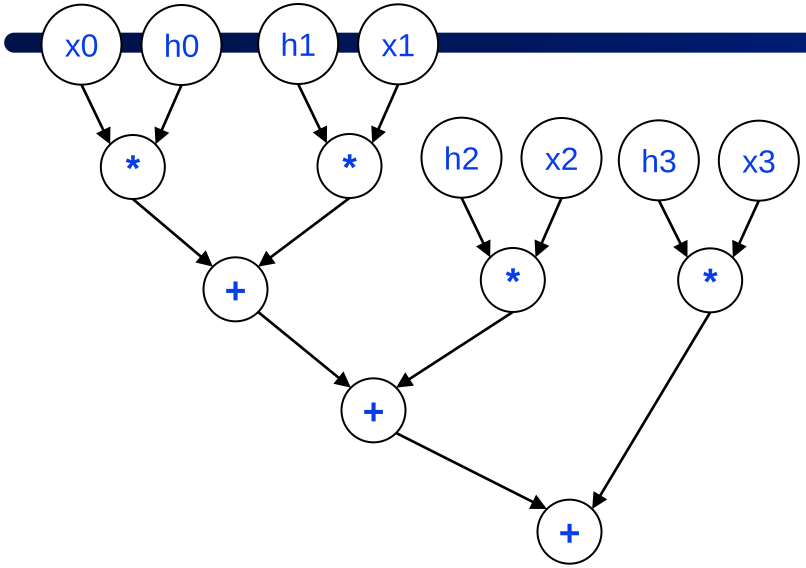
Nb_opr(+) = 1

=> 4 input ports / 1 output port





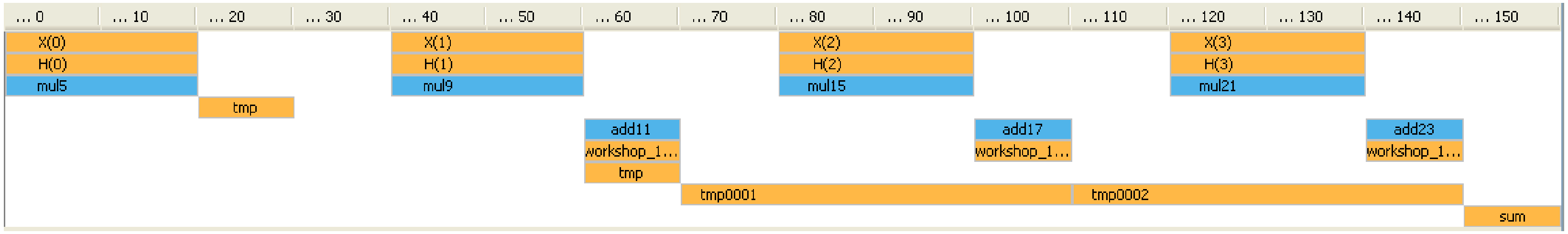
I/O timing constraints



With I/O timing constraints
1 data per 4 cycles
Latency (arch) = 150 ns

Iteration_period = 170ns
Nb_opr(*) = 1
Nb_opr(+) = 1

=> 1 input port / 1 output port



Synthesis steps

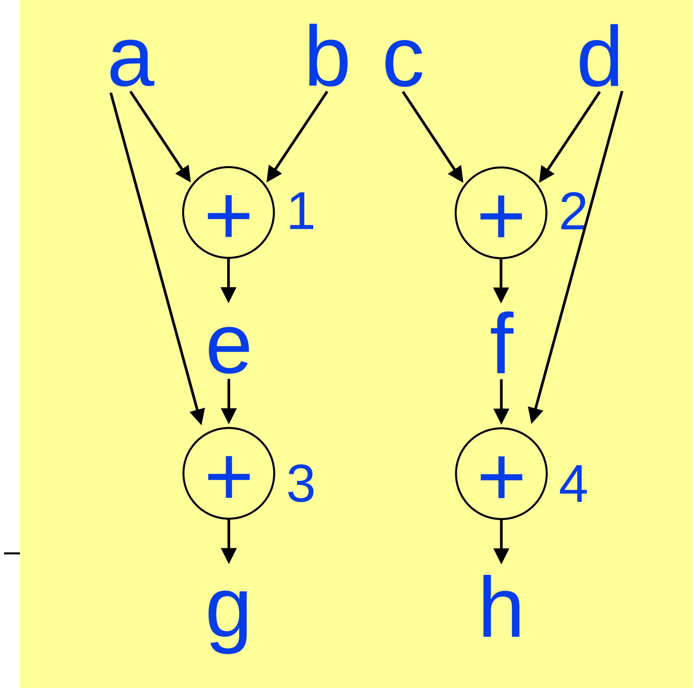
- □ **Compilation**
  - Generates a formal modeling of the specification
- □ **Selection**
  - Chooses the architecture of the operators
- □ **Allocation**
  - Defines the number of operators for each selected type
- □ **Scheduling**
  - Defines the execution date of each operation
- □ **Binding (or Assignment)**
  - Defines which operator will execute a given operation
  - Defines which memory element will store a data
- □ **Architecture generation**
  - Writes out the RTL source code in the target language e.g. VHDL

Specification

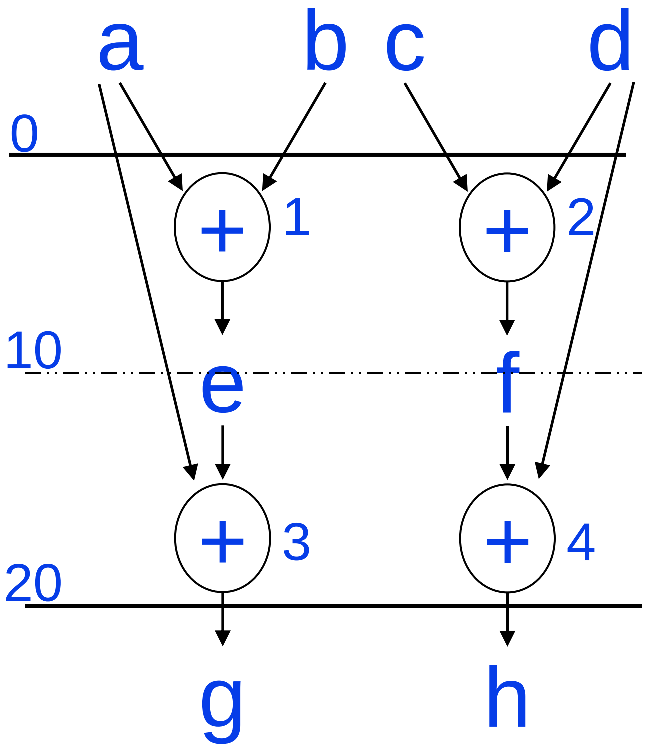
```
Void  example (int a, int b, int g, int c, int d, int h)
{
            int e,f;
            e = a+b;
            g = a+e;
            f = c+d;
            h = f+d;
}
```

Compilation => DFG

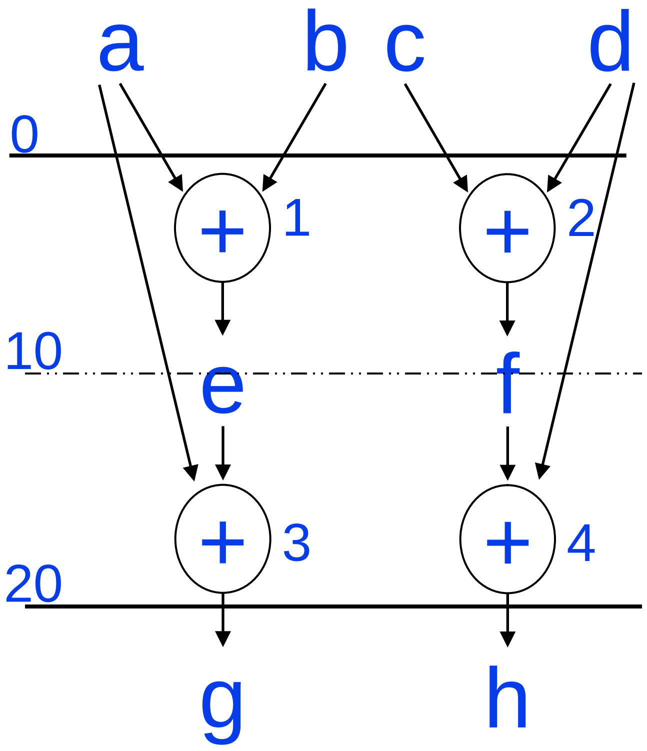
```
Void  example (int a, int b, int g, int c, int d, int h)
{
            int e,f;
            e = a+b;
            g = a+e;
            f = c+d;
            h = f+d;
}
```

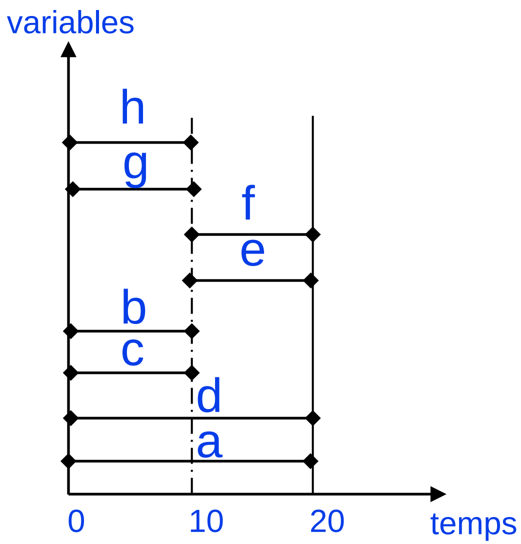


Scheduling



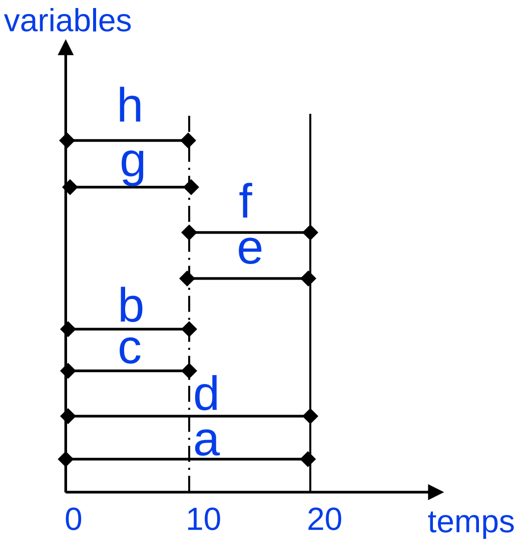
Timing information



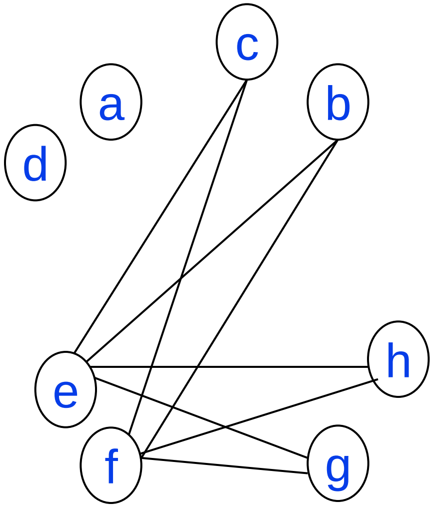


Data lifetimes

Formal model for variable binding

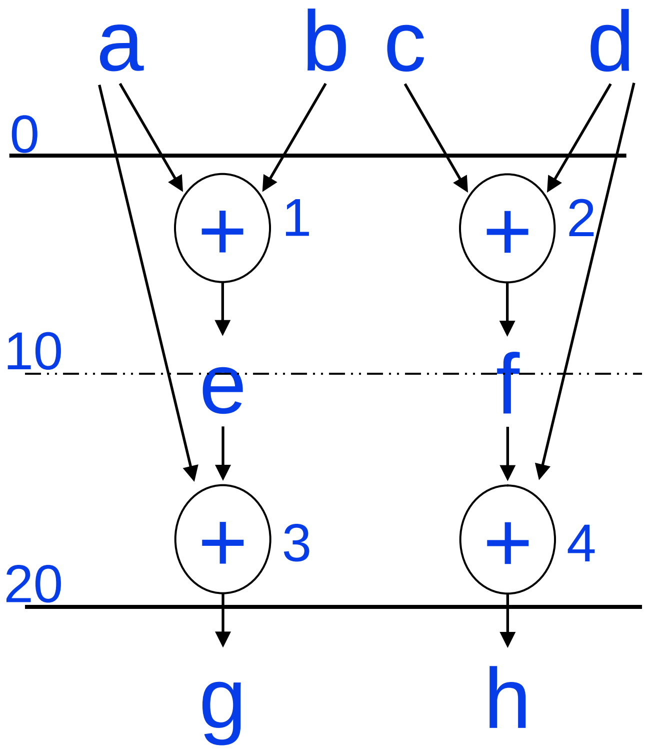


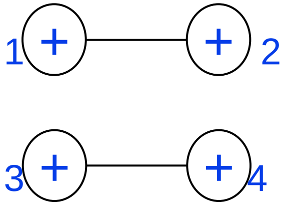
(a) Data lifetimes



(b) Compatibility graph

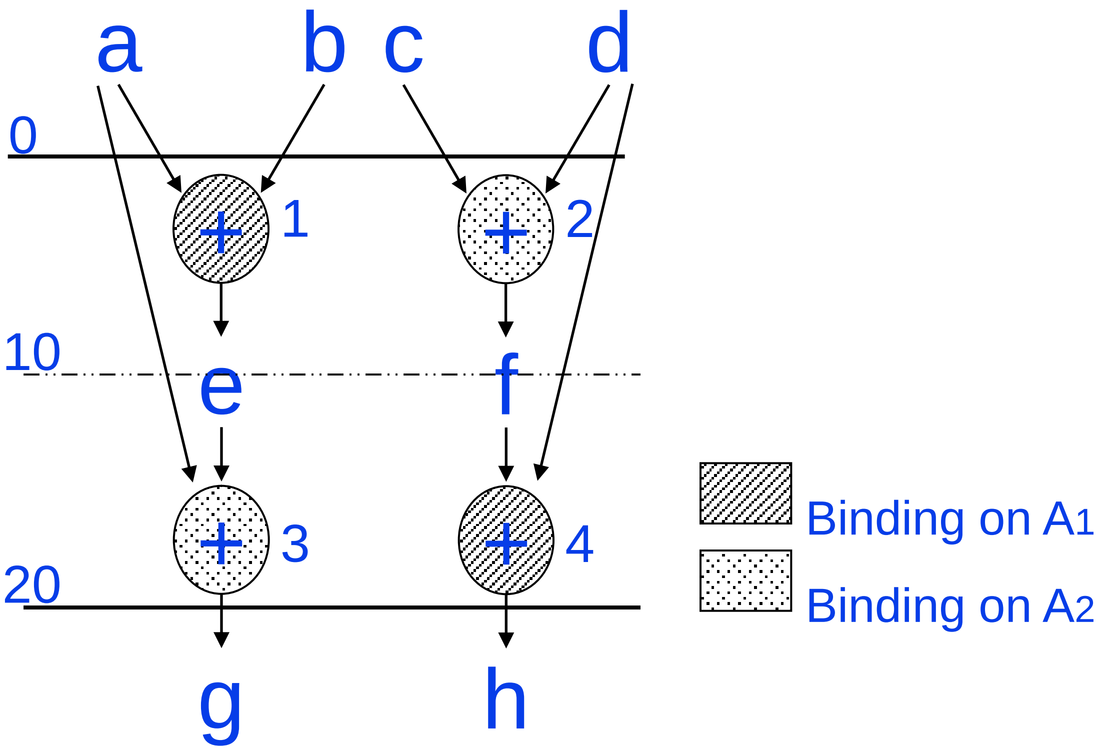
Timing information and formal model





(c) Compatibility graph

Operation binding



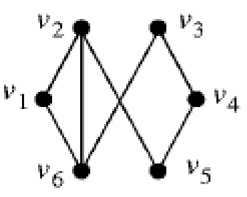
▨ Binding on A1

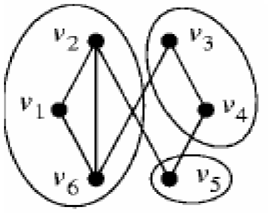
░ Binding on A2

Compatibility and conflict graphs

Clique partitioning : Binding based on a compatibility graph.
Edge exists between two data which lifetimes are not overlapping: they can share the same register.

Compatibility graph

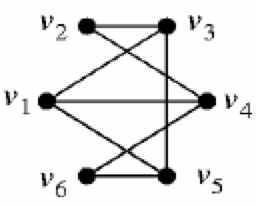


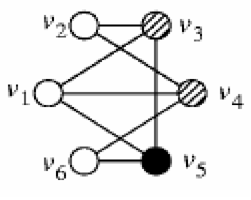


clique (sub-graph)

Graph coloring: Binding based on a conflict graph.
 Edge exists between two data which lifetimes are overlapping: they can not share the same register.

Incompatibility graph

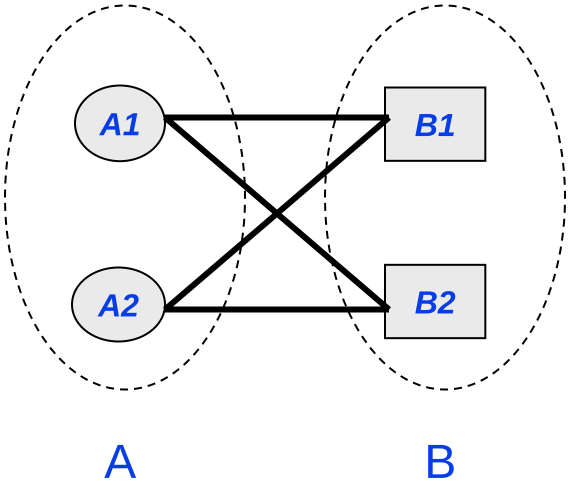




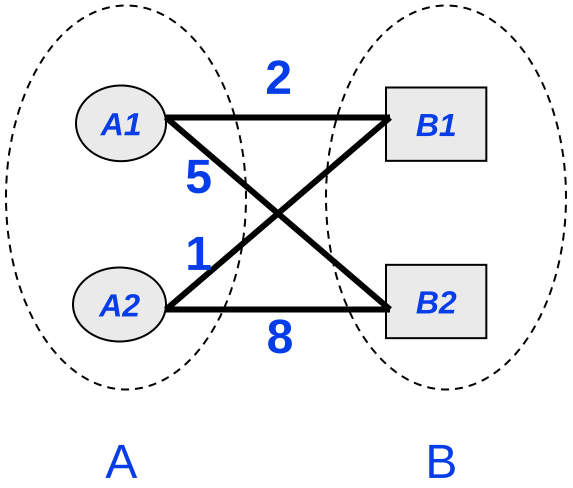
Graph coloring

(weighted) Bipartite Graph

- □ A bipartite graph is a graph whose vertices can be divided into two disjoint sets A and B such that every edge connects a vertex in A to one in B



Bipartite Graph

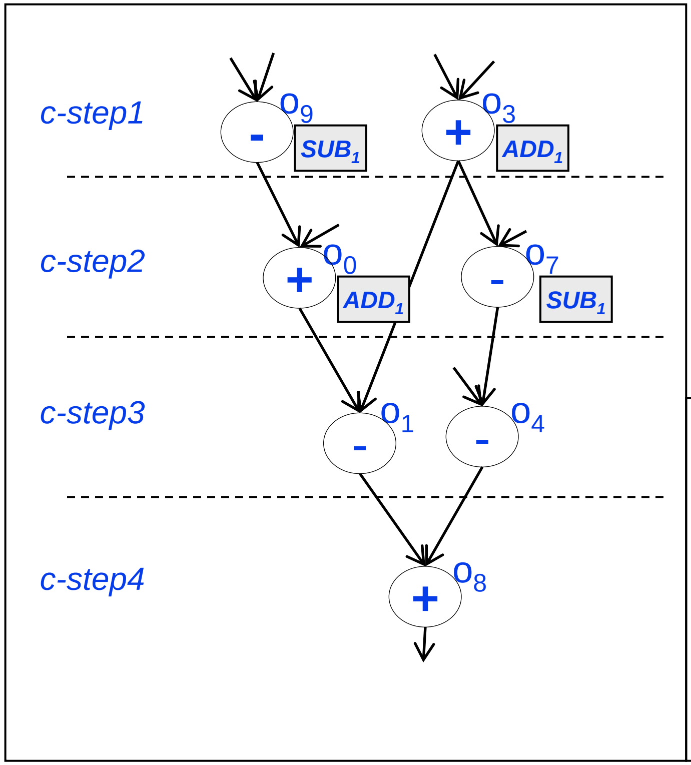


Weighted Bipartite Graph

Example

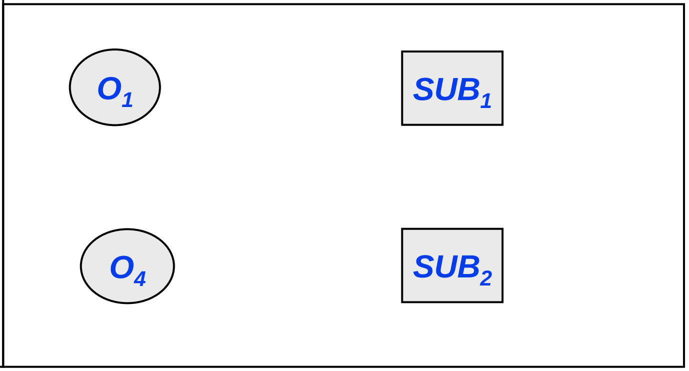
Goal: maximize the use of existing connections between operators (Muxes optimization) while minimizing their size

weight = combination between the size and the number of connection

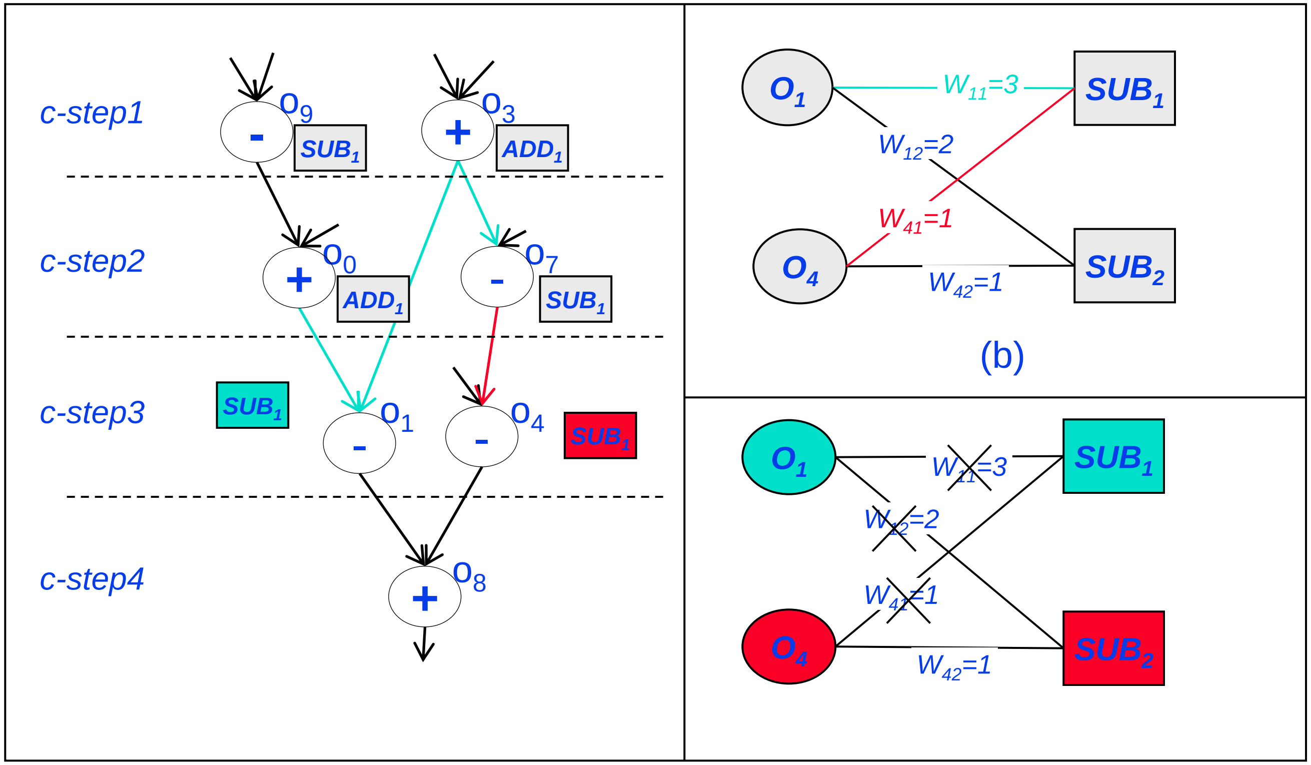


For each cycle (control step):

- Create a bipartite graph: free operators, operations to bind
- Compute weights

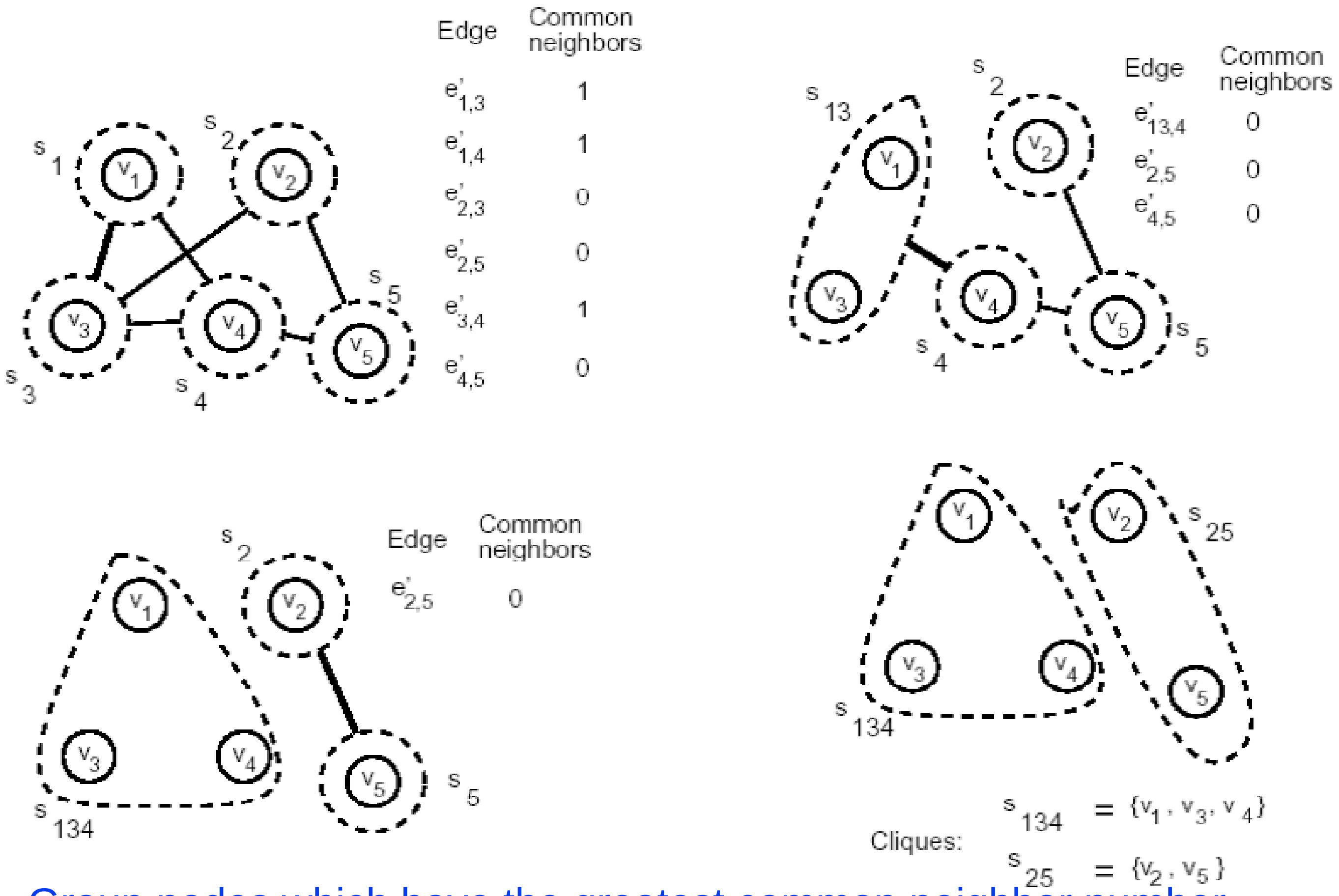


Bipartite Weighted Matching



Maximum Weighted Bipartite Matching : Hungarian method (munkres algorithm)

Clique partitioning algorithm: Tseng's Algorithm



Edge	Common neighbors
$e'_{1,3}$	1
$e'_{1,4}$	1
$e'_{2,3}$	0
$e'_{2,5}$	0
$e'_{3,4}$	1
$e'_{4,5}$	0

Edge	Common neighbors
$e'_{13,4}$	0
$e'_{2,5}$	0
$e'_{4,5}$	0

Edge	Common neighbors
$e'_{2,5}$	0

Cliques: $s_{134} = \{v_1, v_3, v_4\}$

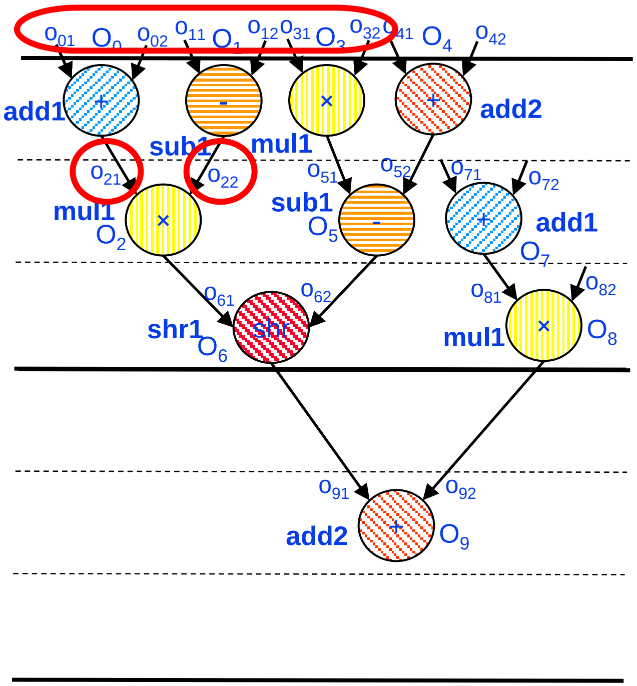
$s_{25} = \{v_2, v_5\}$

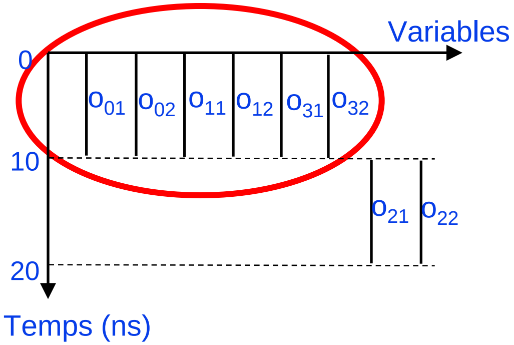
1. Group nodes which have the greatest common neighbor number
2. Repeat until all the edges are removed
3. Each clique corresponds to a storage unit

Data binding : the Left Edge algorithm

□ **Data are ordered by increasing birth date**

□ **Leftmost data are bound to distinct registers**

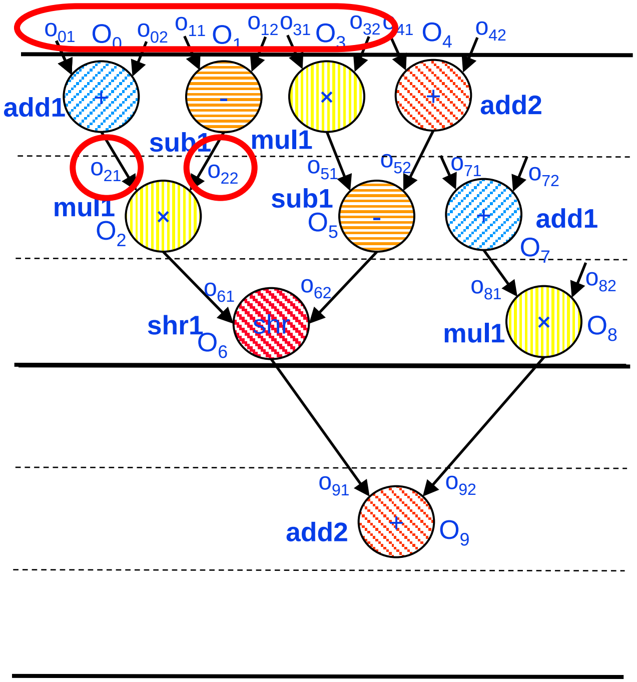


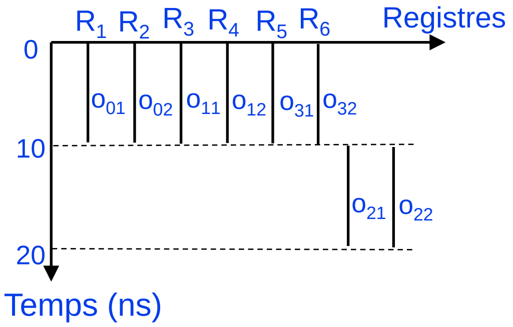


Data binding : the Left Edge algorithm

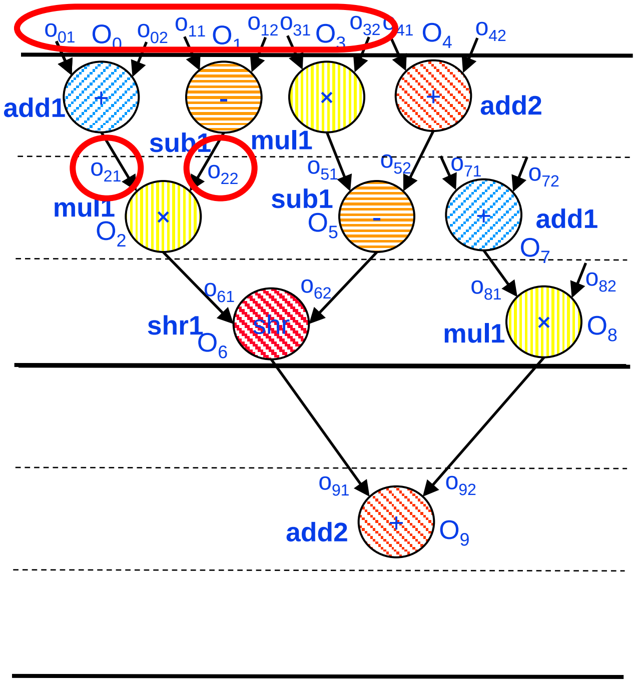
□ **Data are ordered by increasing birth date**

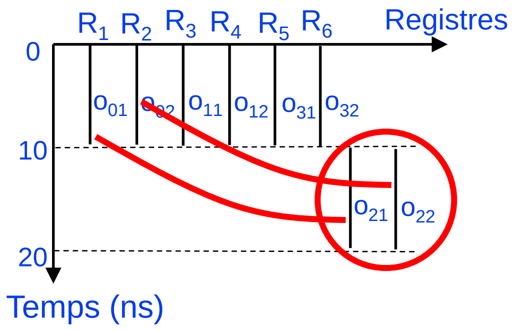
□ **Leftmost data are bound to distinct registers**

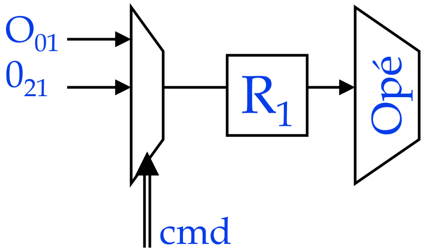



R_1 R_2 R_3 R_4 R_5 R_6
Registres
0
10
20
o_{01} o_{02} o_{11} o_{12} o_{31} o_{32}
o_{21} o_{22}
Temps (ns)

Data binding : the Left Edge algorithm







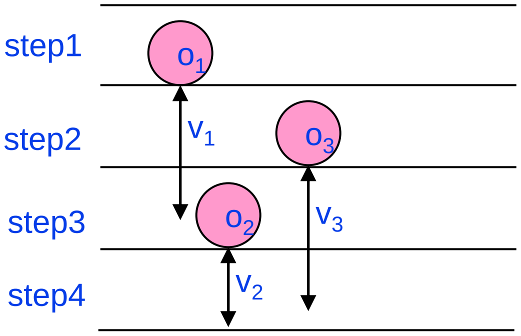
Left-edge algorithm does not take into account multiplexor cost

Resource Binding

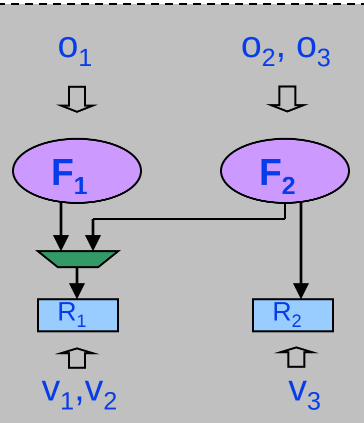
Multiplexer and interconnect costs are significant.

Cyclic inter-dependency exists between FU binding and register binding

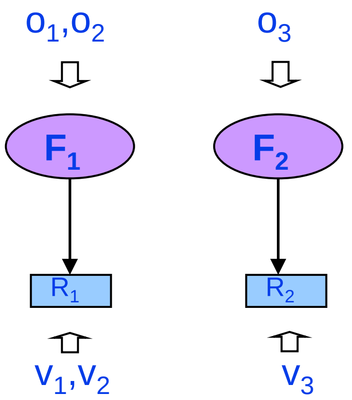
To minimize interconnection, one task needs the other’s result to make accurate decision



(1)A scheduling example



(2a) FU binding + REG binding



(2b) REG binding + FU binding

Resource constraints: 2 FUs, 2 REGs

The inter-dependency is far more complicated in real designs

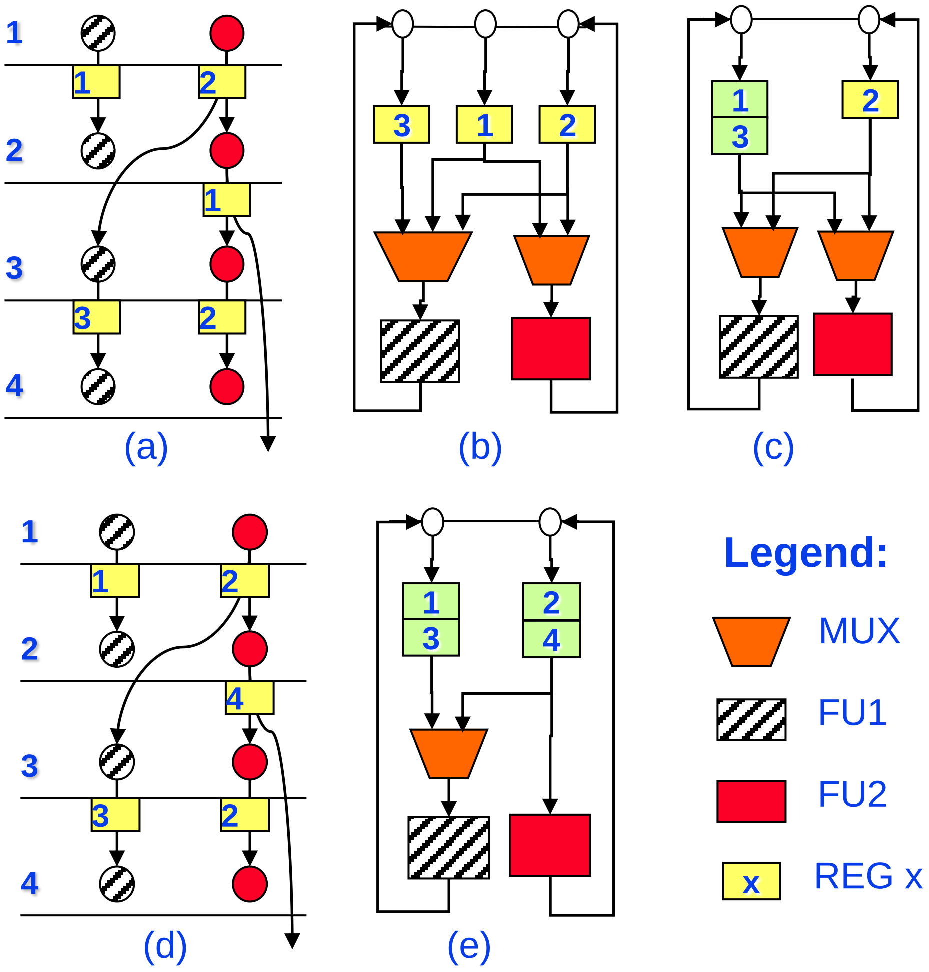
Use « manual allocation » to change FU binding arround the best point

Use a metaHeuristic: Variable Neighborhood Search, simulated annealing or Tabu search

Resource Binding

Register files may be used to hide the multiplexers, which are replaced by dedicated decoders

Merge registers with non-overlapping access dates



(a) (b) (c)

(d) (e)

Outline

- Lab-STICC
- General context
- High-Level Synthesis
  - Brief introduction
  - “In details”
- GAUT
  - Overview
  - Results
- Conclusion
- References

GAUT

- **An academic, free and open source HLS tool**
- **Dedicated to DSP applications**
  - Data-dominated algorithm
    - ***1D, 2D Filters***
    - ***Transforms (Fourrier, Hadamar, DCT…)***
    - ***Channel Coding, source coding algorithms***
- **Input : bit-accurate C/C++ algorithm**
  - bit-accurate integer and fixed-point from Mentor Graphics
- **Output : RTL Architecture**
  - VHDL
  - SystemC
    - ***CABA: Cycle accurate and Bit accurate***
    - ***TLM: Transaction level model***
    - ***Compatible with both SocLib and MPARM virtual prototyping platforms***
- **Automated Test-bench generation**
- **Automated operators characterization**

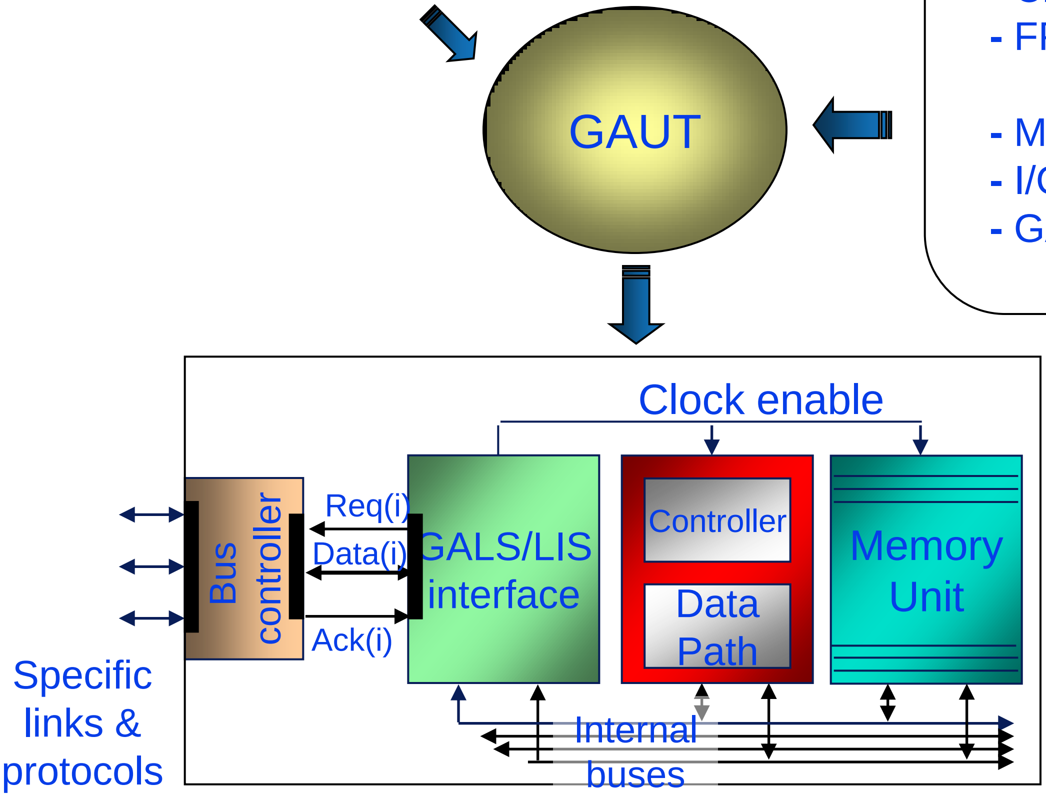
GAUT: Constraints

Bit accurate
Algorithm in bit-accurate C/C++

Synthesis constraints

- Initiation Interval (Data average throughput)
- Clock frequency
- FPGA/ASIC target technology

- Memory architecture and mapping
- I/O Timing diagram (scheduling + ports)
- GALS/LIS Interface (FIFO protocol)



GAUT: Design flow

Bit accu

Algorithm in bit-a

traints

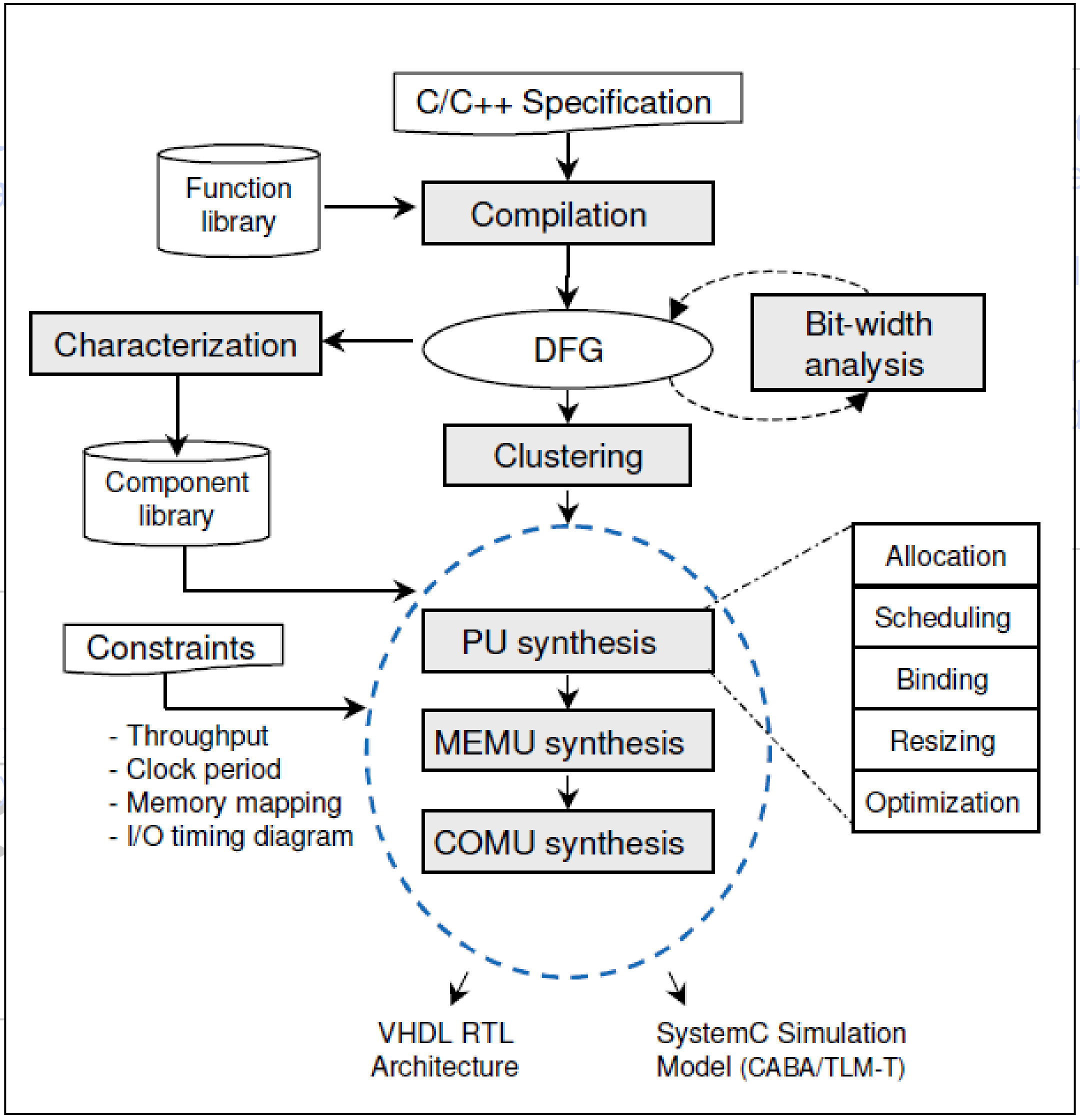
erage throughput)

logy

mapping

luling + ports)

protocol)



Bus controller

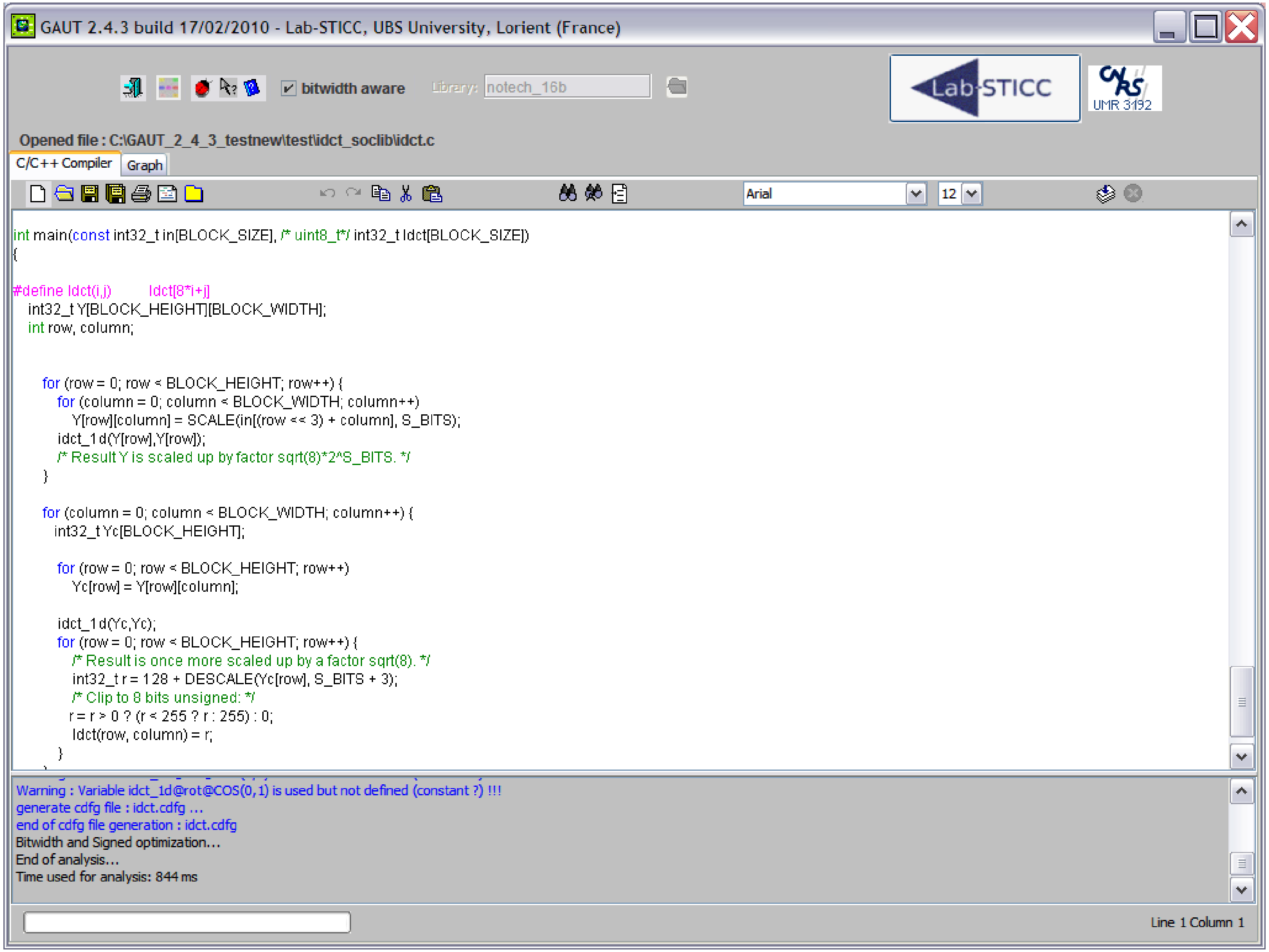
Req(i

Data(i)

Ack(i)

Specific links & protocols

GAUT: Compilation

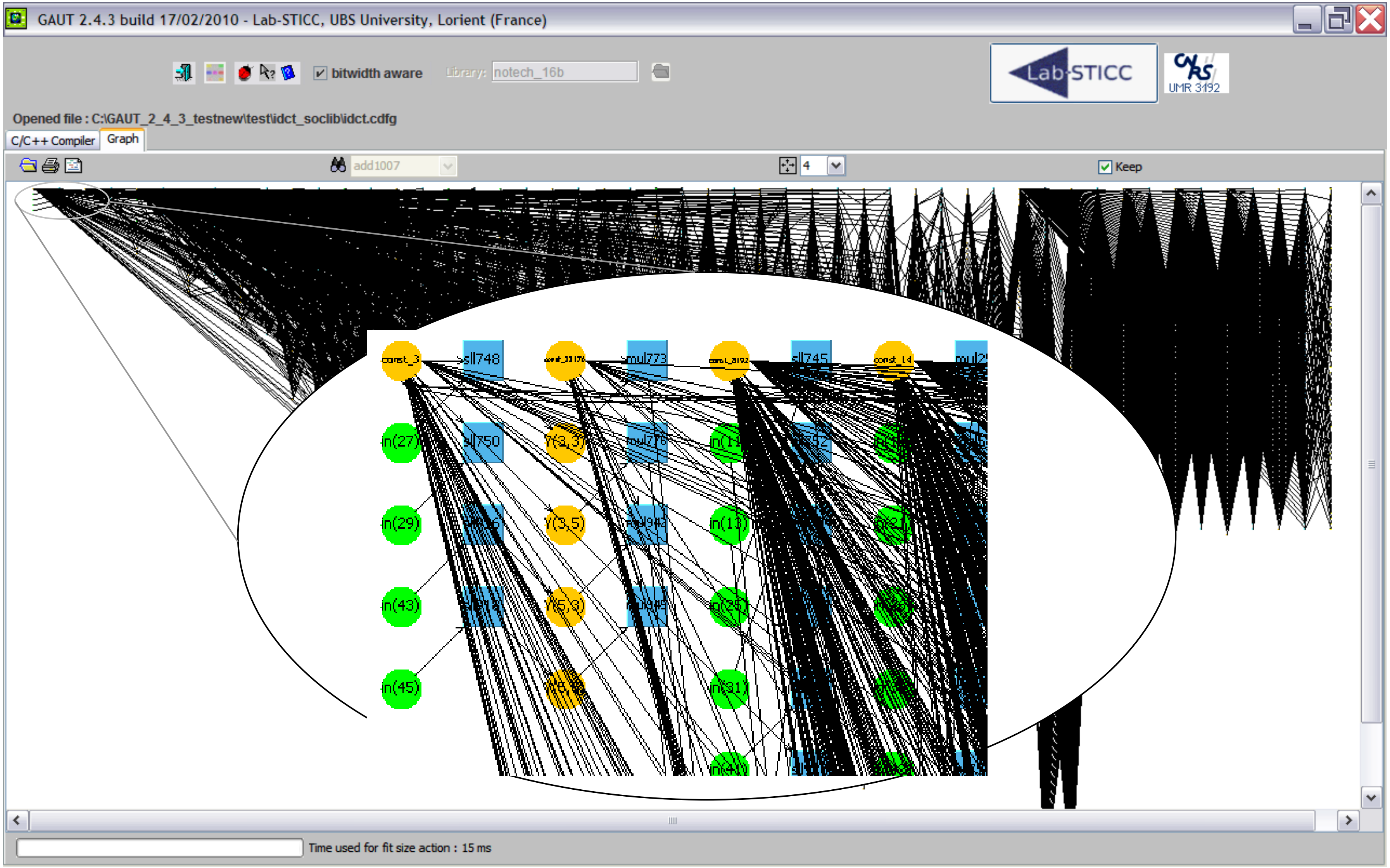


Opened file : C:\GAUT_2_4_3_testnew\test\idct_soclib\idct.c

```c
int main(const int32_t in[BLOCK_SIZE], /* uint8_t*/ int32_t Idct[BLOCK_SIZE])
{

#define Idct(i,j)       Idct[8*i+j]
   int32_t Y[BLOCK_HEIGHT][BLOCK_WIDTH];
   int row, column;


      for (row = 0; row < BLOCK_HEIGHT; row++) {
         for (column = 0; column < BLOCK_WIDTH; column++)
            Y[row][column] = SCALE(in[(row << 3) + column], S_BITS);
         idct_1d(Y[row],Y[row]);
         /* Result Y is scaled up by factor sqrt(8)*2^S_BITS. */
      }

      for (column = 0; column < BLOCK_WIDTH; column++) {
        int32_t Yc[BLOCK_HEIGHT];

         for (row = 0; row < BLOCK_HEIGHT; row++)
            Yc[row] = Y[row][column];

         idct_1d(Yc,Yc);
         for (row = 0; row < BLOCK_HEIGHT; row++) {
            /* Result is once more scaled up by a factor sqrt(8). */
            int32_t r = 128 + DESCALE(Yc[row], S_BITS + 3);
            /* Clip to 8 bits unsigned: */
           r = r > 0 ? (r < 255 ? r : 255) : 0;
            Idct(row, column) = r;
         }
```

```
Warning : Variable idct_1d@rot@COS(0,1) is used but not defined (constant ?) !!!
generate cdfg file : idct.cdfg ...
end of cdfg file generation : idct.cdfg
Bitwidth and Signed optimization...
End of analysis...
Time used for analysis: 844 ms
```

GAUT: DFG viewer



GAUT 2.4.3 build 17/02/2010 - Lab-STICC, UBS University, Lorient (France)

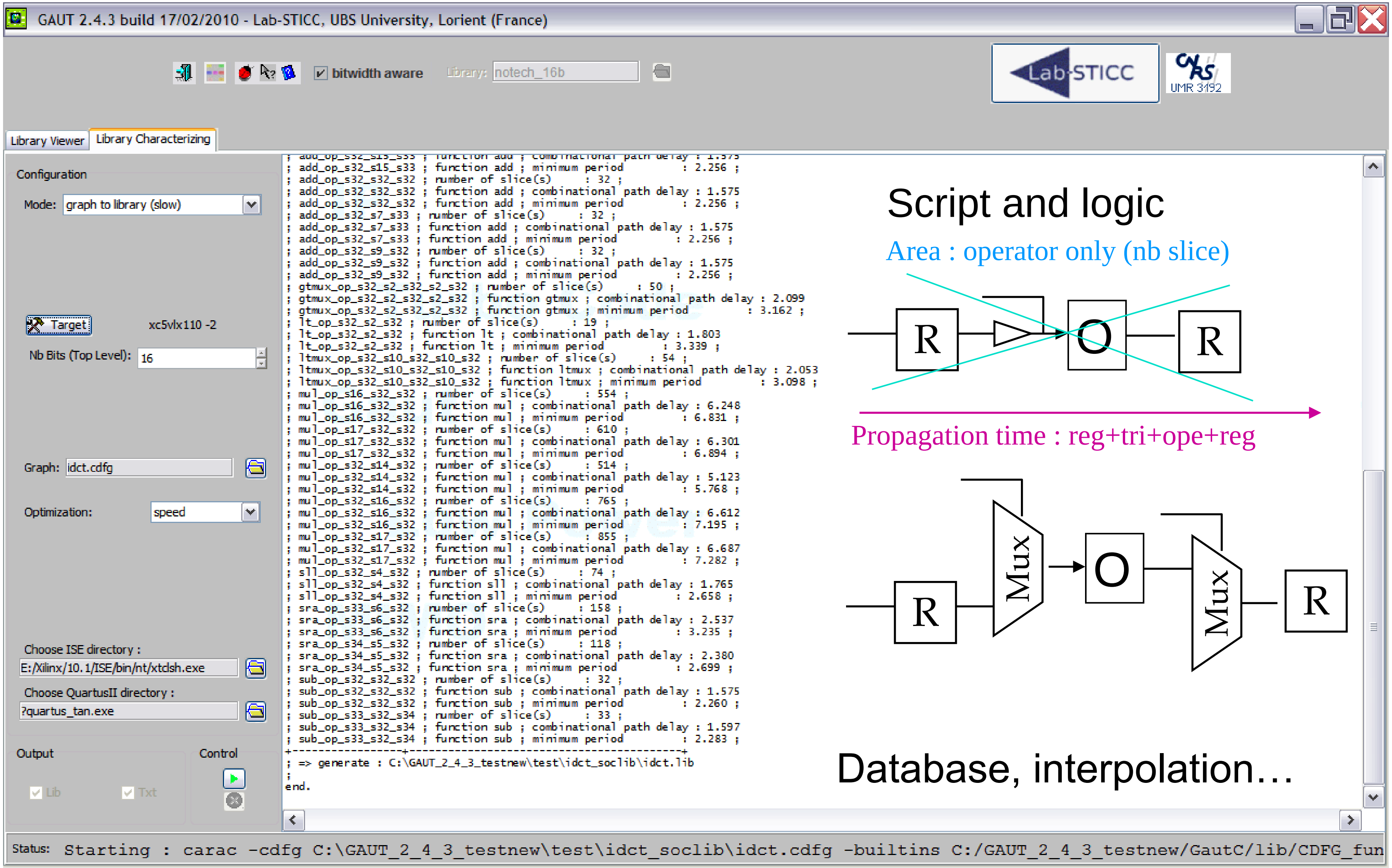
☑ bitwidth aware

Library: notech_16b

Opened file : C:\GAUT_2_4_3_testnew\test\idct_soclib\idct.cdfg

Time used for fit size action : 15 ms

GAUT: Operators characterization



GAUT 2.4.3 build 17/02/2010 - Lab-STICC, UBS University, Lorient (France)

☑ bitwidth aware

Library: notech_16b

Mode: graph to library (slow)

Target: xc5vlx110 -2

Nb Bits (Top Level): 16

Graph: idct.cdfg

Optimization: speed

Choose ISE directory : E:/Xilinx/10.1/ISE/bin/nt/xtdsh.exe

Choose QuartusII directory : ?quartus_tan.exe

☑ Lib ☑ Txt

```
; add_op_s32_s15_s33 ; function add ; combinational path delay : 1.575
; add_op_s32_s15_s33 ; function add ; minimum period          : 2.256 ;
; add_op_s32_s32_s32 ; number of slice(s)      : 32 ;
; add_op_s32_s32_s32 ; function add ; combinational path delay : 1.575
; add_op_s32_s32_s32 ; function add ; minimum period          : 2.256 ;
; add_op_s32_s7_s33 ; number of slice(s)      : 32 ;
; add_op_s32_s7_s33 ; function add ; combinational path delay : 1.575
; add_op_s32_s7_s33 ; function add ; minimum period          : 2.256 ;
; add_op_s32_s9_s32 ; number of slice(s)      : 32 ;
; add_op_s32_s9_s32 ; function add ; combinational path delay : 1.575
; add_op_s32_s9_s32 ; function add ; minimum period          : 2.256 ;
; gtmux_op_s32_s2_s32_s2_s32 ; number of slice(s)      : 50 ;
; gtmux_op_s32_s2_s32_s2_s32 ; function gtmux ; combinational path delay : 2.099
; gtmux_op_s32_s2_s32_s2_s32 ; function gtmux ; minimum period          : 3.162 ;
; lt_op_s32_s2_s32 ; number of slice(s)      : 19 ;
; lt_op_s32_s2_s32 ; function lt ; combinational path delay : 1.803
; lt_op_s32_s2_s32 ; function lt ; minimum period          : 3.339 ;
; ltmux_op_s32_s10_s32_s10_s32 ; number of slice(s)      : 54 ;
; ltmux_op_s32_s10_s32_s10_s32 ; function ltmux ; combinational path delay : 2.053
; ltmux_op_s32_s10_s32_s10_s32 ; function ltmux ; minimum period          : 3.098 ;
; mul_op_s16_s32_s32 ; number of slice(s)      : 554 ;
; mul_op_s16_s32_s32 ; function mul ; combinational path delay : 6.248
; mul_op_s16_s32_s32 ; function mul ; minimum period          : 6.831 ;
; mul_op_s17_s32_s32 ; number of slice(s)      : 610 ;
; mul_op_s17_s32_s32 ; function mul ; combinational path delay : 6.301
; mul_op_s17_s32_s32 ; function mul ; minimum period          : 6.894 ;
; mul_op_s32_s14_s32 ; number of slice(s)      : 514 ;
; mul_op_s32_s14_s32 ; function mul ; combinational path delay : 5.123
; mul_op_s32_s14_s32 ; function mul ; minimum period          : 5.768 ;
; mul_op_s32_s16_s32 ; number of slice(s)      : 765 ;
; mul_op_s32_s16_s32 ; function mul ; combinational path delay : 6.612
; mul_op_s32_s16_s32 ; function mul ; minimum period          : 7.195 ;
; mul_op_s32_s17_s32 ; number of slice(s)      : 855 ;
; mul_op_s32_s17_s32 ; function mul ; combinational path delay : 6.687
; mul_op_s32_s17_s32 ; function mul ; minimum period          : 7.282 ;
; sll_op_s32_s4_s32 ; number of slice(s)      : 74 ;
; sll_op_s32_s4_s32 ; function sll ; combinational path delay : 1.765
; sll_op_s32_s4_s32 ; function sll ; minimum period          : 2.658 ;
; sra_op_s33_s6_s32 ; number of slice(s)      : 158 ;
; sra_op_s33_s6_s32 ; function sra ; combinational path delay : 2.537
; sra_op_s33_s6_s32 ; function sra ; minimum period          : 3.235 ;
; sra_op_s34_s5_s32 ; number of slice(s)      : 118 ;
; sra_op_s34_s5_s32 ; function sra ; combinational path delay : 2.380
; sra_op_s34_s5_s32 ; function sra ; minimum period          : 2.699 ;
; sub_op_s32_s32_s32 ; number of slice(s)      : 32 ;
; sub_op_s32_s32_s32 ; function sub ; combinational path delay : 1.575
; sub_op_s32_s32_s32 ; function sub ; minimum period          : 2.260 ;
; sub_op_s33_s32_s34 ; number of slice(s)      : 33 ;
; sub_op_s33_s32_s34 ; function sub ; combinational path delay : 1.597
; sub_op_s33_s32_s34 ; function sub ; minimum period          : 2.283 ;
+------------------+----------------------------------------+
; => generate : C:\GAUT_2_4_3_testnew\test\idct_soclib\idct.lib
;
end.
```

```
Status: Starting : carac -cdfg C:\GAUT_2_4_3_testnew\test\idct_soclib\idct.cdfg -builtins C:/GAUT_2_4_3_testnew/GautC/lib/CDFG_fun
```

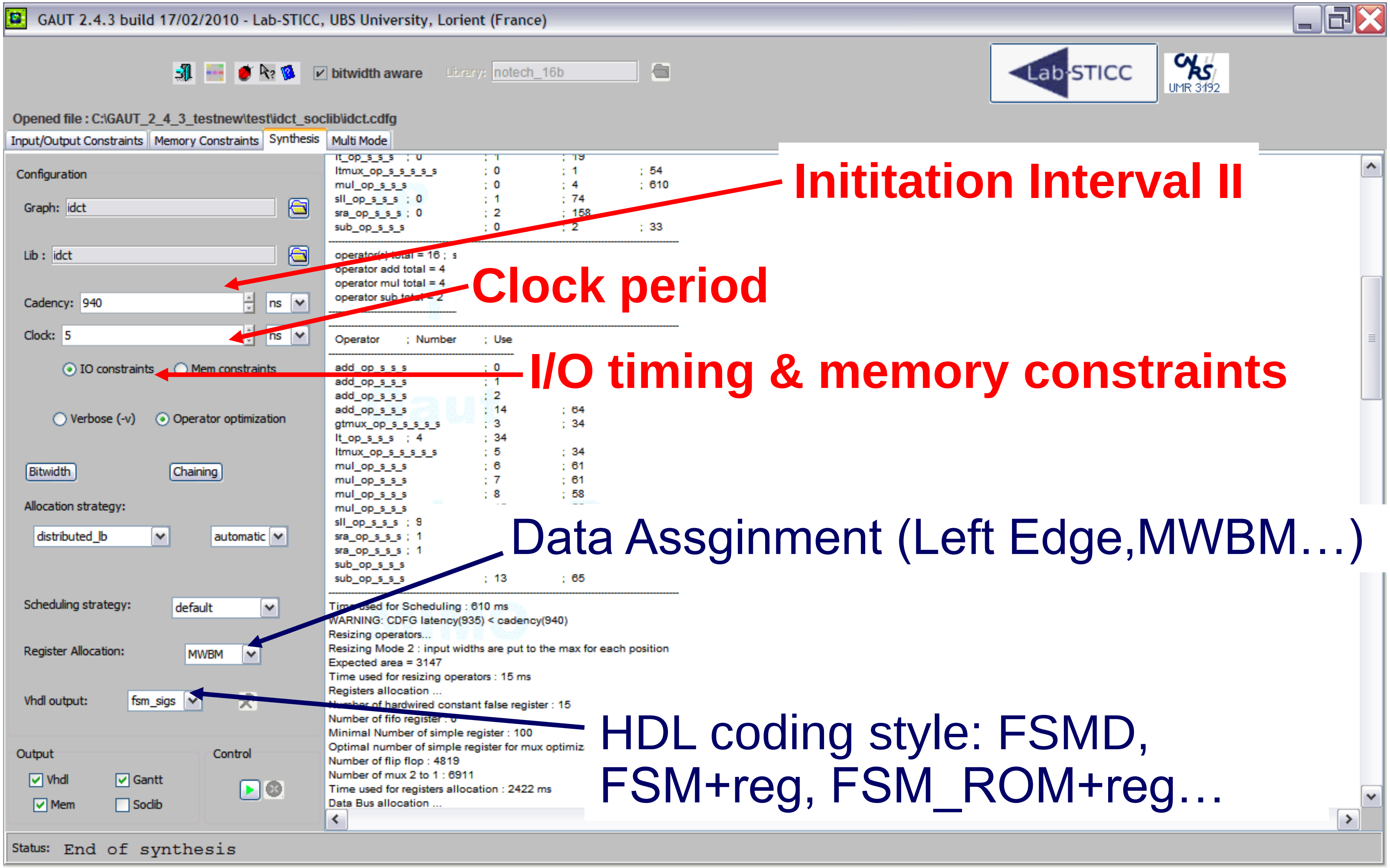
Script and logic

Area : operator only (nb slice)

Propagation time : reg+tri+ope+reg

Database, interpolation…

GAUT: Synthesis steps



GAUT 2.4.3 build 17/02/2010 - Lab-STICC, UBS University, Lorient (France)

☑ bitwidth aware   Library: notech_16b

Opened file : C:\GAUT_2_4_3_testnew\test\idct_soclib\idct.cdfg

Configuration

Graph: idct

Lib : idct

Cadency: 940 ns

Clock: 5 ns

● IO constraints ○ Mem constraints

○ Verbose (-v) ● Operator optimization

Allocation strategy: distributed_lb   automatic

Scheduling strategy: default

Register Allocation: MWBM

Vhdl output: fsm_sigs

Output: ☑ Vhdl ☑ Gantt ☑ Mem ☐ Soclib

```
lt_op_s_s_s   ;  0          ;  1          ;  19
ltmux_op_s_s_s_s_s          ;  0          ;  1          ;  54
mul_op_s_s_s                ;  0          ;  4          ;  610
sll_op_s_s_s  ;  0          ;  1          ;  74
sra_op_s_s_s  ;  0          ;  2          ;  158
sub_op_s_s_s                ;  0          ;  2          ;  33
------------------------------------------------------------
operator(s) total = 16 ;  s
operator add total = 4
operator mul total = 4
operator sub total = 2
---------------------------------

------------------------------------------------------------
Operator      ;  Number      ;  Use
-------------------------------------------
add_op_s_s_s                ;  0
add_op_s_s_s                ;  1
add_op_s_s_s                ;  2
add_op_s_s_s                ;  14         ;  64
gtmux_op_s_s_s_s_s          ;  3          ;  34
lt_op_s_s_s   ;  4          ;  34
ltmux_op_s_s_s_s_s          ;  5          ;  34
mul_op_s_s_s                ;  6          ;  61
mul_op_s_s_s                ;  7          ;  61
mul_op_s_s_s                ;  8          ;  58
mul_op_s_s_s
sll_op_s_s_s  ;  9
sra_op_s_s_s  ;  1
sra_op_s_s_s  ;  1
sub_op_s_s_s
sub_op_s_s_s                ;  13         ;  65
------------------------------------------------------------
Time used for Scheduling : 610 ms
WARNING: CDFG latency(935) < cadency(940)
Resizing operators...
Resizing Mode 2 : input widths are put to the max for each position
Expected area = 3147
Time used for resizing operators : 15 ms
Registers allocation ...
Number of hardwired constant false register : 15
Number of fifo register : 0
Minimal Number of simple register : 100
Optimal number of simple register for mux optimiz
Number of flip flop : 4819
Number of mux 2 to 1 : 6911
Time used for registers allocation : 2422 ms
Data Bus allocation ...
```

Status: End of synthesis

Initation Interval II

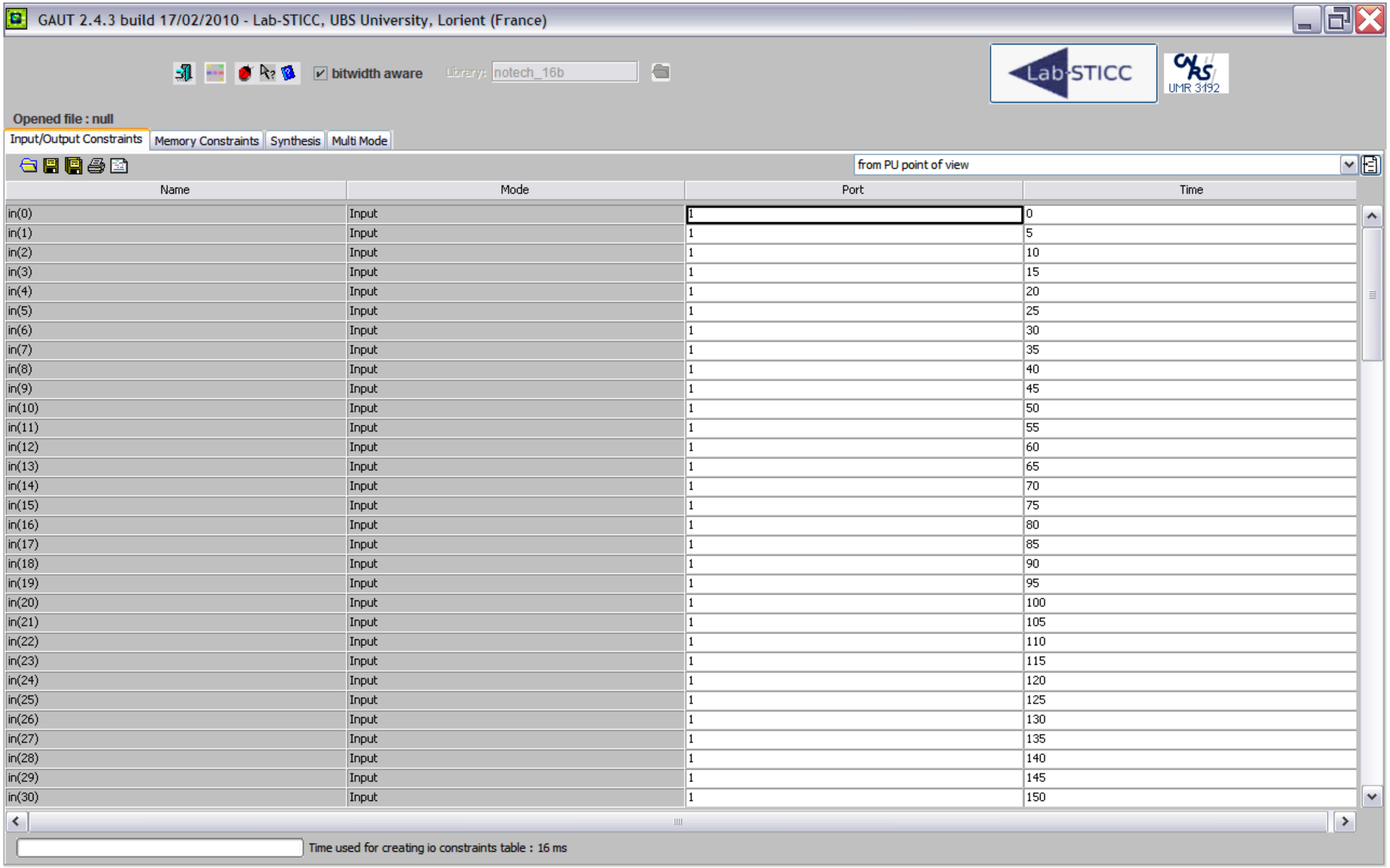
Clock period

I/O timing & memory constraints

Data Assginment (Left Edge,MWBM…)

HDL coding style: FSMD, FSM+reg, FSM_ROM+reg…

GAUT: I/O and memory constraints



GAUT 2.4.3 build 17/02/2010 - Lab-STICC, UBS University, Lorient (France)

☑ bitwidth aware

Library: notech_16b

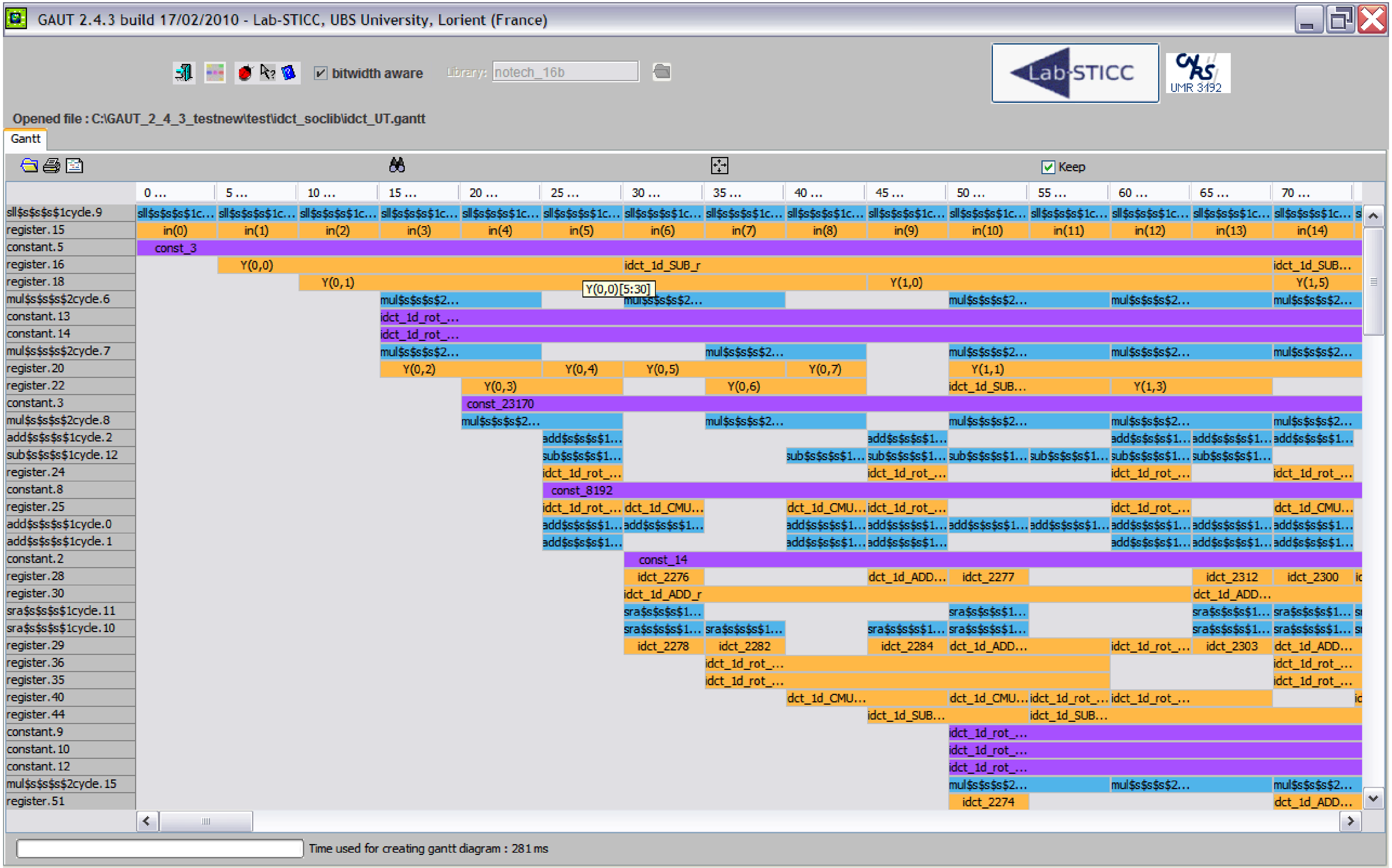
Opened file : null

from PU point of view

Name	Mode	Port	Time
in(0)	Input	1	0
in(1)	Input	1	5
in(2)	Input	1	10
in(3)	Input	1	15
in(4)	Input	1	20
in(5)	Input	1	25
in(6)	Input	1	30
in(7)	Input	1	35
in(8)	Input	1	40
in(9)	Input	1	45
in(10)	Input	1	50
in(11)	Input	1	55
in(12)	Input	1	60
in(13)	Input	1	65
in(14)	Input	1	70
in(15)	Input	1	75
in(16)	Input	1	80
in(17)	Input	1	85
in(18)	Input	1	90
in(19)	Input	1	95
in(20)	Input	1	100
in(21)	Input	1	105
in(22)	Input	1	110
in(23)	Input	1	115
in(24)	Input	1	120
in(25)	Input	1	125
in(26)	Input	1	130
in(27)	Input	1	135
in(28)	Input	1	140
in(29)	Input	1	145
in(30)	Input	1	150

Time used for creating io constraints table : 16 ms

GAUT: Gantt viewer



GAUT 2.4.3 build 17/02/2010 - Lab-STICC, UBS University, Lorient (France)

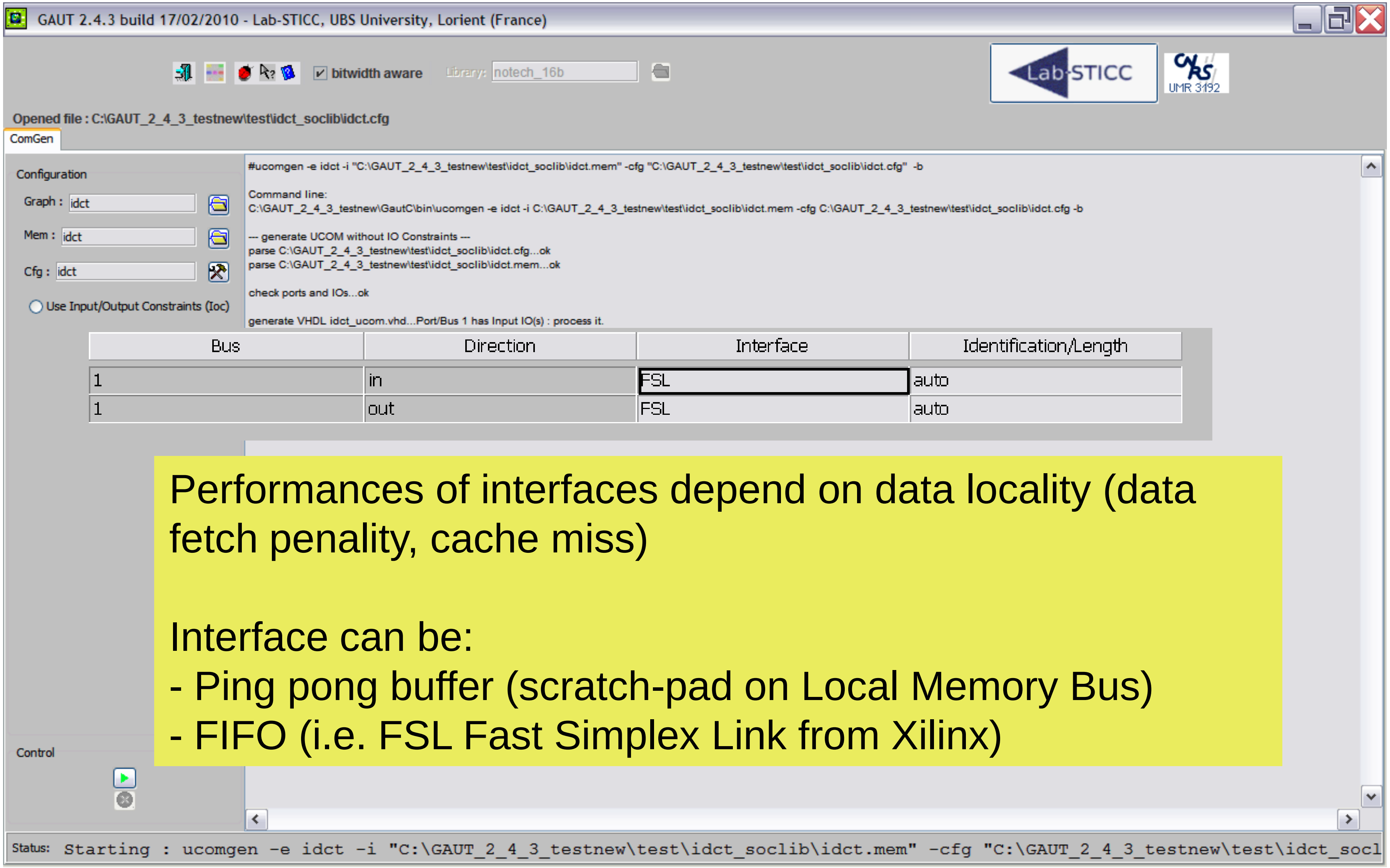
☑ bitwidth aware

Library: notech_16b

Opened file : C:\GAUT_2_4_3_testnew\test\idct_soclib\idct_UT.gantt

Time used for creating gantt diagram : 281 ms

GAUT: Interface synthesis



Opened file : C:\GAUT_2_4_3_testnew\test\idct_soclib\idct.cfg

Graph : idct

Mem : idct

Cfg : idct

○ Use Input/Output Constraints (Ioc)

```
#ucomgen -e idct -i "C:\GAUT_2_4_3_testnew\test\idct_soclib\idct.mem" -cfg "C:\GAUT_2_4_3_testnew\test\idct_soclib\idct.cfg"  -b

Command line:
C:\GAUT_2_4_3_testnew\GautC\bin\ucomgen -e idct -i C:\GAUT_2_4_3_testnew\test\idct_soclib\idct.mem -cfg C:\GAUT_2_4_3_testnew\test\idct_soclib\idct.cfg -b

--- generate UCOM without IO Constraints ---
parse C:\GAUT_2_4_3_testnew\test\idct_soclib\idct.cfg...ok
parse C:\GAUT_2_4_3_testnew\test\idct_soclib\idct.mem...ok

check ports and IOs...ok

generate VHDL idct_ucom.vhd...Port/Bus 1 has Input IO(s) : process it.
```

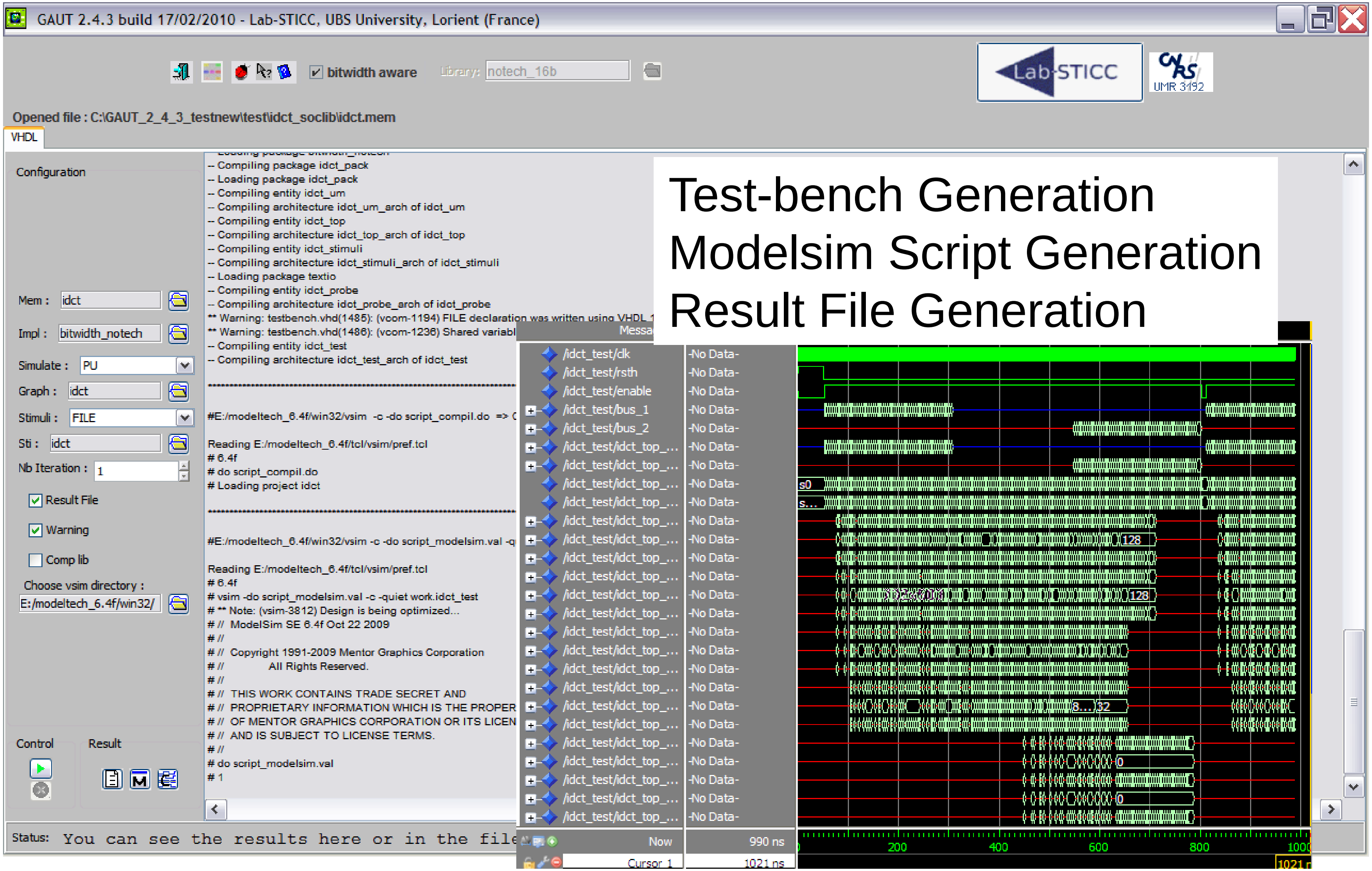
Bus	Direction	Interface	Identification/Length
1	in	FSL	auto
1	out	FSL	auto

Status: `Starting : ucomgen -e idct -i "C:\GAUT_2_4_3_testnew\test\idct_soclib\idct.mem" -cfg "C:\GAUT_2_4_3_testnew\test\idct_socl`

Performances of interfaces depend on data locality (data fetch penality, cache miss)

Interface can be:
- Ping pong buffer (scratch-pad on Local Memory Bus)
- FIFO (i.e. FSL Fast Simplex Link from Xilinx)

GAUT: Test-bench generation



GAUT 2.4.3 build 17/02/2010 - Lab-STICC, UBS University, Lorient (France)

☑ bitwidth aware   Library: notech_16b

Opened file : C:\GAUT_2_4_3_testnew\test\idct_soclib\idct.mem

VHDL

Configuration

- Mem : idct
- Impl : bitwidth_notech
- Simulate : PU
- Graph : idct
- Stimuli : FILE
- Sti : idct
- Nb Iteration : 1

☑ Result File

☑ Warning

☐ Comp lib

Choose vsim directory : E:/modeltech_6.4f/win32/

Control   Result

```
-- Compiling package idct_pack
-- Loading package idct_pack
-- Compiling entity idct_um
-- Compiling architecture idct_um_arch of idct_um
-- Compiling entity idct_top
-- Compiling architecture idct_top_arch of idct_top
-- Compiling entity idct_stimuli
-- Compiling architecture idct_stimuli_arch of idct_stimuli
-- Loading package textio
-- Compiling entity idct_probe
-- Compiling architecture idct_probe_arch of idct_probe
** Warning: testbench.vhd(1485): (vcom-1194) FILE declaration was written using VHDL 1
** Warning: testbench.vhd(1486): (vcom-1236) Shared variabl
-- Compiling entity idct_test
-- Compiling architecture idct_test_arch of idct_test

****************************************************************************

#E:/modeltech_6.4f/win32/vsim  -c -do script_compil.do  => (

Reading E:/modeltech_6.4f/tcl/vsim/pref.tcl
# 6.4f
# do script_compil.do
# Loading project idct

****************************************************************************

#E:/modeltech_6.4f/win32/vsim -c -do script_modelsim.val -q

Reading E:/modeltech_6.4f/tcl/vsim/pref.tcl
# 6.4f
# vsim -do script_modelsim.val -c -quiet work.idct_test
# ** Note: (vsim-3812) Design is being optimized...
# //  ModelSim SE 6.4f Oct 22 2009
# //
# //  Copyright 1991-2009 Mentor Graphics Corporation
# //              All Rights Reserved.
# //
# //  THIS WORK CONTAINS TRADE SECRET AND
# //  PROPRIETARY INFORMATION WHICH IS THE PROPER
# //  OF MENTOR GRAPHICS CORPORATION OR ITS LICEN
# //  AND IS SUBJECT TO LICENSE TERMS.
# //
# do script_modelsim.val
# 1
```

Status: You can see the results here or in the file

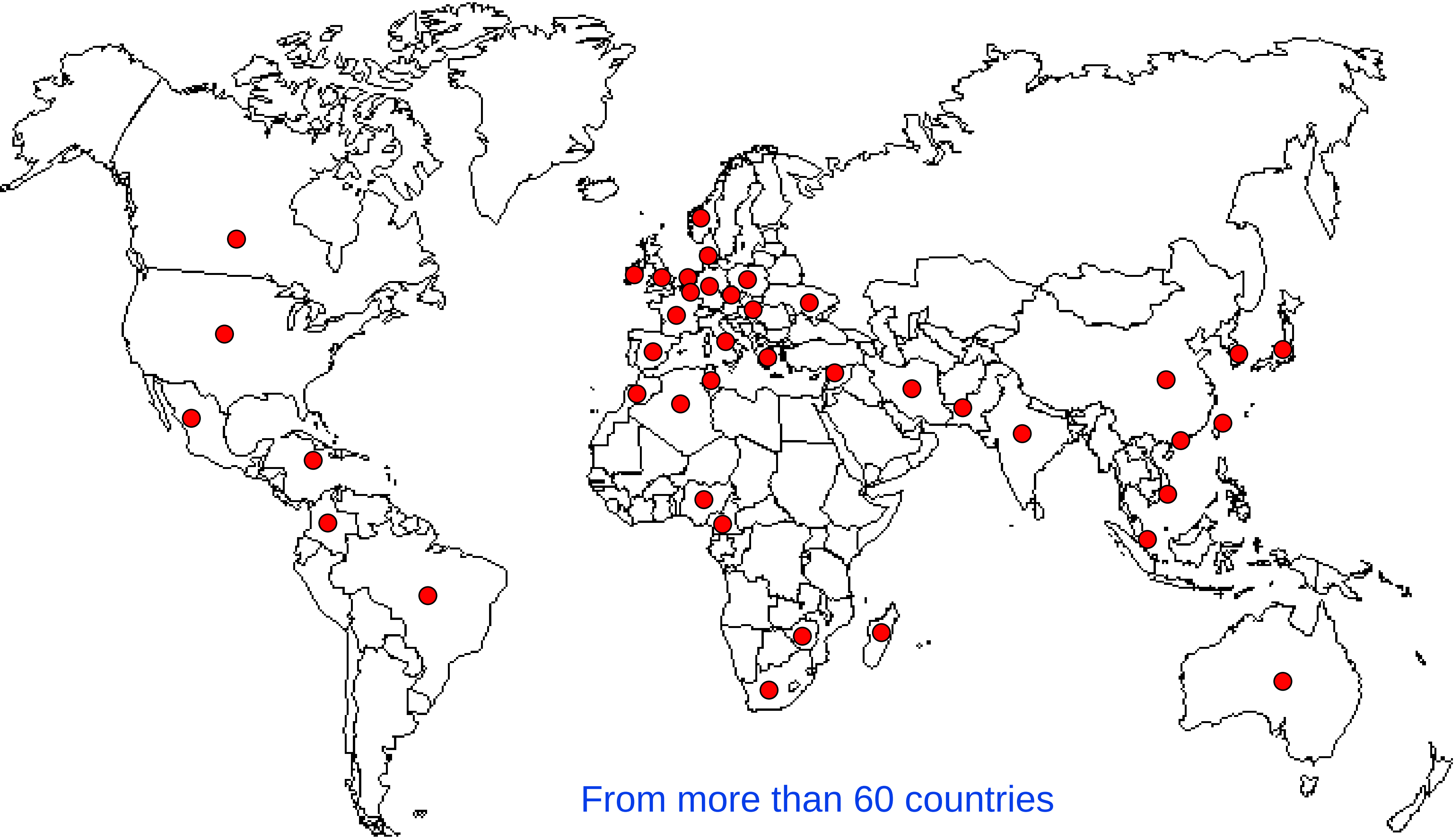
```
Messa
/idct_test/clk            -No Data-
/idct_test/rsth           -No Data-
/idct_test/enable         -No Data-
/idct_test/bus_1          -No Data-
/idct_test/bus_2          -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
/idct_test/idct_top_...   -No Data-
Now        990 ns
Cursor 1   1021 ns
```

Test-bench Generation

Modelsim Script Generation

Result File Generation

GAUT: more than 100 downloads each year

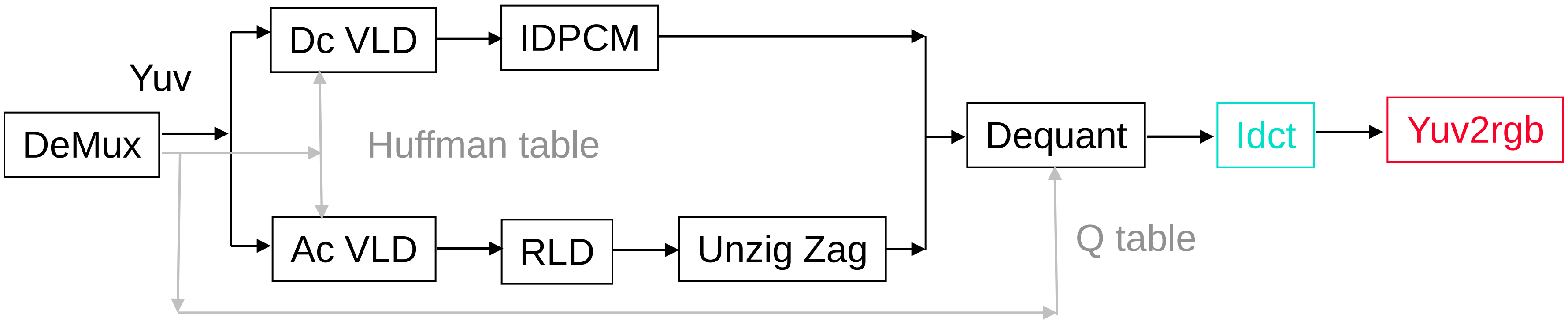


From more than 60 countries

Outline

- □ Lab-STICC
- □ General context
- □ High-Level Synthesis
  - ■ Brief introduction
  - ■ “In details”
- □ GAUT
  - ■ Overview
  - ■ Results
- □ Conclusion
- □ References

Experimental results: MJPEG decoding



Yuv

DeMux

Dc VLD

IDPCM

Huffman table

Ac VLD

RLD

Unzig Zag

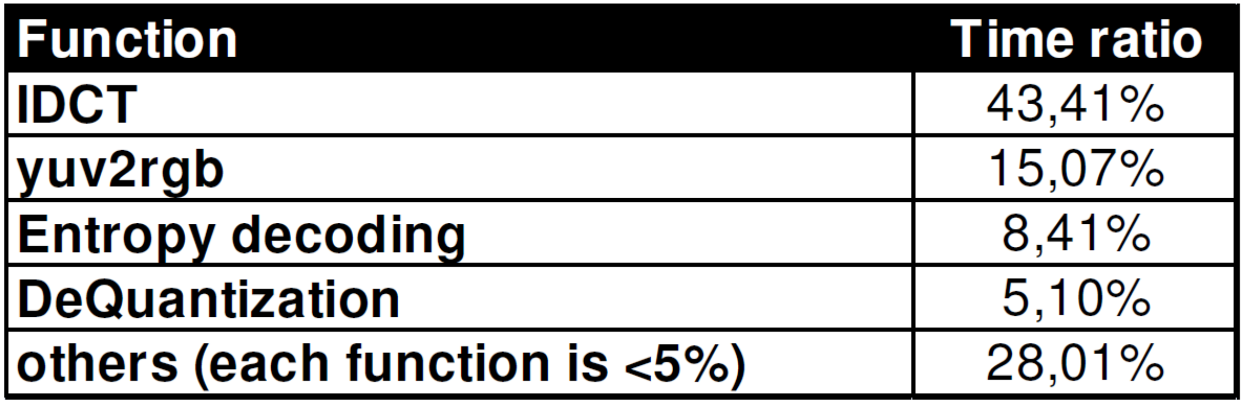
Dequant

Q table

Idct

Yuv2rgb

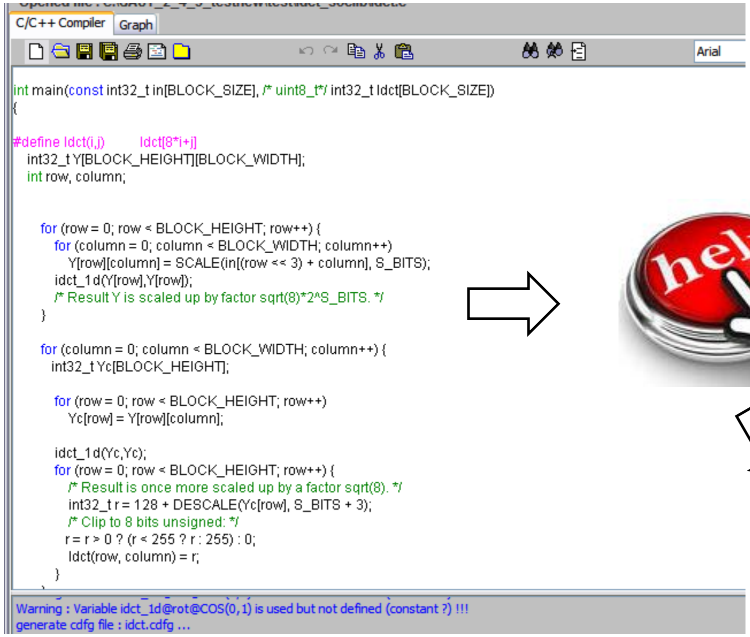
Block Diagram of mjpeg baseline decoder



Function	Time ratio
IDCT	43,41%
yuv2rgb	15,07%
Entropy decoding	8,41%
DeQuantization	5,10%
others (each function is <5%)	28,01%

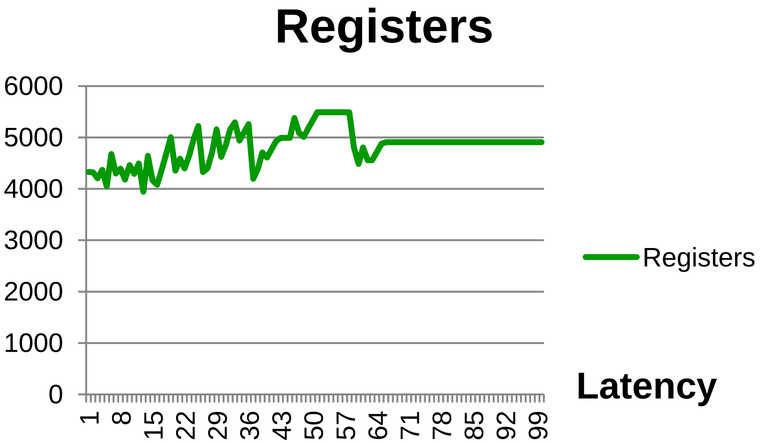
Execution time ratio for software MJPEG decoding
(by using gprof)

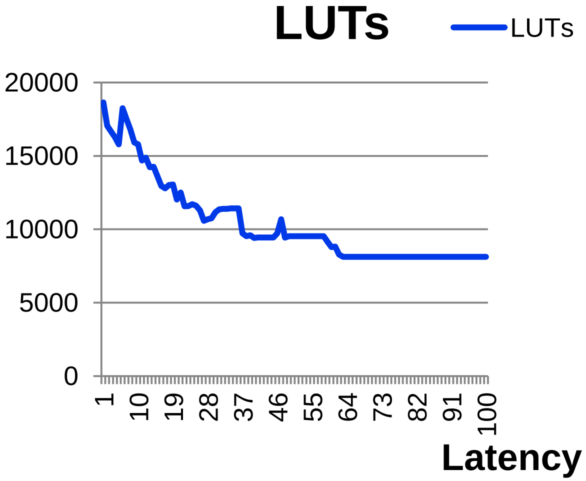
Resource estimation for IDCT

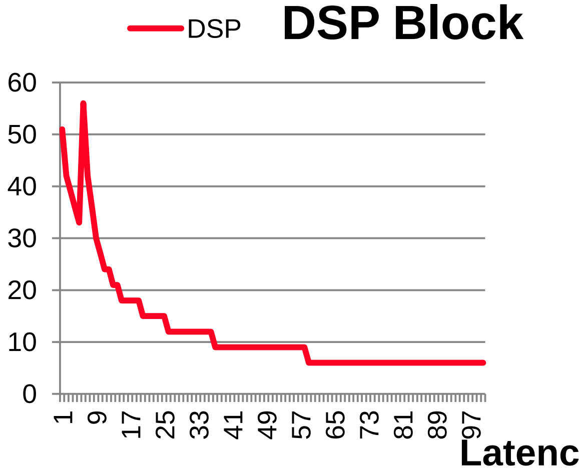


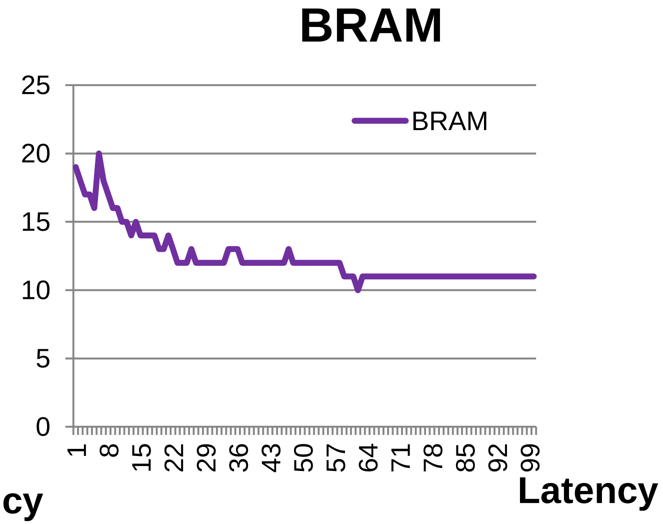
```
int main(const int32_t in[BLOCK_SIZE], /* uint8_t*/ int32_t Idct[BLOCK_SIZE])
{

#define Idct(i,j)        Idct[8*i+j]
   int32_t Y[BLOCK_HEIGHT][BLOCK_WIDTH];
   int row, column;


      for (row = 0; row < BLOCK_HEIGHT; row++) {
         for (column = 0; column < BLOCK_WIDTH; column++)
            Y[row][column] = SCALE(in[(row << 3) + column], S_BITS);
         idct_1d(Y[row],Y[row]);
         /* Result Y is scaled up by factor sqrt(8)*2^S_BITS. */
      }

      for (column = 0; column < BLOCK_WIDTH; column++) {
        int32_t Yc[BLOCK_HEIGHT];

         for (row = 0; row < BLOCK_HEIGHT; row++)
            Yc[row] = Y[row][column];

         idct_1d(Yc,Yc);
         for (row = 0; row < BLOCK_HEIGHT; row++) {
            /* Result is once more scaled up by a factor sqrt(8). */
            int32_t r = 128 + DESCALE(Yc[row], S_BITS + 3);
            /* Clip to 8 bits unsigned: */
           r = r > 0 ? (r < 255 ? r : 255) : 0;
            Idct(row, column) = r;
         }
```

```
Warning : Variable idct_1d@rot@COS(0,1) is used but not defined (constant ?) !!!
generate cdfg file : idct.cdfg ...
```

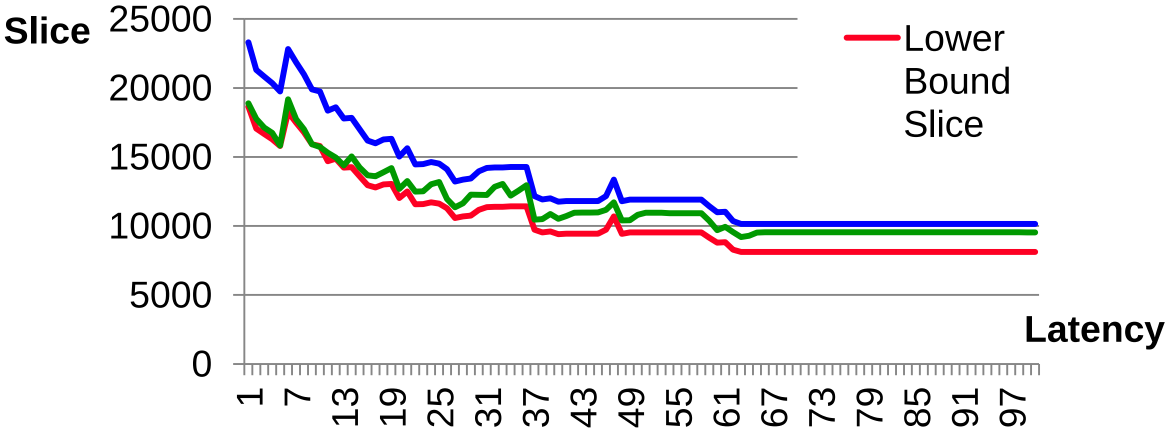


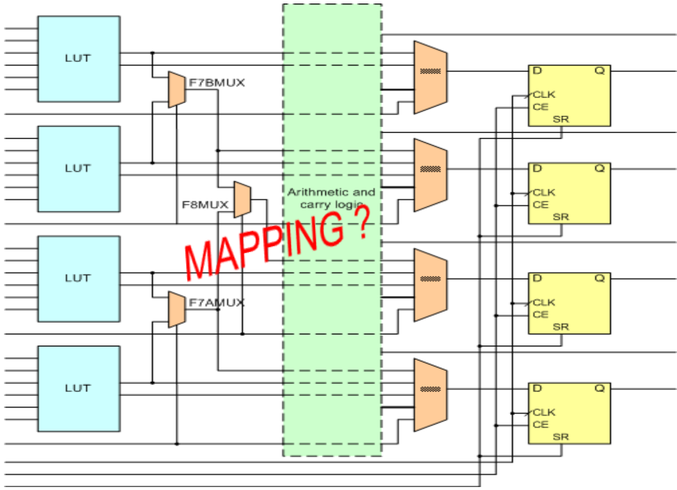






Resource estimation for IDCT

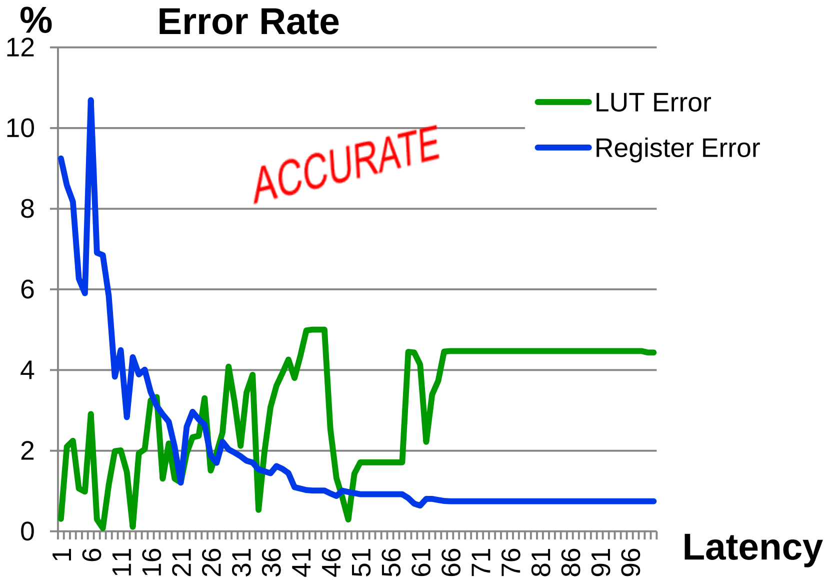




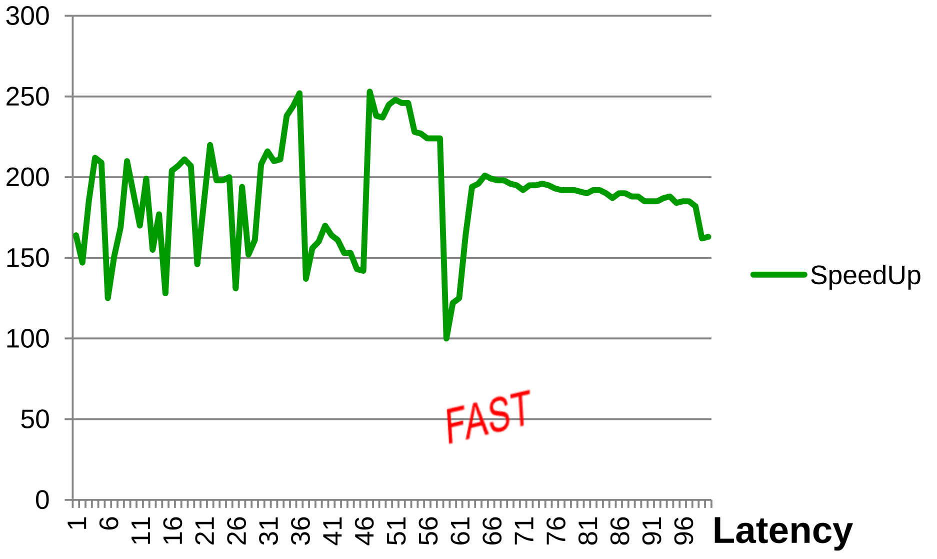
MAPPING ?

SpeedUp = Logic Synthesis Time/HLS Synthesis Time

Error Rate

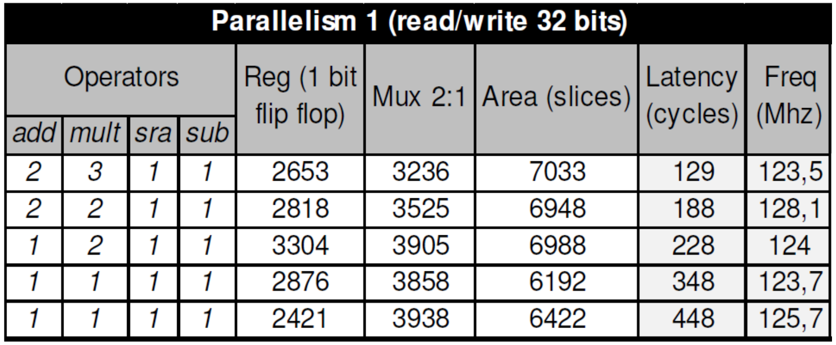


ACCURATE

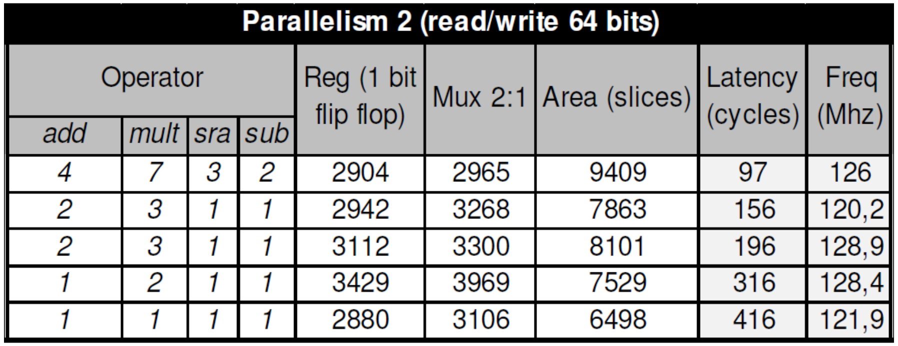


FAST

Synthesis results

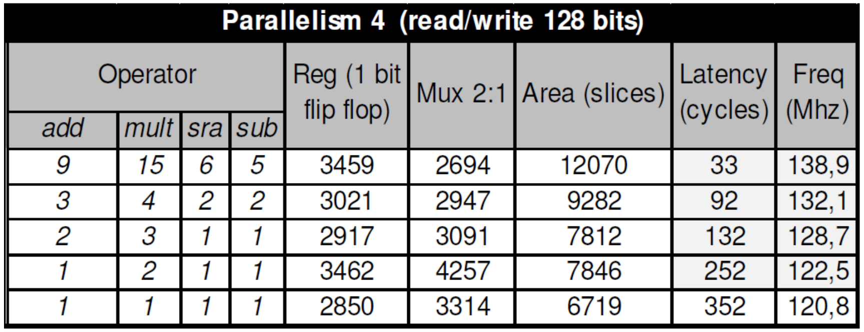


Parallelism 1 (read/write 32 bits)								
Operators				Reg (1 bit flip flop)	Mux 2:1	Area (slices)	Latency (cycles)	Freq (Mhz)
add	*mult*	*sra*	*sub*					
2	*3*	*1*	*1*	2653	3236	7033	129	123,5
2	*2*	*1*	*1*	2818	3525	6948	188	128,1
1	*2*	*1*	*1*	3304	3905	6988	228	124
1	*1*	*1*	*1*	2876	3858	6192	348	123,7
1	*1*	*1*	*1*	2421	3938	6422	448	125,7



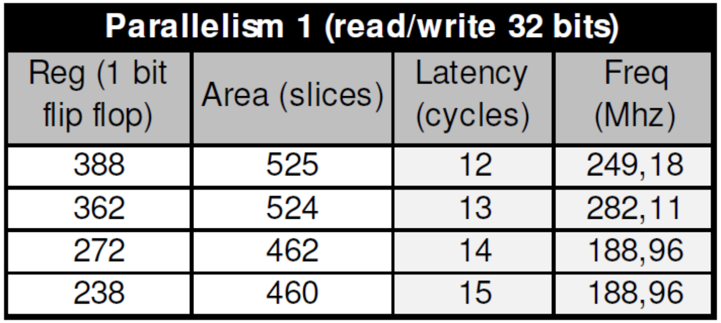
Parallelism 2 (read/write 64 bits)								
Operator				Reg (1 bit flip flop)	Mux 2:1	Area (slices)	Latency (cycles)	Freq (Mhz)
add	*mult*	*sra*	*sub*					
4	*7*	*3*	*2*	2904	2965	9409	97	126
2	*3*	*1*	*1*	2942	3268	7863	156	120,2
2	*3*	*1*	*1*	3112	3300	8101	196	128,9
1	*2*	*1*	*1*	3429	3969	7529	316	128,4
1	*1*	*1*	*1*	2880	3106	6498	416	121,9

IDCT



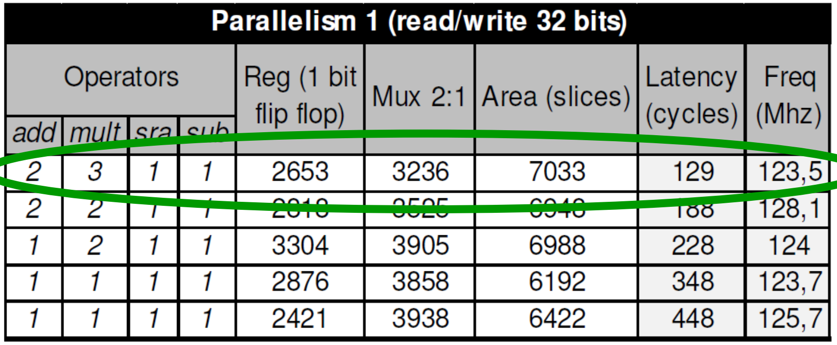
Parallelism 4 (read/write 128 bits)								
Operator				Reg (1 bit flip flop)	Mux 2:1	Area (slices)	Latency (cycles)	Freq (Mhz)
add	*mult*	*sra*	*sub*					
9	*15*	*6*	*5*	3459	2694	12070	33	138,9
3	*4*	*2*	*2*	3021	2947	9282	92	132,1
2	*3*	*1*	*1*	2917	3091	7812	132	128,7
1	*2*	*1*	*1*	3462	4257	7846	252	122,5
1	*1*	*1*	*1*	2850	3314	6719	352	120,8

YUV2RGB



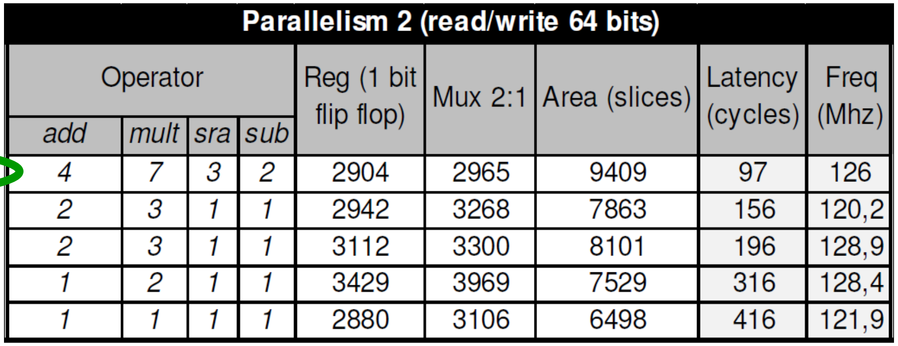
Parallelism 1 (read/write 32 bits)			
Reg (1 bit flip flop)	Area (slices)	Latency (cycles)	Freq (Mhz)
388	525	12	249,18
362	524	13	282,11
272	462	14	188,96
238	460	15	188,96

Synthesis results



Parallelism 1 (read/write 32 bits)

Operators				Reg (1 bit flip flop)	Mux 2:1	Area (slices)	Latency (cycles)	Freq (Mhz)
add	*mult*	*sra*	*sub*					
2	*3*	*1*	*1*	2653	3236	7033	129	123,5
2	*2*	*1*	*1*	2818	3525	6948	188	128,1
1	*2*	*1*	*1*	3304	3905	6988	228	124
1	*1*	*1*	*1*	2876	3858	6192	348	123,7
1	*1*	*1*	*1*	2421	3938	6422	448	125,7

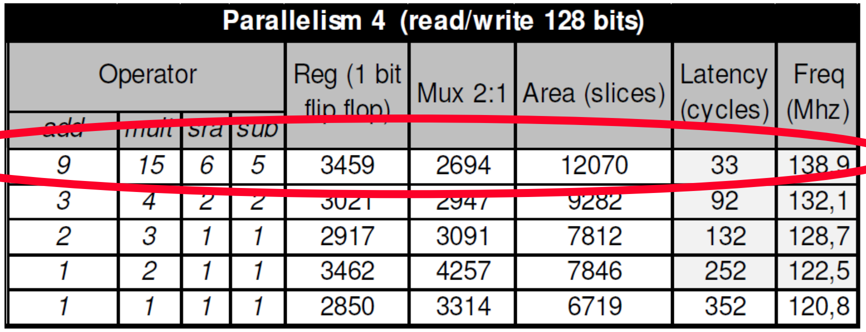


Parallelism 2 (read/write 64 bits)

Operator				Reg (1 bit flip flop)	Mux 2:1	Area (slices)	Latency (cycles)	Freq (Mhz)
add	*mult*	*sra*	*sub*					
4	*7*	*3*	*2*	2904	2965	9409	97	126
2	*3*	*1*	*1*	2942	3268	7863	156	120,2
2	*3*	*1*	*1*	3112	3300	8101	196	128,9
1	*2*	*1*	*1*	3429	3969	7529	316	128,4
1	*1*	*1*	*1*	2880	3106	6498	416	121,9

Virtual prototyping

IDCT

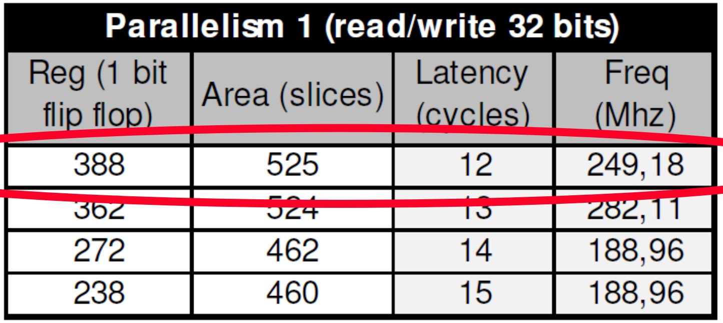


Parallelism 4 (read/write 128 bits)

Operator				Reg (1 bit flip flop)	Mux 2:1	Area (slices)	Latency (cycles)	Freq (Mhz)
add	*mult*	*sra*	*sub*					
9	*15*	*6*	*5*	3459	2694	12070	33	138,9
3	*4*	*2*	*2*	3021	2947	9282	92	132,1
2	*3*	*1*	*1*	2917	3091	7812	132	128,7
1	*2*	*1*	*1*	3462	4257	7846	252	122,5
1	*1*	*1*	*1*	2850	3314	6719	352	120,8

Hardware prototyping

YUV2RGB



Parallelism 1 (read/write 32 bits)

Reg (1 bit flip flop)	Area (slices)	Latency (cycles)	Freq (Mhz)
388	525	12	249,18
362	524	13	282,11
272	462	14	188,96
238	460	15	188,96

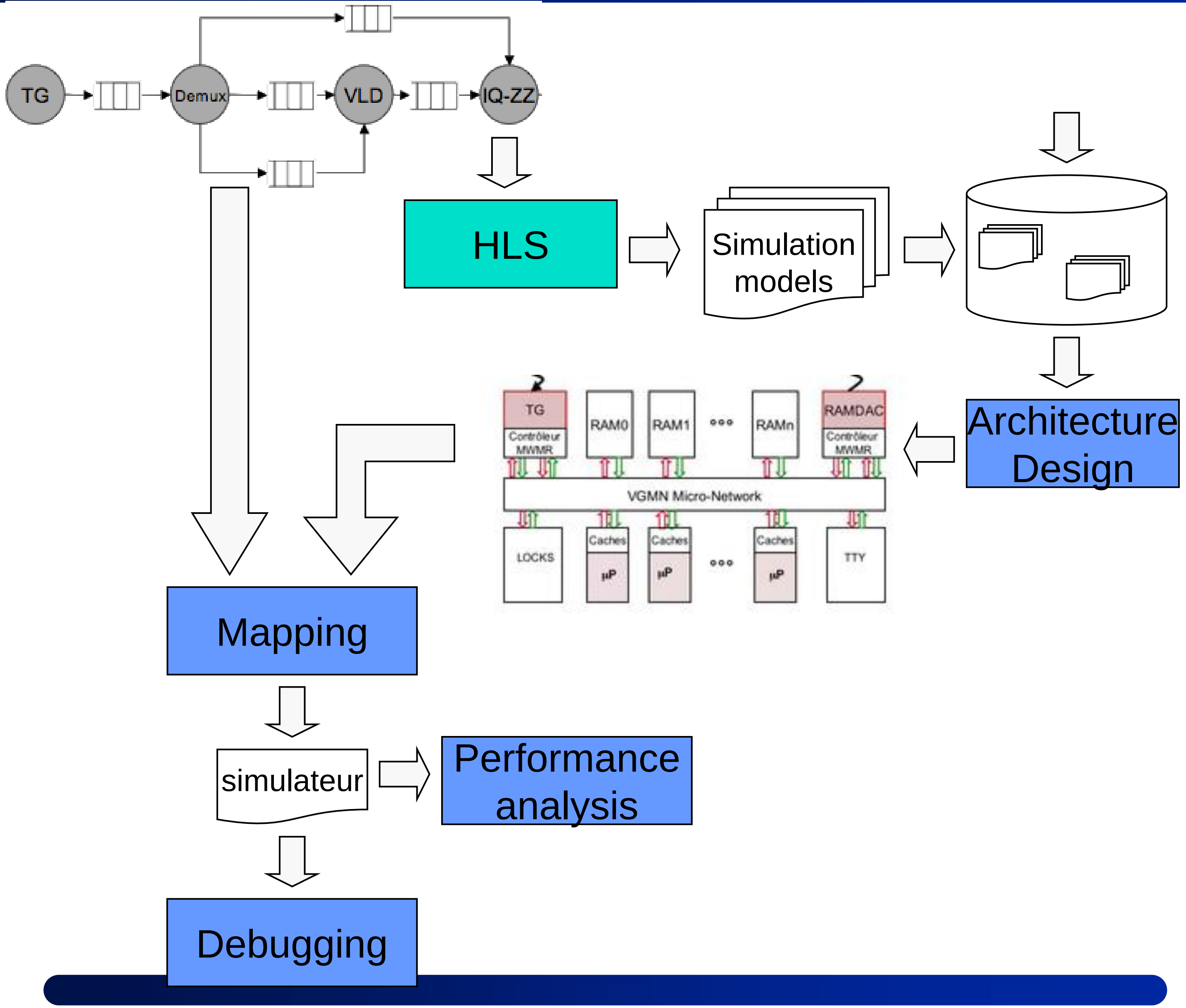
SoCLib: a virtual prototyping platform

- French National Research Project (ANR)
- Free and open source virtual prototyping environment
  - **Library of SystemC simulation models**
  - **Hardware components**
    - *CPUs, HW-ACCs, memories, busses*
    - *VCI/OCP interface protocol is used*
  - **Two types of model are available for each HW component**
    - *CABA (Cycle Accurate / Bit Accurate)*
    - *TLM-DT (Transaction Level Modeling with Distributed Time)*
  - **Software components**
    - *OS, API…*
  - **Associated tools**
    - *Simulation, configuration, debug*
    - *Automatic generation of simulation models*

www.soclib.fr

- GAUT is used, to generate simulation models of HW-ACC
  - **CABA and TLM-DT**

SoCLib: Design flow



HLS

Simulation models

Architecture Design

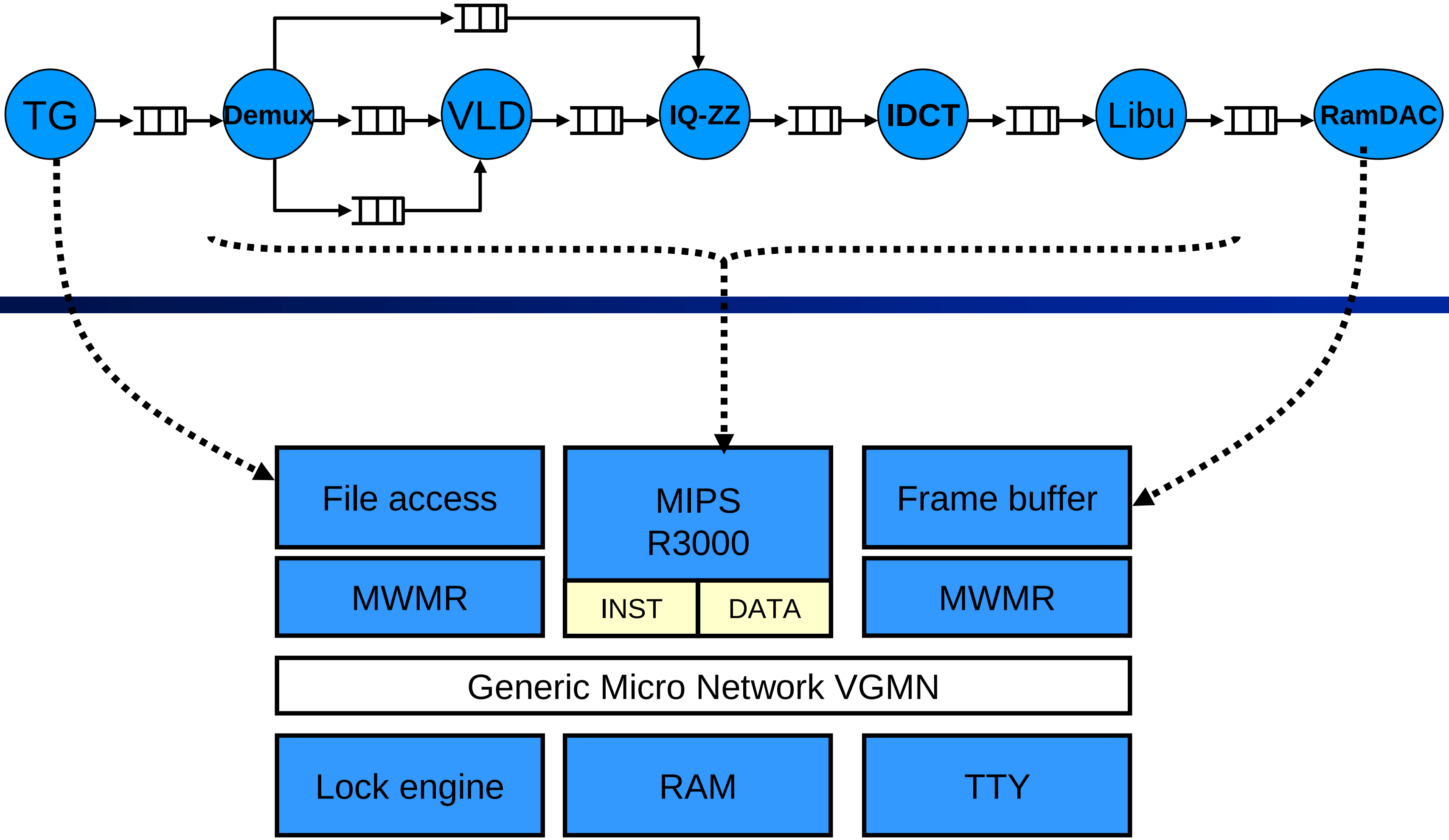
Mapping

simulateur

Performance analysis

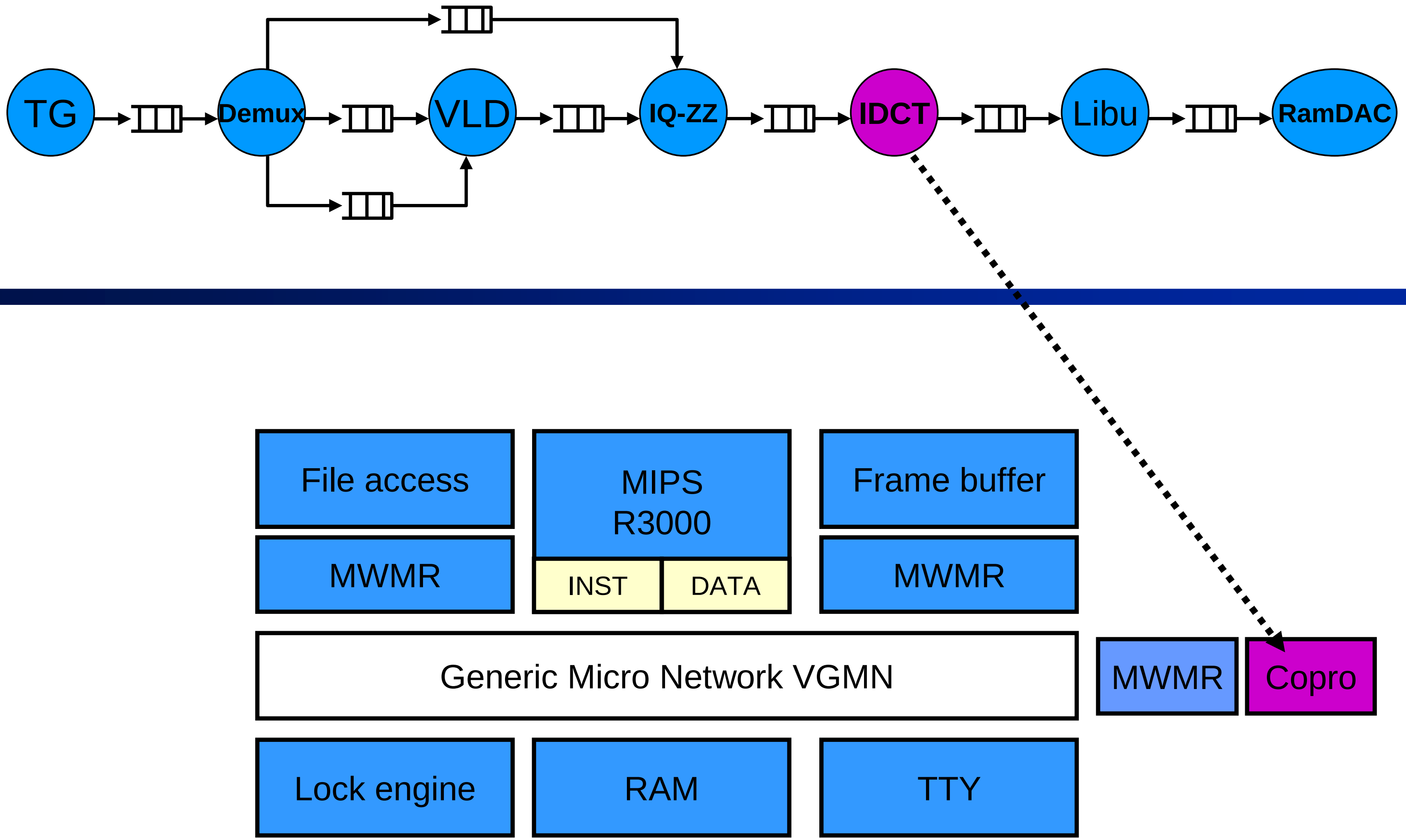
Debugging

Experiments: architecture #1



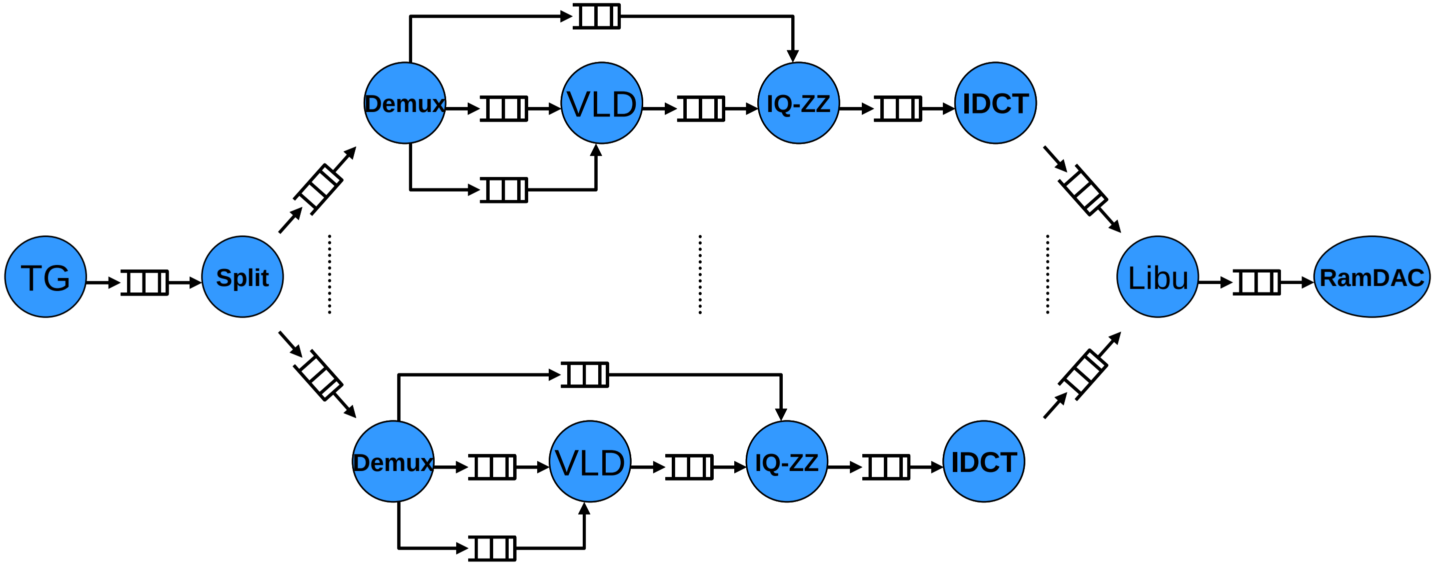
Pure software implementation on a mono-processor architecture

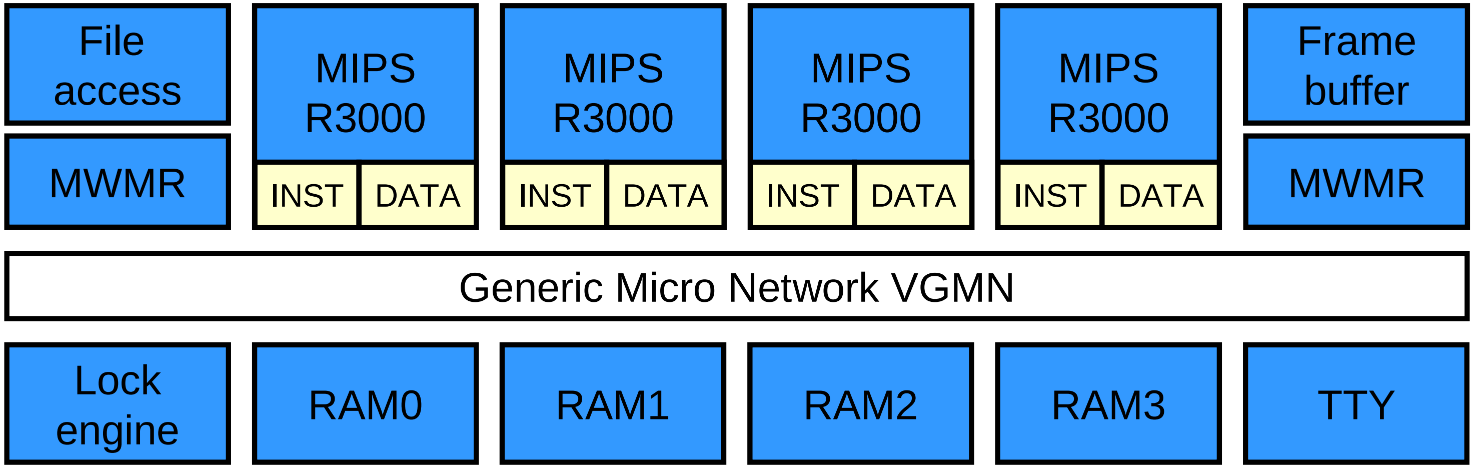
Experiments: architecture #2



Software implementation on a mono-processor architecture
+ IDCT as HW accelerator

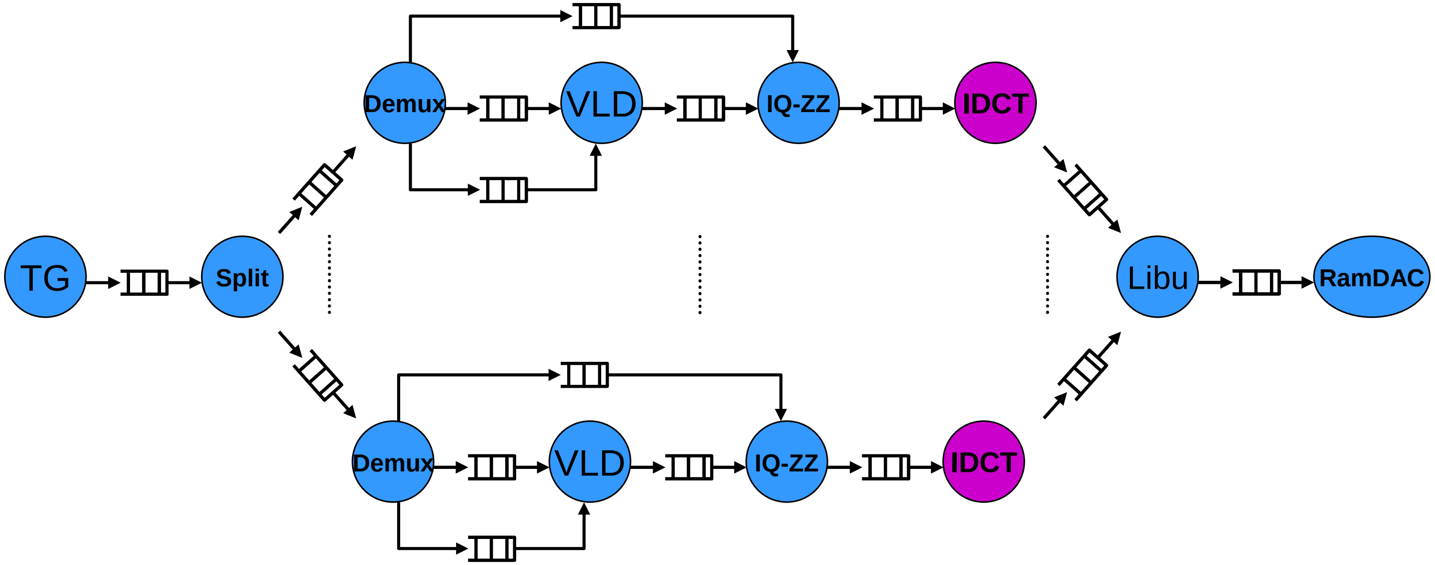
Experiments: architecture #3

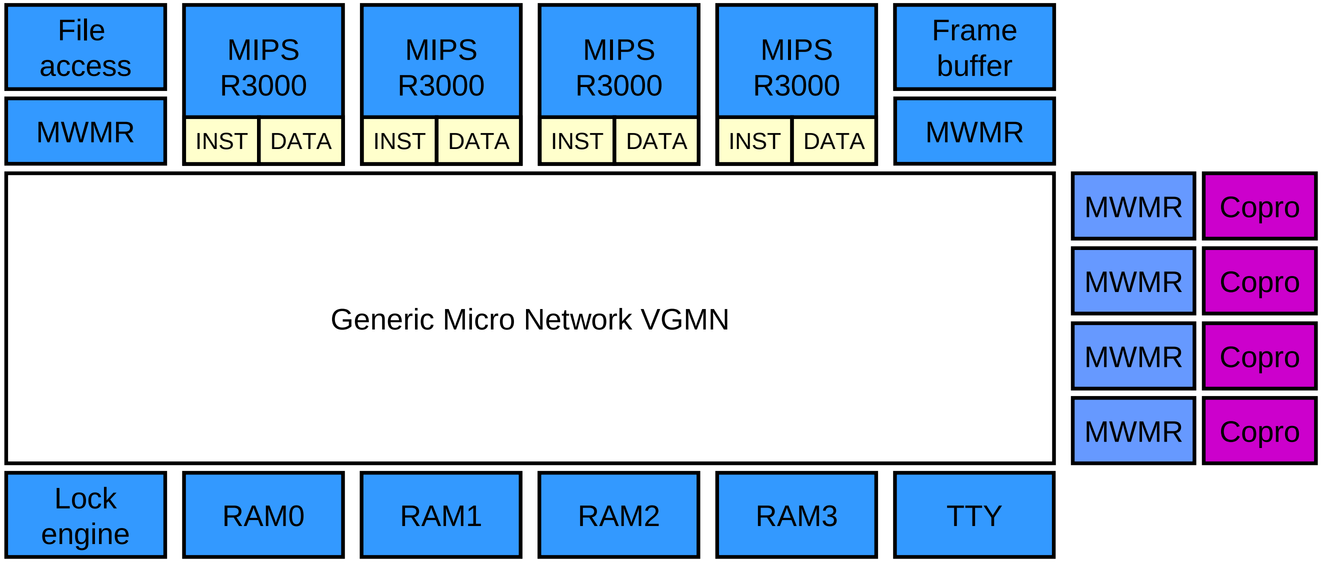




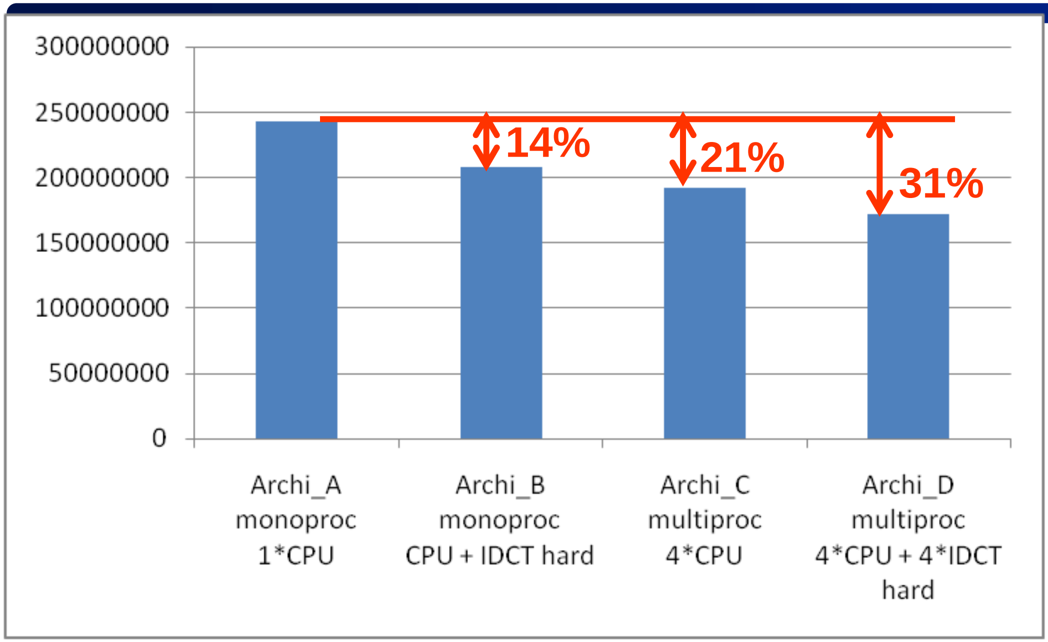
Parallelized software implementation on a multiprocessor architecture

Experiments: architecture #4





MJPEG Results

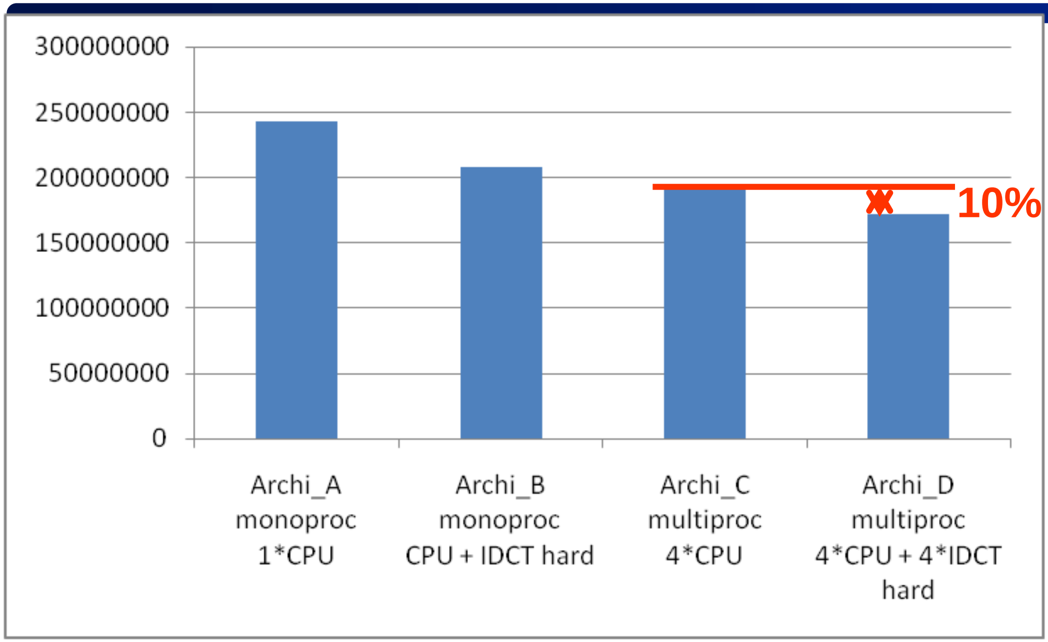


Execution time of the application (in cycles) to process 50 images of 48*48 pixels

IDCT generated by GAUT reduces the application latency by 14%

Parallelization of the application on 4 CPUs reduces the latency by 21%

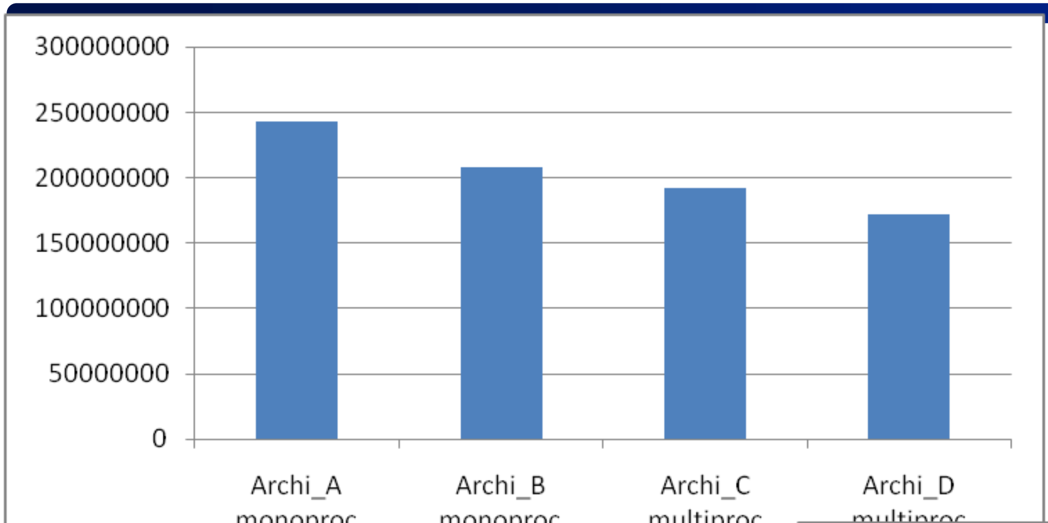
MJPEG Results



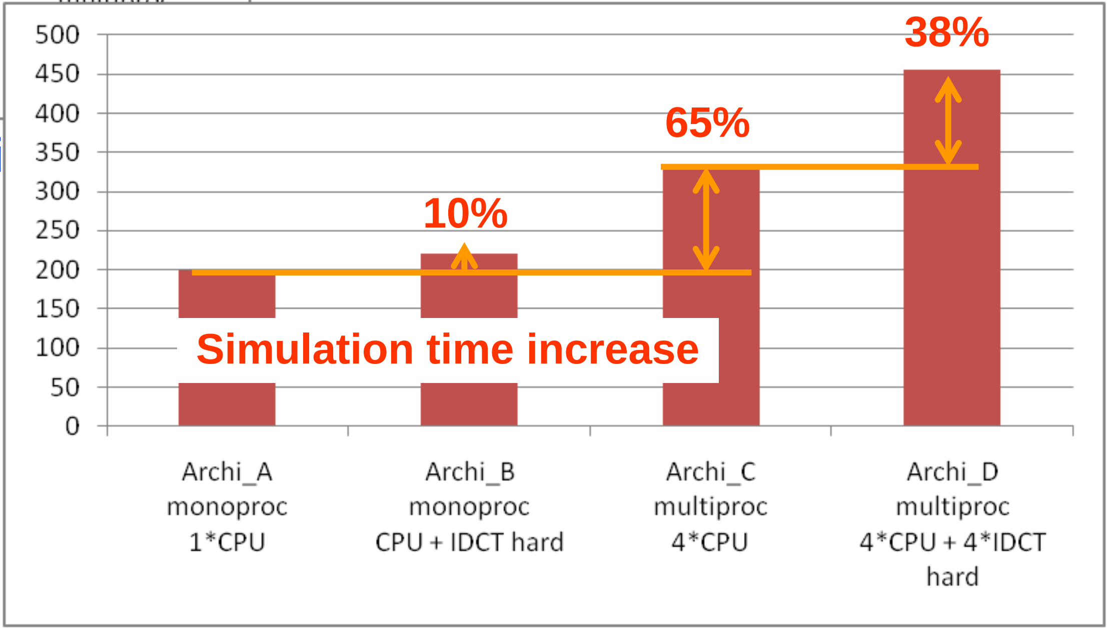
Execution time of the application (in cycles) to process 50 images of 48*48 pixels

The 4 HW IDCT in the multiprocessor architecture further reduce the latency by 10%

MJPEG Results



Execution time of the application (i
to process 50 images of 48*48



Simulation time (in secondes)

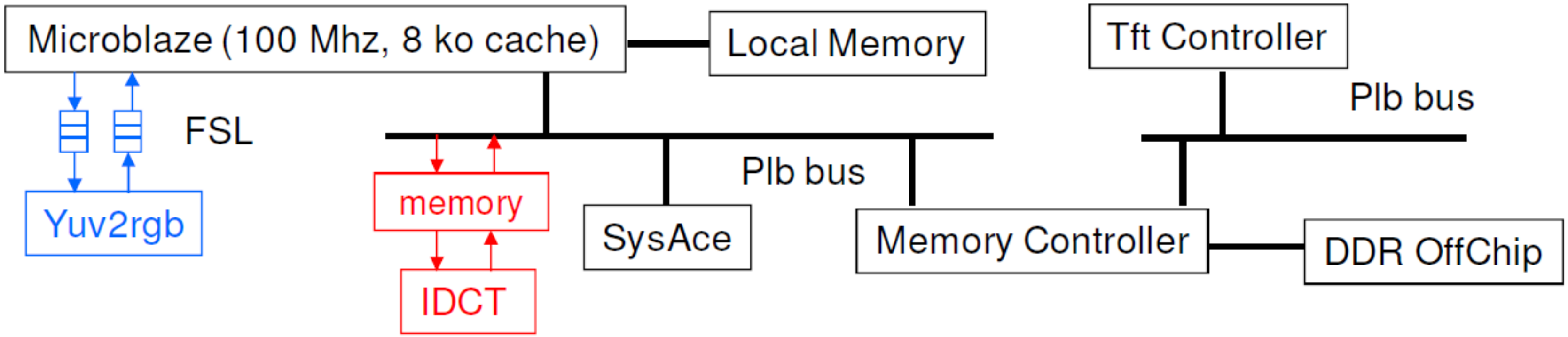
MJPEG: Hardware prototyping

□ Real time decoding: 24 QCIF images/sec

- IDCT: maximum I/O bandwidth (4 parallel input ports) and the lower latency (33 cycles, Freq. 138,9Mhz)
- YUV2RGB: minimum latency (12 cycles, Freq. 249,18Mhz)

□ Compared to a pure SW implementation

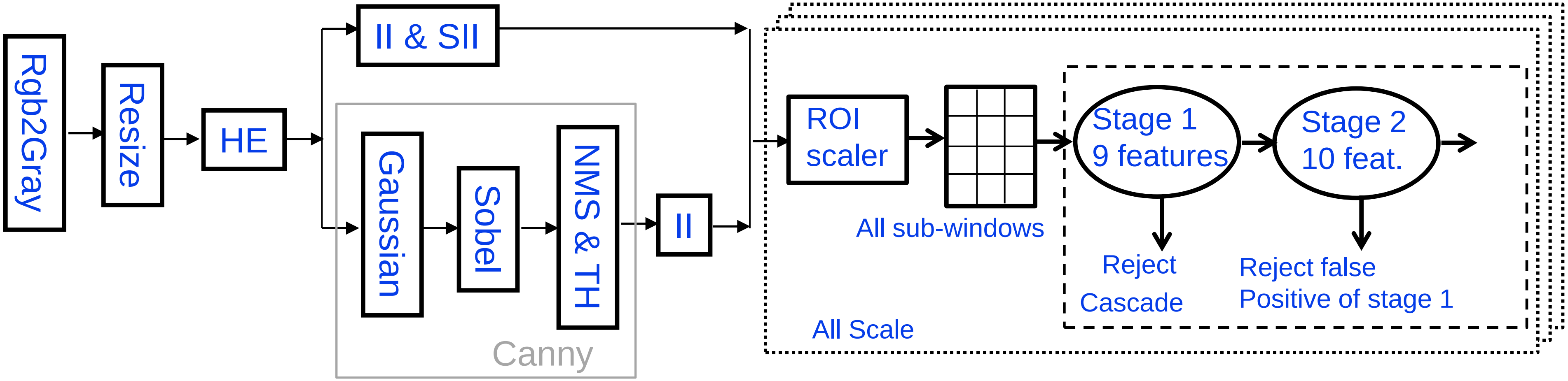
- 10x speed-up for the IDCT function
- 5x speed-up for the yuv2rgb function



SoC design on a FPGA Xilinx Virtex 5 LX110 (XUPV5) board

Viola Jones: Hardware prototyping

- Block Diagram of a Viola Jones Face detector

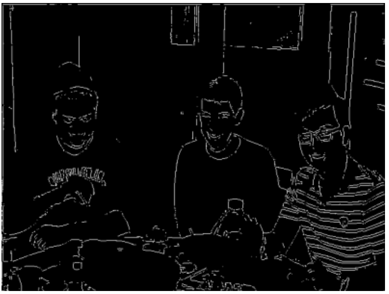


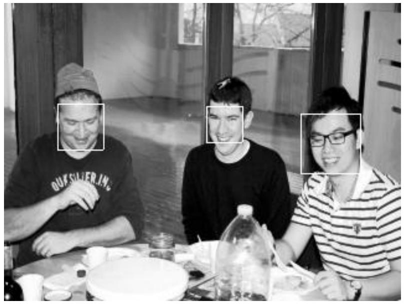
- 7x speed-up compared to a pure sw implementation

Rgb2gray

Contrast Enhancement

Noise Reduction

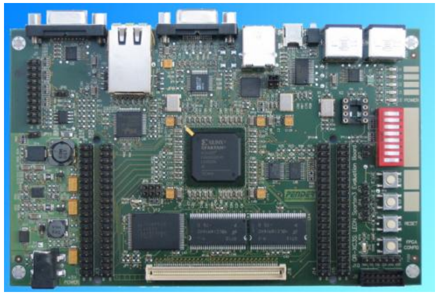

Canny Edge Detector


Face Detection

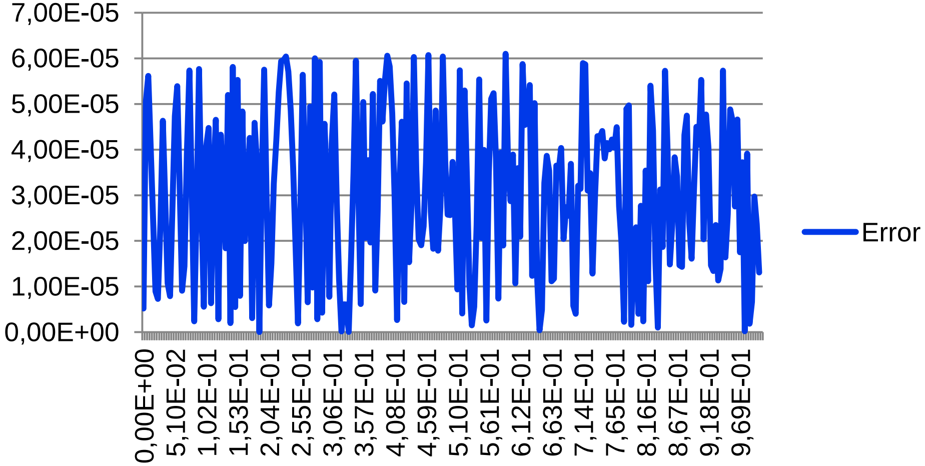
HLS for Hardware prototyping

Slope detection : acos (cordic) hwpu
Texture detection: gaussian filter and square root hwpu
Soc Leon3 interface (AHB, Grlib)

SpeedUp >= 140
Error <= 0.00006

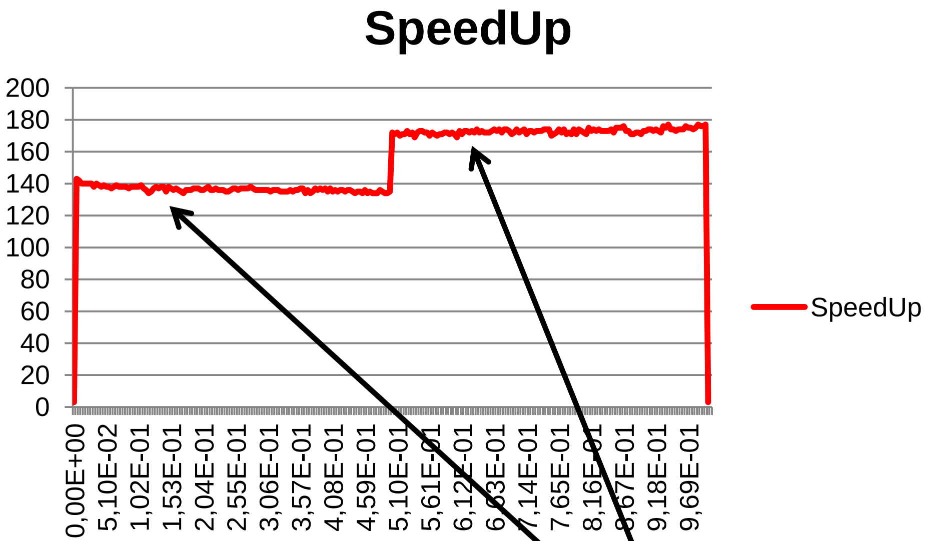


Error



$$2^{-n} \approx 0.00001 \le error \le 2^{-(n-2)} \approx 0.00006$$

n =16, number of rotation



```
/* __ieee754_acos(x)
 * For |x|<=0.5 acos(x) = pi/2 - (x + x*x^2*R(x^2))
 * where
 * R(x^2) is a rational approximation
 *                  of (asin(x)-x)/x^3
 * For x>0.5
 * acos(x) = pi/2 - (pi/2 - 2asin(sqrt((1-x)/2)))
 */
```

Prototyping platform

Sundance platform

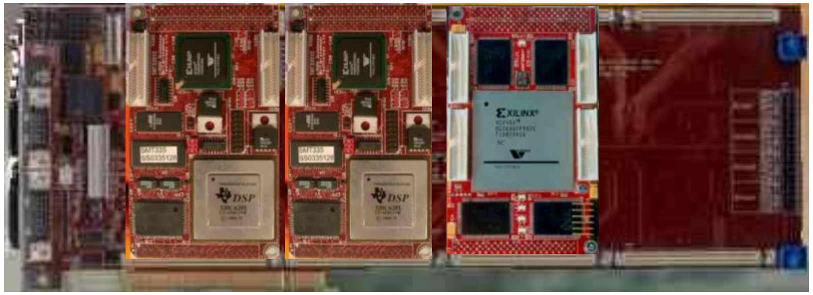
Mother board

Daughter boards

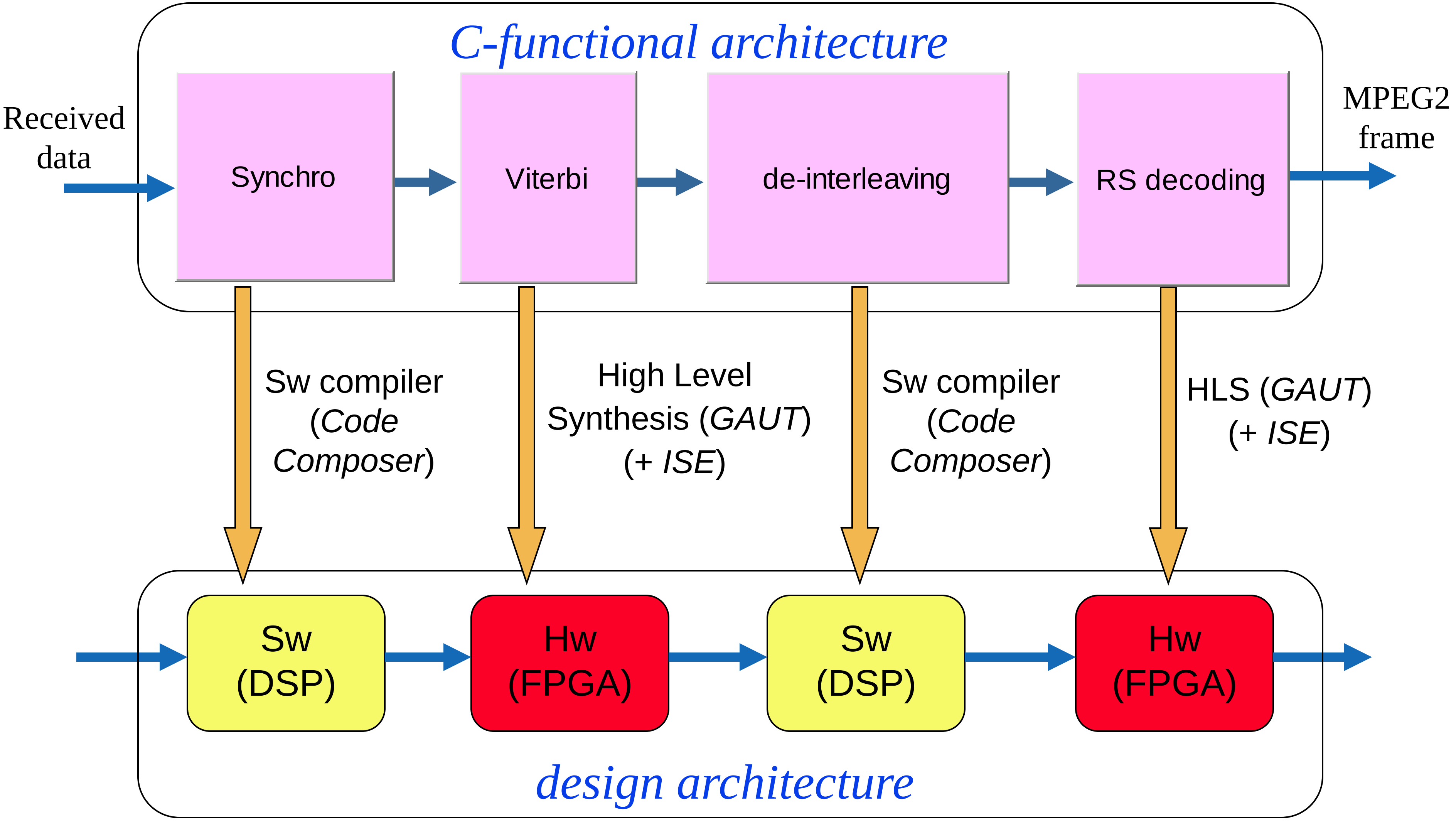
- *DSP C62 C67 (Texas Instrument)*
- *FPGA Virtex 1000E (Xilinx)*

Interconnection matrix

- *Point to point links : Com Port (CP, up to 20 Mbytes/sec) and Sundance Digital Bus (SDB, up to 200 Mbytes/sec)*

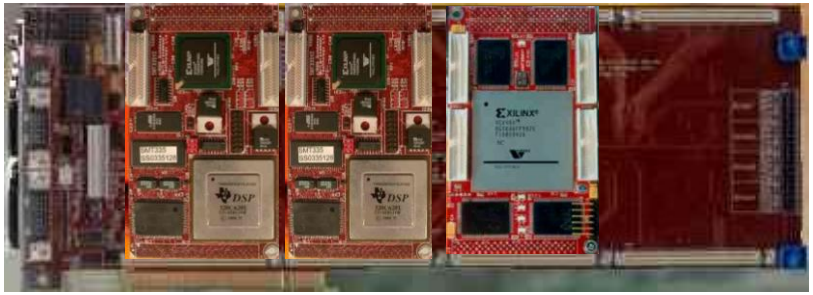


DVB-DSNG receiver architecture mapping



DVB-DSNG receiver

- Synchronization and interleaving : Sw : C62 DSP
- Viterbi and Reed Solomon decoders : Hw : Virtex-1000E FPGA
- 4 SDB links
- 26 Mbps throughput (limited by the synchronization bloc…C64 for higher throughputs)

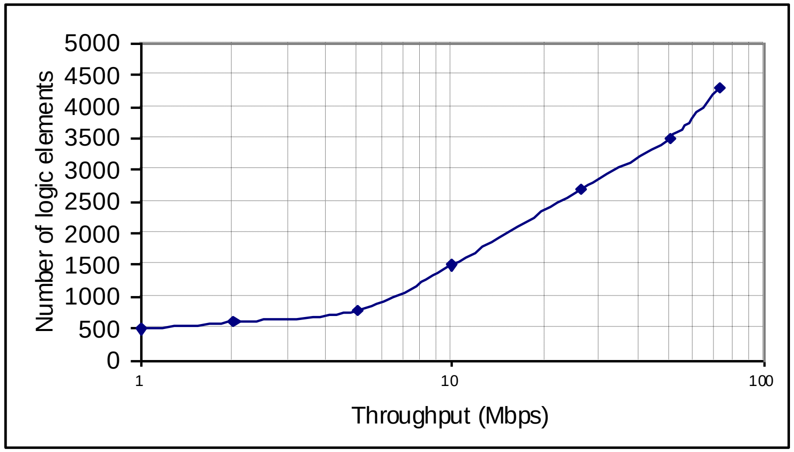


Viterbi decoding

- functional/application parameters : state number, throughput

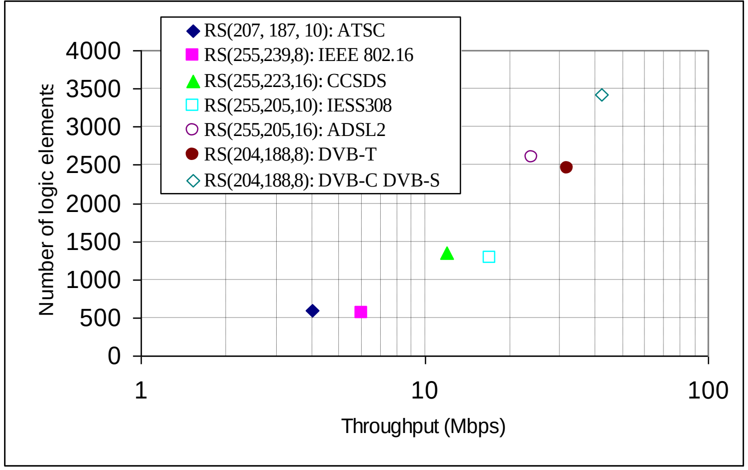
State Number	8	16	32	64	128
Throughput (Mbps)	44	39	35	26	22
Synthesis Time (s)	1	1	3	9	27
Number of logic elements	223	434	1130	2712	7051

- DVB-DSNG standard : throughput : 1.5 to 72 Mbps, 64 states Viterbi decoder

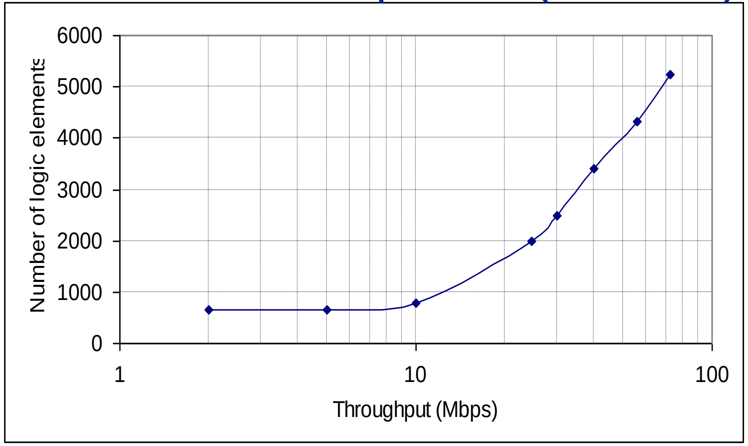


Reed Solomon decoding

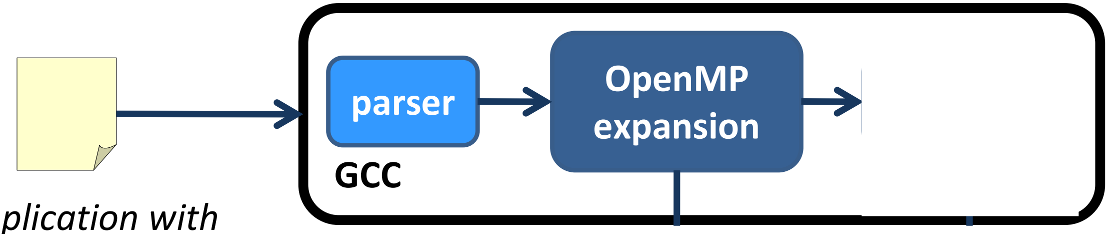
- functional/application parameters : number of input symbols, data symbols, throughput



- DVB-DSNG standard : 1.5 to 72 Mbps, RS (204/188) decoder

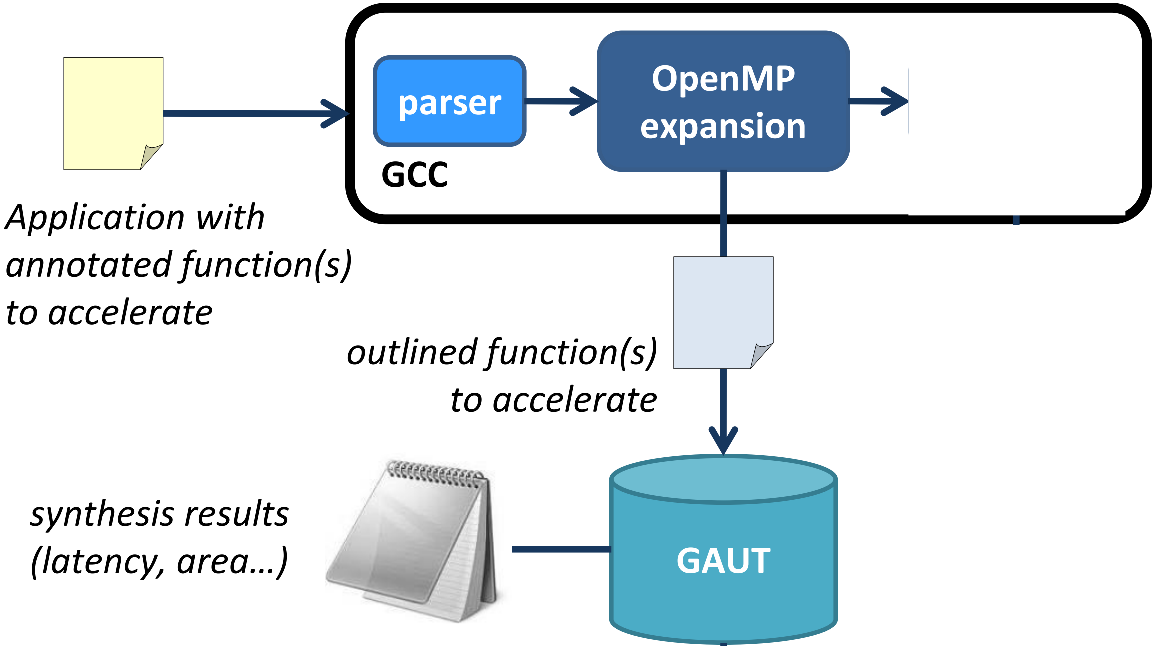


OpenMP/HLS & manycores/HWPU

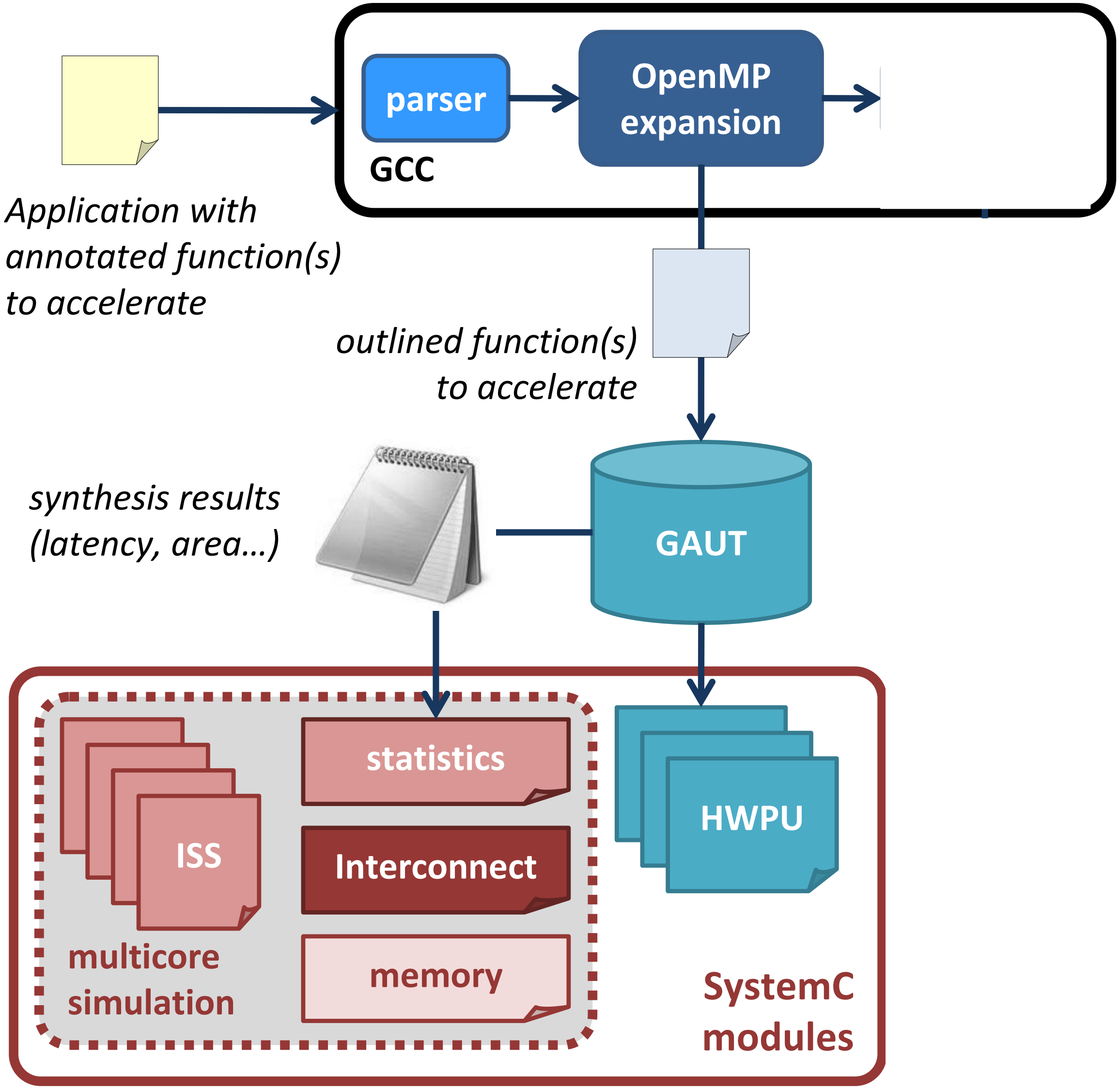


Application with annotated function(s) to accelerate

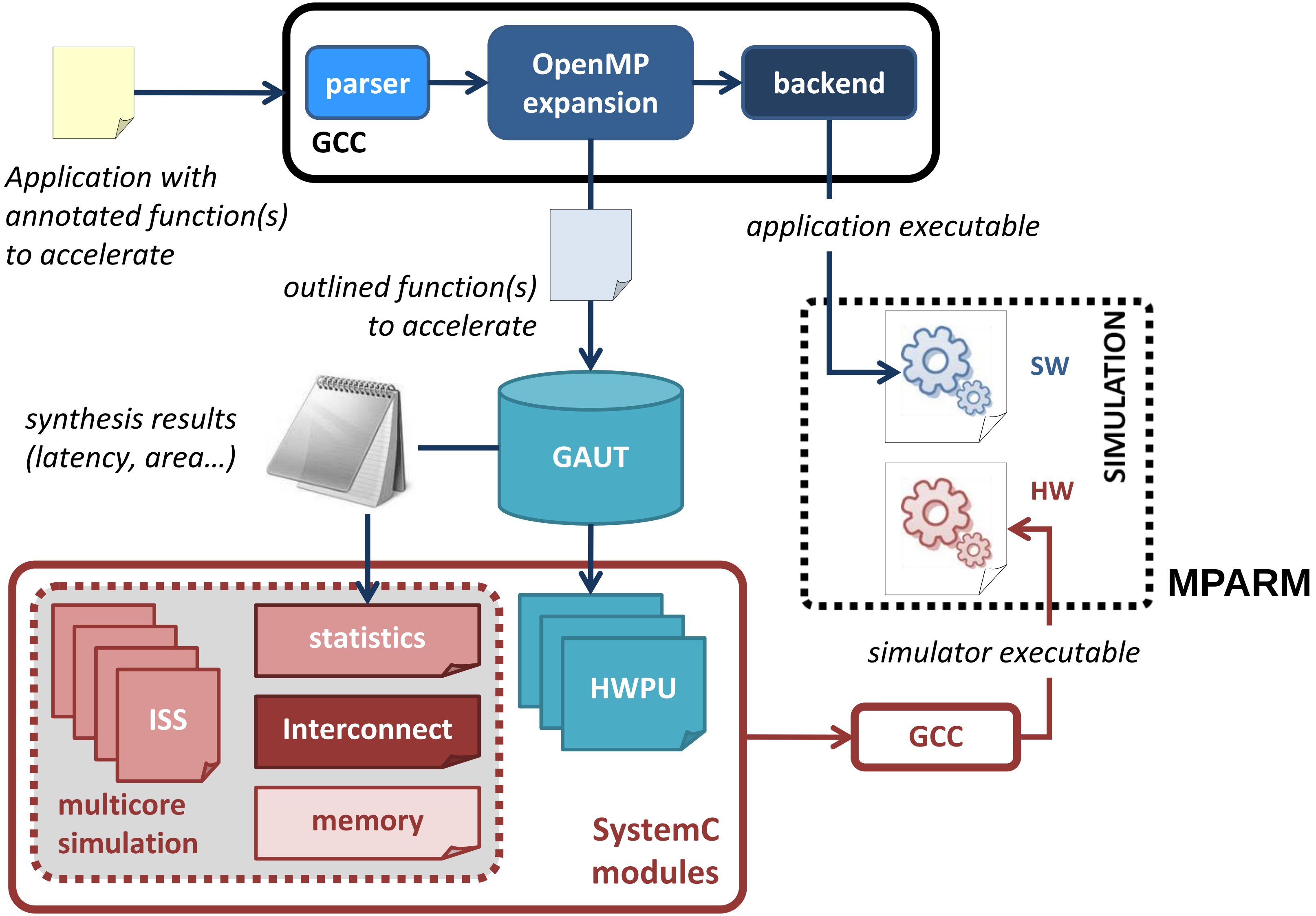
OpenMP/HLS & manycores/HWPU


parser
OpenMP expansion
GCC
Application with annotated function(s) to accelerate
outlined function(s) to accelerate
synthesis results (latency, area…)
GAUT

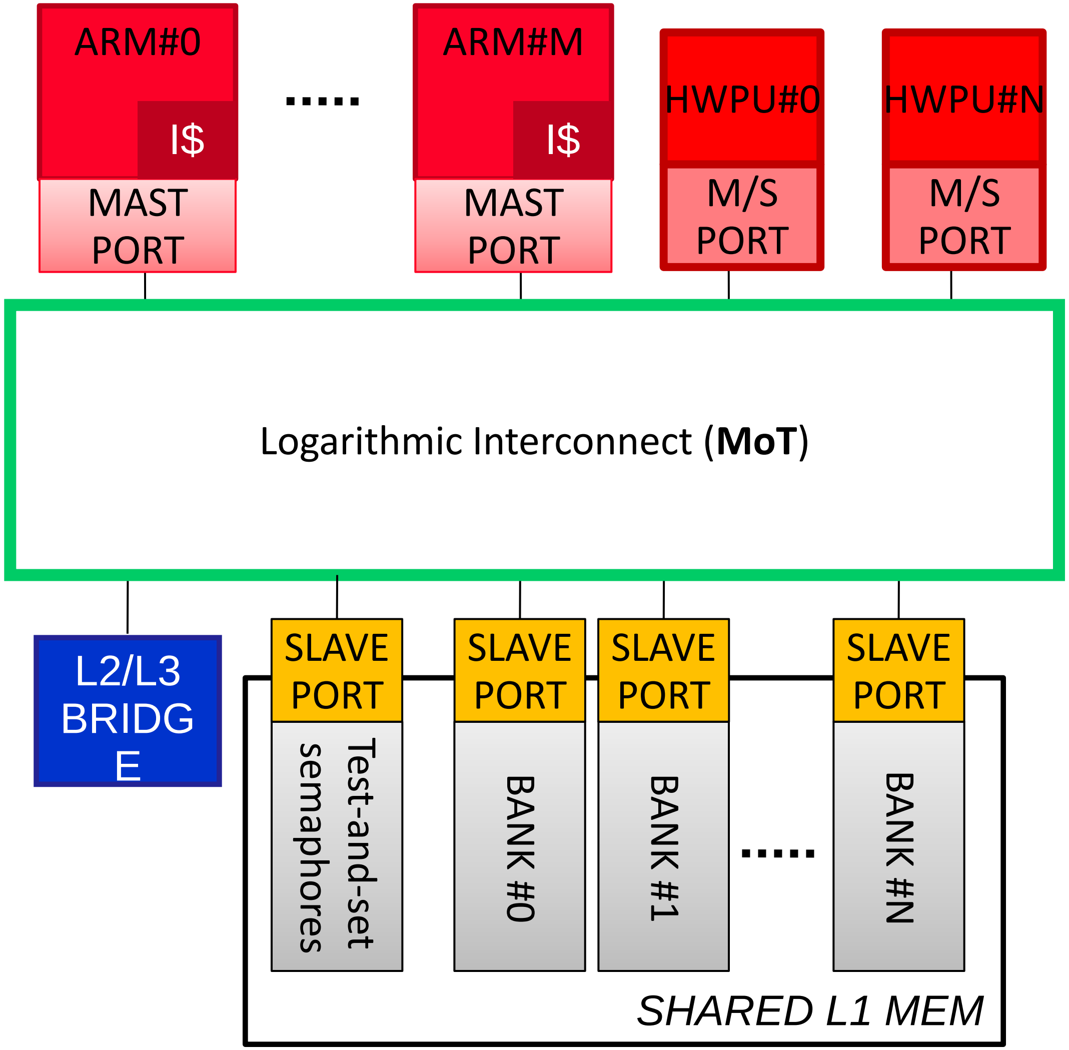
OpenMP/HLS & manycores/HWPU

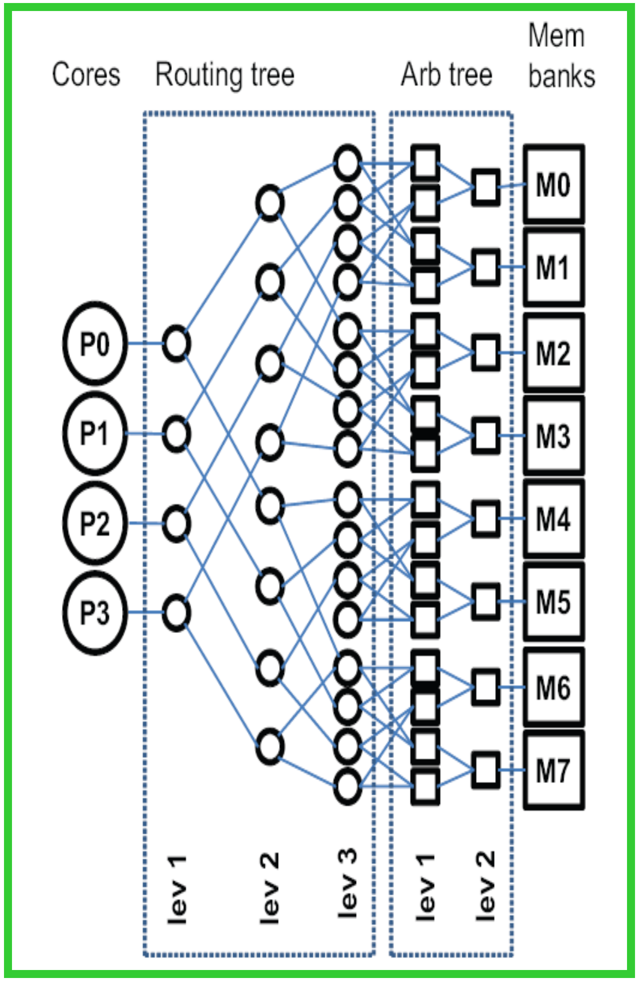

parser
OpenMP expansion
GCC
Application with annotated function(s) to accelerate
outlined function(s) to accelerate
synthesis results (latency, area...)
GAUT
statistics
ISS
Interconnect
memory
multicore simulation
HWPU
SystemC modules

OpenMP/HLS & manycores/HWPU



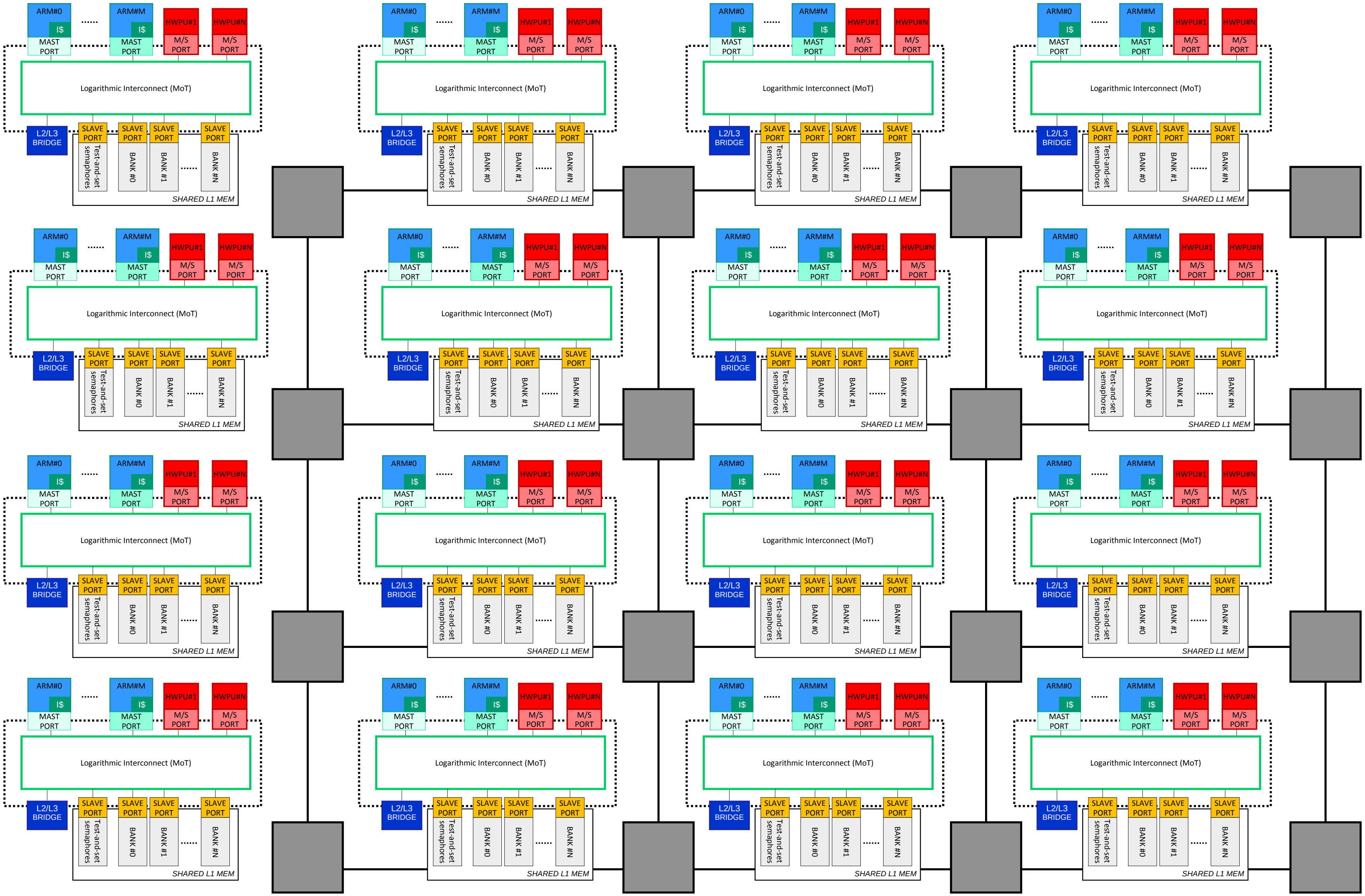
MPARM Architecture





Mesh Of Tree

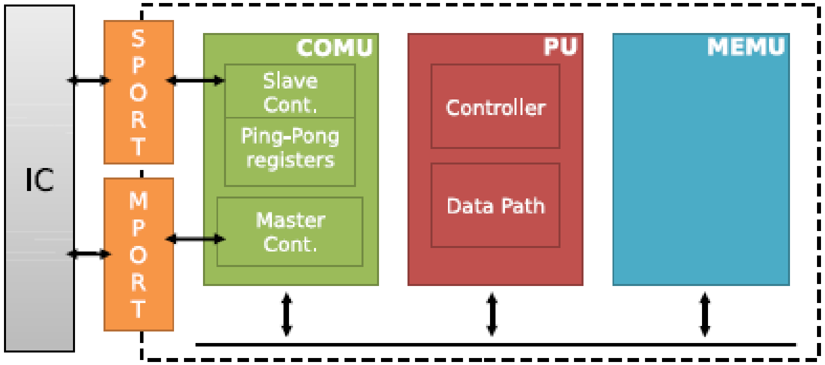
Target architecture



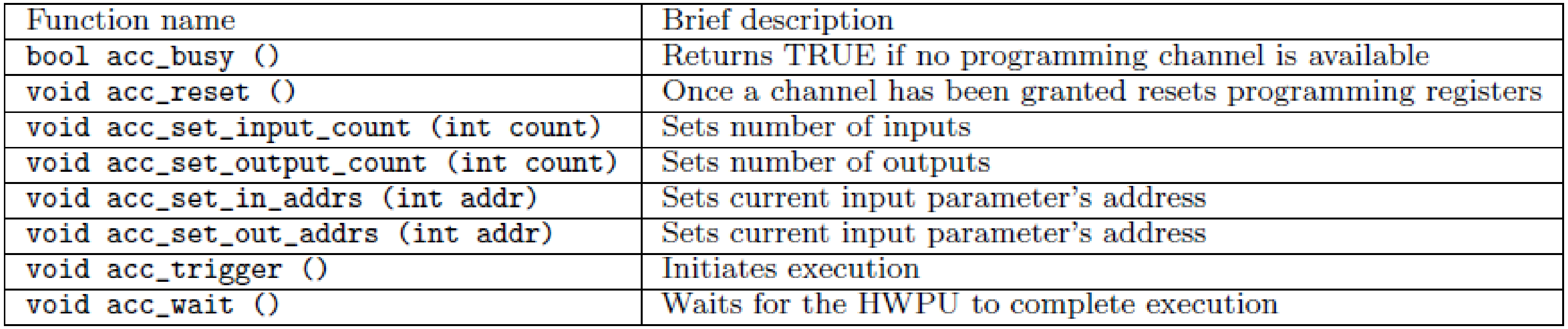
HWPU Integration

□ Interface de communication

- Maître / Esclave
- Registres de configuration
  - ❑ ***Nombre d’entrées***
  - ❑ ***Nombre de sorties***
  - ❑ ***Emplacements des entrées***
  - ❑ ***Emplacements des sorties***
  - ❑ ***Mode***
  - ❑ ***Etat du HWPU***
  - ❑ ***Démarrage***
- Les registres de configuration peuvent être doublés
  - ❑ ***Recouvrement de la configuration et du calcul***

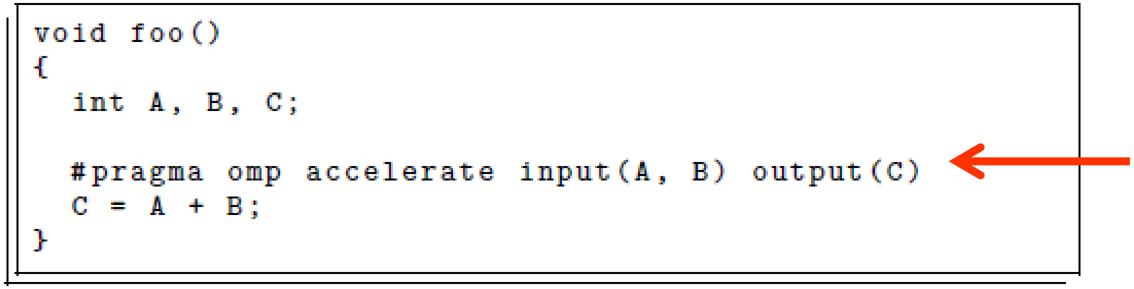


□ Interface de programmation



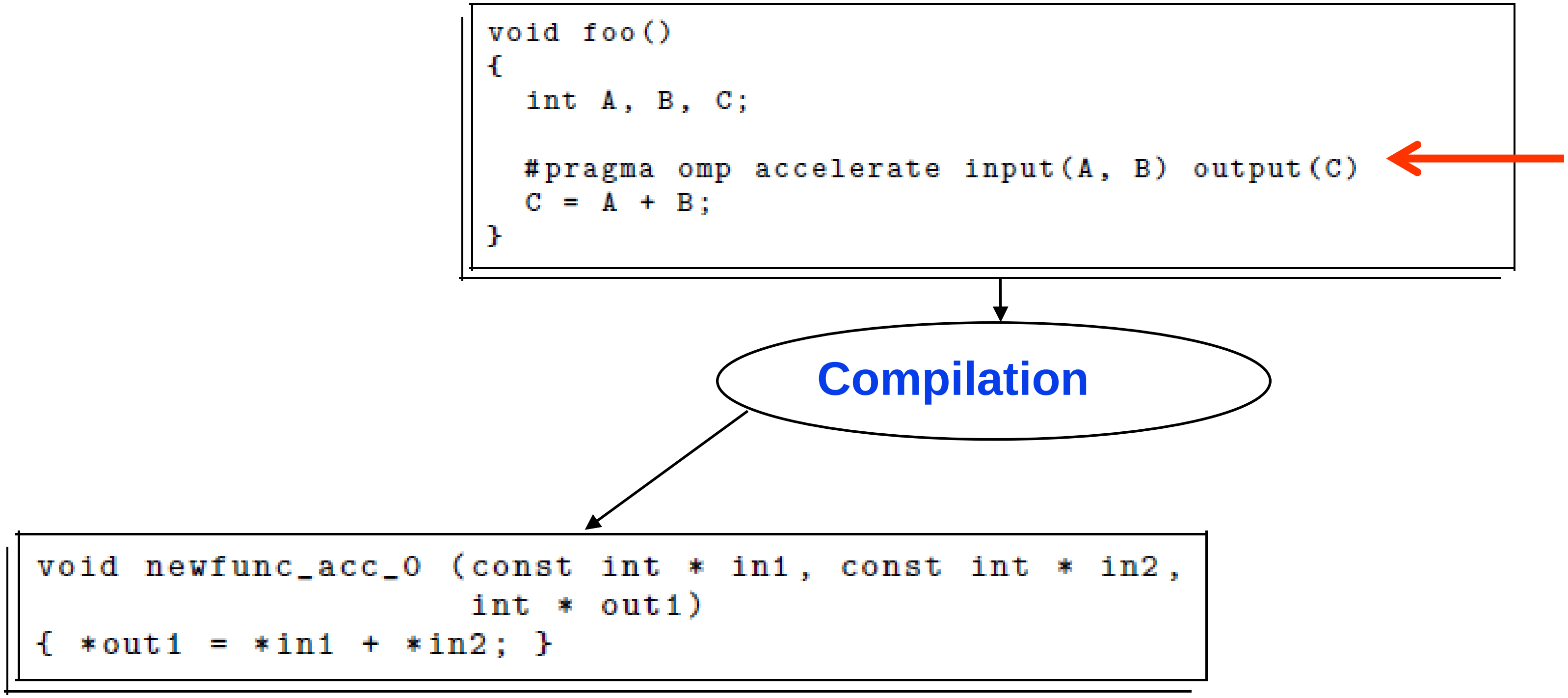
Function name	Brief description
`bool acc_busy ()`	Returns TRUE if no programming channel is available
`void acc_reset ()`	Once a channel has been granted resets programming registers
`void acc_set_input_count (int count)`	Sets number of inputs
`void acc_set_output_count (int count)`	Sets number of outputs
`void acc_set_in_addrs (int addr)`	Sets current input parameter’s address
`void acc_set_out_addrs (int addr)`	Sets current input parameter’s address
`void acc_trigger ()`	Initiates execution
`void acc_wait ()`	Waits for the HWPU to complete execution

Example



```
void foo()
{
   int A, B, C;

   #pragma omp accelerate input(A, B) output(C)
   C = A + B;
}
```

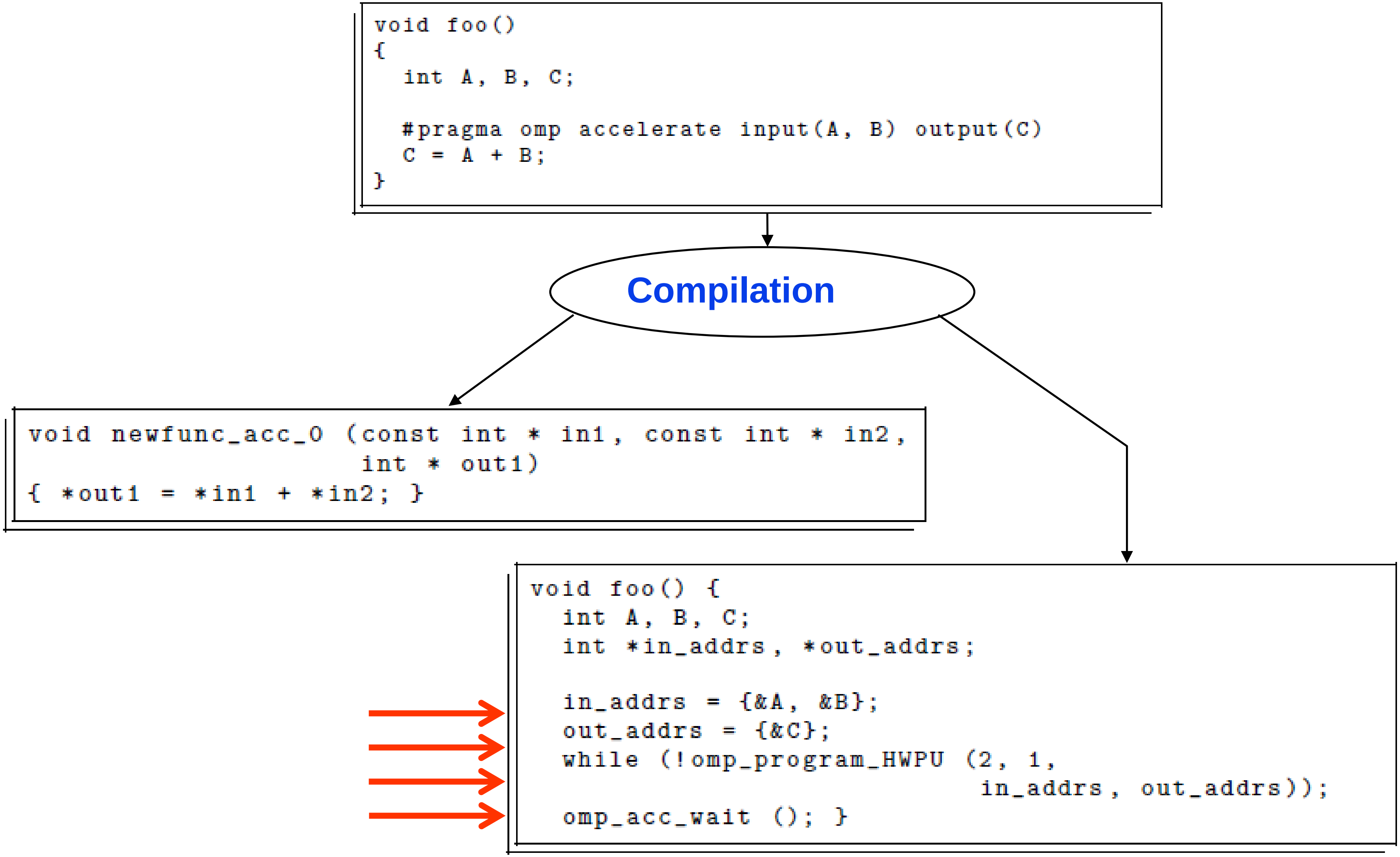
Example



```
void foo()
{
   int A, B, C;

   #pragma omp accelerate input(A, B) output(C)
   C = A + B;
}
```

```
void newfunc_acc_0 (const int * in1, const int * in2,
                    int * out1)
{ *out1 = *in1 + *in2; }
```

Example



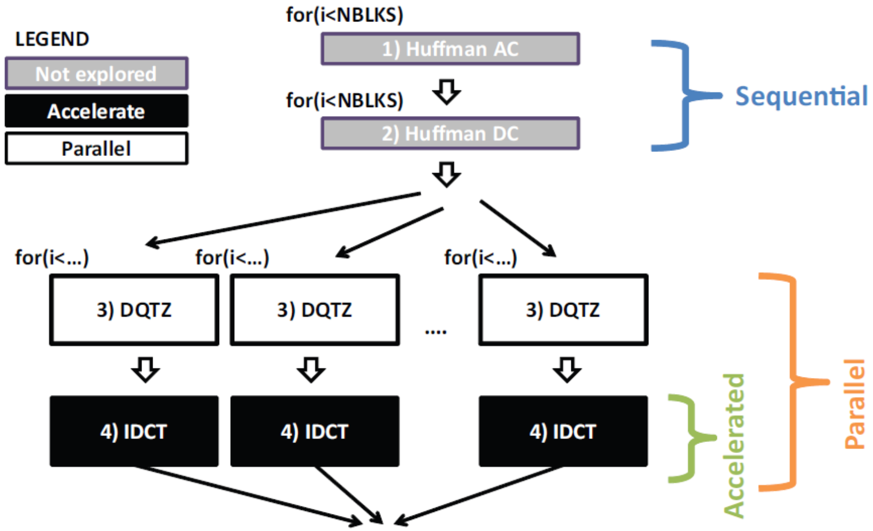
```
void foo()
{
   int A, B, C;

   #pragma omp accelerate input(A, B) output(C)
   C = A + B;
}
```

Compilation

```
void newfunc_acc_0 (const int * in1, const int * in2,
                    int * out1)
{ *out1 = *in1 + *in2; }
```

```
void foo() {
   int A, B, C;
   int *in_addrs, *out_addrs;

   in_addrs = {&A, &B};
   out_addrs = {&C};
   while (!omp_program_HWPU (2, 1,
                             in_addrs, out_addrs));
   omp_acc_wait (); }
```

Results



LEGEND

Not explored

Accelerate

Parallel

for(i<NBLKS)

1) Huffman AC

for(i<NBLKS)

2) Huffman DC

Sequential

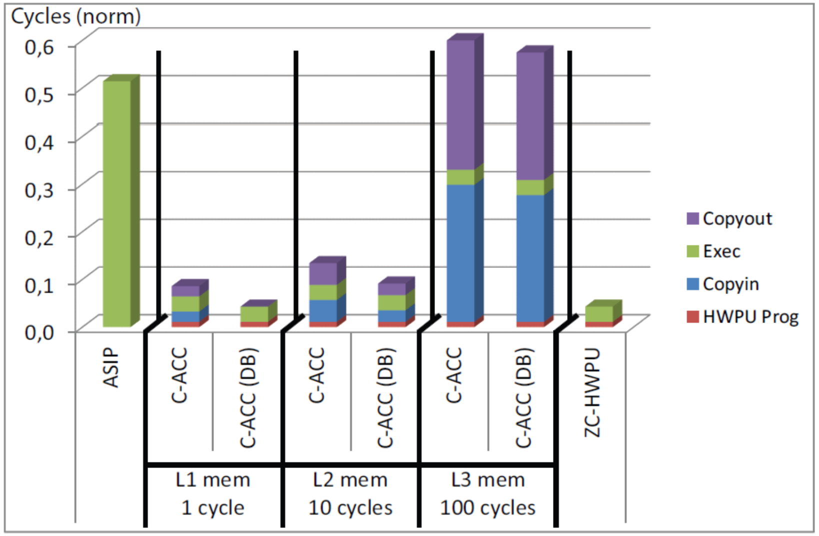
for(i<...) for(i<...) for(i<...)

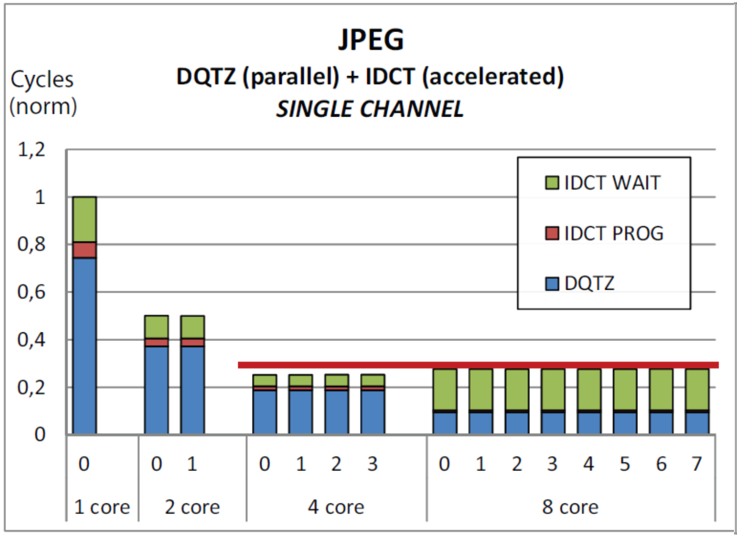
3) DQTZ 3) DQTZ .... 3) DQTZ

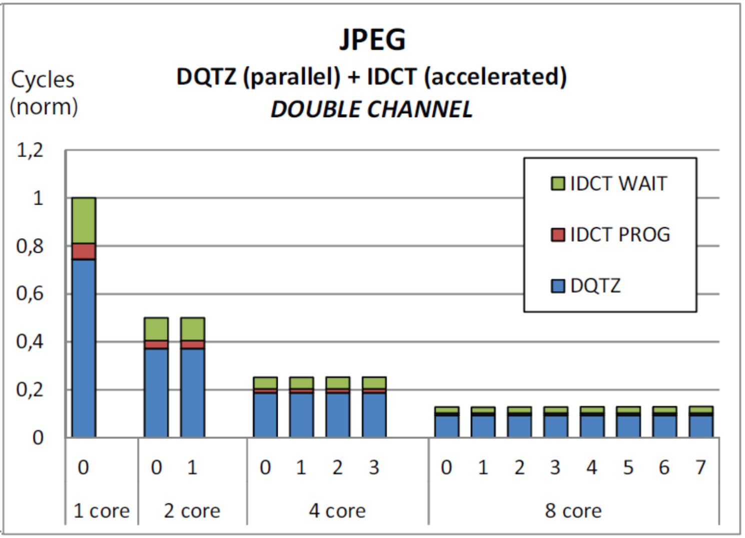
4) IDCT 4) IDCT 4) IDCT

Accelerated

Parallel







GAUT 4 (not yet available, but soon…)

- □ **An open source HLS tool**
  - ■ For both data and control-dominated algorithms (CDFG)
- □ **Input :**
  - ■ C/C++ bit-accurate integer sand fixed-points from Mentor Graphics
  - ■ SystemC : C and C++ lack the constructs and semantics to represent design hierarchy, timing, synchronization/concurrency
  - ■ Floating point
- □ **Output : RTL Architecture**
  - ■ VHDL , Verilog
  - ■ SystemC (CABA + TLM)
  - ■ Resource and timing estimation
- □ **Automated Test-bench generation**
- □ **Automated operators characterization**
- □ **Automated interface generation**
  - ■ AXI, AHB, FSL, …

GAUT 4 (not yet available, but soon…)

□ Constraints

- Clock, I/O protocols, loop transformations (unrolling, merging, loop pipelining with Initiation Interval), memory mapping, function inlining, resource constraints

□ Objectives

- Minimization: area i.e. resources, latency, power consumption…
- Maximization: throughput

□ Keys features

- ❑ ***Used robust and state of the art compilation technology to extract instruction-level (Vectorization) and loop level parallelism (Polyhedral model: graphite for GCC, Polly for LLVM)***
- ❑ ***Many scheduling strategies : modulo scheduling (SMS,IMS) , Force Directed List Scheduling (FDLS), System of difference constraint (SDC)…***
- ❑ ***Memory analysis and optimizations: automatic partitioning of array elements to reduce conflicts and increase throughput***
- ❑ ***Pattern mining for efficient resource sharing***
- ❑ ***Hierarchy synthesis and function level parallelism/pipelining***
- ❑ ***Design Space Exploration with directives (Loop transformation, memory partitionning) and constraints (script): one body of code, many hardware outcomes***

Conclusion

- HLS allows to automatically generate several RTL architectures
  - From an algorithmic/behavioral description and a set of constraints

- HLS allows to generate
  - VHDL models for synthesis purpose
  - SystemC simulation models for virtual prototyping

- HLS allows to explore the design space of
  - Hardware accelerators
  - MPSoC architectures including HW accelerators

- GAUT is free downloadable at
  - http://lab-sticc.fr/www-gaut

References

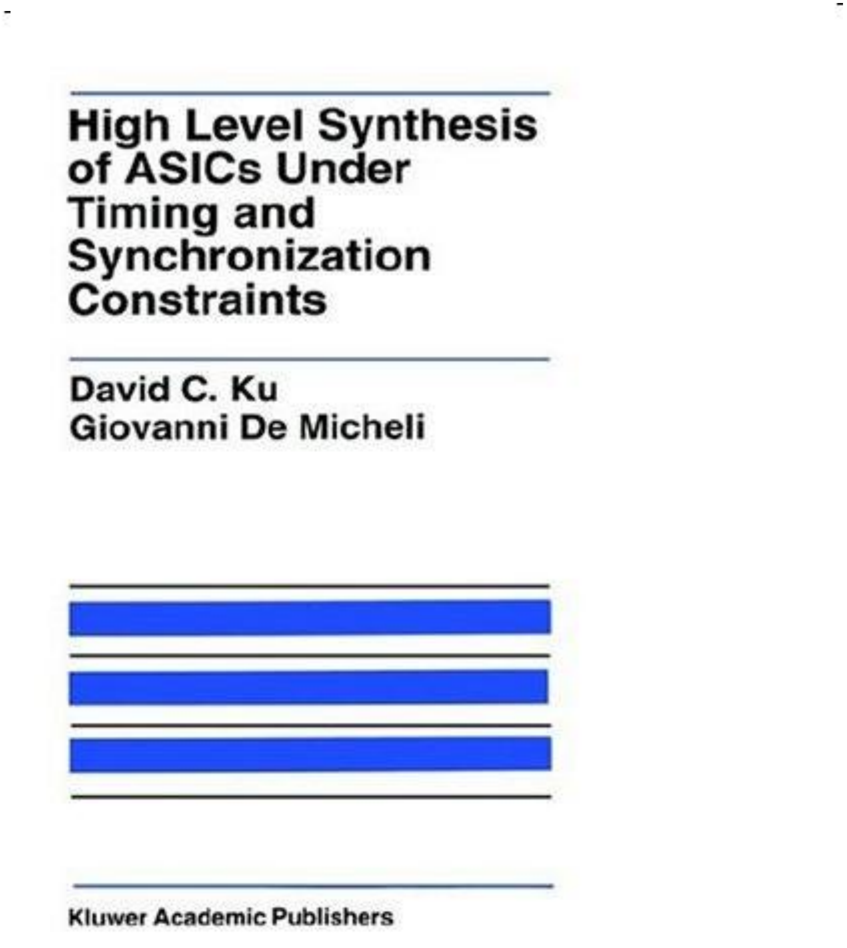
HIGH–LEVEL SYNTHESIS

Introduction to Chip and System Design

Edited by

Daniel D. Gajski

KLUWER ACADEMIC PUBLISHERS

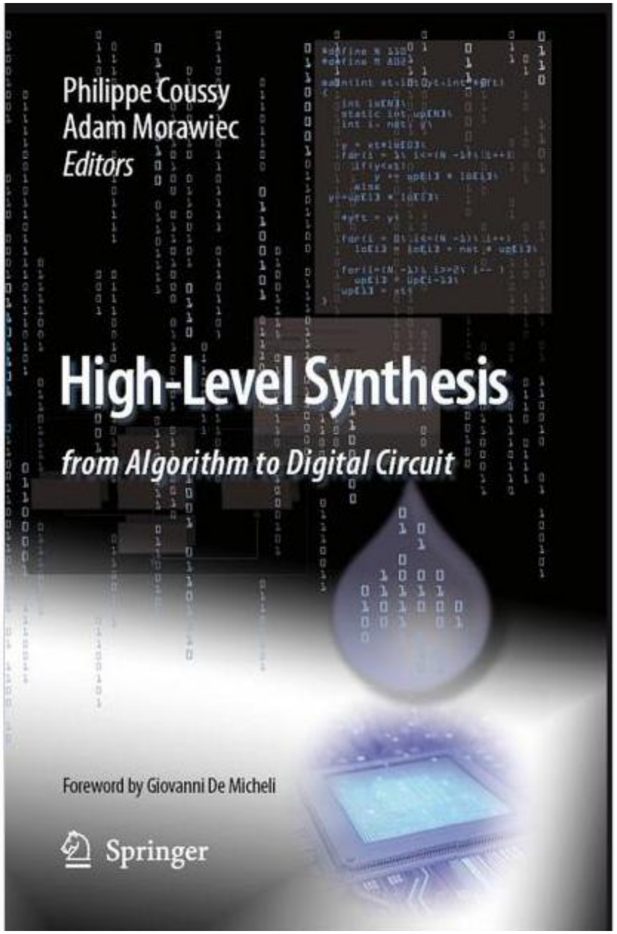


High Level Synthesis of ASICs Under Timing and Synchronization Constraints

David C. Ku
Giovanni De Micheli

Kluwer Academic Publishers

References


Philippe Coussy
Adam Morawiec
Editors
High-Level Synthesis
from Algorithm to Digital Circuit
Foreword by Giovanni De Micheli
Springer

IEEE
JULY/AUGUST 2009
Design&Test
of Computers
http://www.computer.org/dt
SPECIAL ISSUE ON HIGH-LEVEL SYNTHESIS
· Past, Present, and Future · Functional Equivalence Verification Tools
· High Level Dataflow Transformations Using Taylor Expansion Diagrams
· Subword Switching Activity Minimization to Optimize Dynamic Power Consumption
· Virtual Roundtable: User Perspectives
IEEE
Celebrating 125 Years of Engineering the Future
IEEE computer society
IEEE CIRCUITS AND SYSTEMS SOCIETY

Academic tools

- Streamroller (Univ. Mich.)
- SPARK (UCSD)
- xPilot (UCLA)
- UGH (TIMA+LIP6)
- MMALPHA (IRISA+CITI+…)
- ROCCC (UC Riverside)
- GAUT (UBS / Lab-STICC)
- …

Commercial tools

- ❑CatapultC (Mentor Graphics => Calypto)
- ❑PICO (Spin-off HP => Synfora => Synopsys)
- ❑Cynthecizer (Forte design)
- ❑Cyber (NEC)
- ❑AutoPilot (AutoESL => Xilinx)
- ❑C to Silicon (Candence)
- ❑Synphony (Synopsys)
- ❑…

Une introduction à la synthèse de haut-niveau

(ou comment générer des architectures matérielles à partir du langage C)

Université de Bretagne-Sud
Lab-STICC

Philippe COUSSY
philippe.coussy@univ-ubs.fr