Innovative Technology Series (ITS) Training Workshop 4

High-Level Synthesis

Benjamin CARRION SCHAFER, Ph.D.
b.carrionschafer@polyu.edu.hk

Let’s start by thanking the Sponsors

IEEE
Hong Kong Section 香港分會

The Institution of Engineering and Technology

香港電子業商會
The Hong Kong Electronic Industries Association

HKIE
THE HONG KONG INSTITUTION OF ENGINEERS
香 港 工 程 師 學 會
Information Technology Division
資 訊 科 技 分 部

Today's Program

9:30 Welcome and Introduction (of myself)

9:35 – 10:30 High-Level Synthesis

- What is High Level Synthesis ?
- Advantages of C-based VLSI design vs. RTL
- Target architecture
- How to write synthesizable C/C++ code

10:30-10:45 Coffee Break

10:45-12:30 Hands on Tutorial

- CyberWorkBench from NEC

Resume

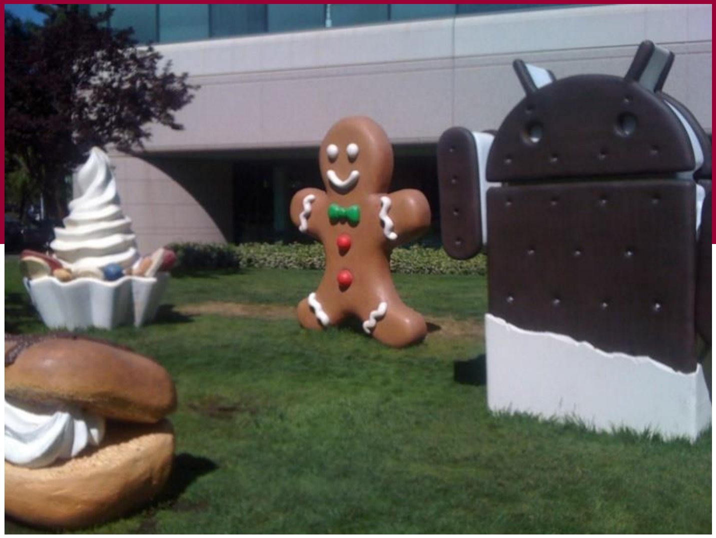
- September 2012-present
  - Assistant Professor, PolyU
- 2007-2012
  - Researcher, NEC Central Research Laboratories, VLSI Design Methodology Group, Kawasaki, Japan
- 2005-2007
  - Visiting Research Scholar, Seoul National University
- 2002-2004
  - Postdoctoral Researcher UCLA, CS Department, USA
- 1999-2002
  - PhD University of Birmingham, UK

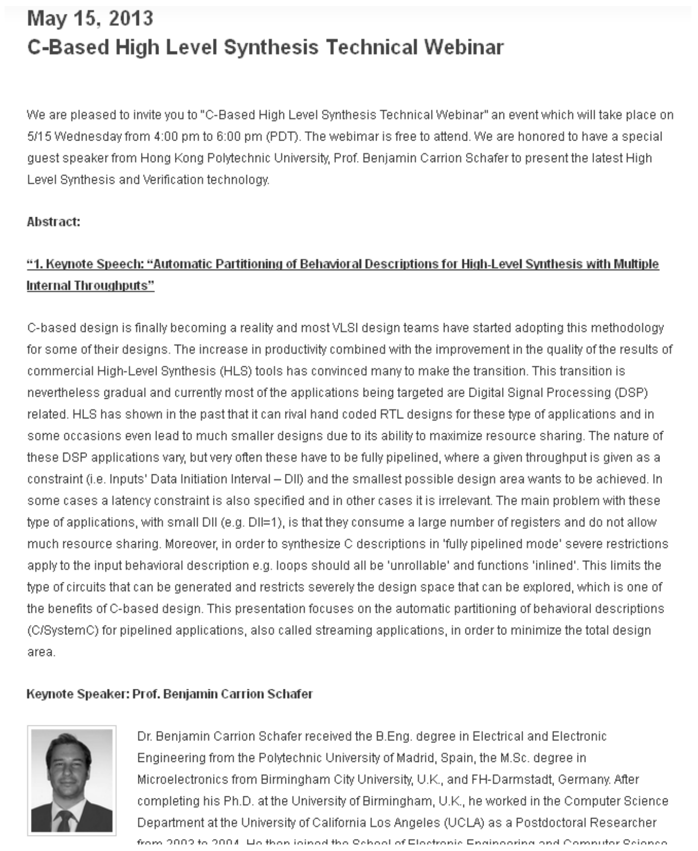
C-Based VLSI Design

- Researched and Developed different parts of NEC's commercial HLS tool "CyberWorkBench" (www.cyberworkbench.com). In particular:
  1. Automatic design space explorer
  2. Fixed point data type synthesis
  3. Power estimator
  4. Integration of HW model in virtual platform
- Hold numerous training courses
- Supported international customer evaluations
- International sales and marketing

Industrial Experience

- On-site customer visits and on-site training courses
- HLS tool Evaluation
- Webinars





May 15, 2013

C-Based High Level Synthesis Technical Webinar

We are pleased to invite you to "C-Based High Level Synthesis Technical Webinar" an event which will take place on 5/15 Wednesday from 4:00 pm to 6:00 pm (PDT). The webimar is free to attend. We are honored to have a special guest speaker from Hong Kong Polytechnic University, Prof. Benjamin Carrion Schafer to present the latest High Level Synthesis and Verification technology.

Abstract:

"1. Keynote Speech: "Automatic Partitioning of Behavioral Descriptions for High-Level Synthesis with Multiple Internal Throughputs"

C-based design is finally becoming a reality and most VLSI design teams have started adopting this methodology for some of their designs. The increase in productivity combined with the improvement in the quality of the results of commercial High-Level Synthesis (HLS) tools has convinced many to make the transition. This transition is nevertheless gradual and currently most of the applications being targeted are Digital Signal Processing (DSP) related. HLS has shown in the past that it can rival hand coded RTL designs for these type of applications and in some occasions even lead to much smaller designs due to its ability to maximize resource sharing. The nature of these DSP applications vary, but very often these have to be fully pipelined, where a given throughput is given as a constraint (i.e. Inputs' Data Initiation Interval – DII) and the smallest possible design area wants to be achieved. In some cases a latency constraint is also specified and in other cases it is irrelevant. The main problem with these type of applications, with small DII (e.g. DII=1), is that they consume a large number of registers and do not allow much resource sharing. Moreover, in order to synthesize C descriptions in 'fully pipelined mode' severe restrictions apply to the input behavioral description e.g. loops should all be 'unrollable' and functions 'inlined'. This limits the type of circuits that can be generated and restricts severely the design space that can be explored, which is one of the benefits of C-based design. This presentation focuses on the automatic partitioning of behavioral descriptions (C/SystemC) for pipelined applications, also called streaming applications, in order to minimize the total design area.

Keynote Speaker: Prof. Benjamin Carrion Schafer

Dr. Benjamin Carrion Schafer received the B.Eng. degree in Electrical and Electronic Engineering from the Polytechnic University of Madrid, Spain, the M.Sc. degree in Microelectronics from Birmingham City University, U.K., and FH-Darmstadt, Germany. After completing his Ph.D. at the University of Birmingham, U.K., he worked in the Computer Science Department at the University of California Los Angeles (UCLA) as a Postdoctoral Researcher from 2003 to 2004. He then joined the School of Electronic Engineering and Computer Science

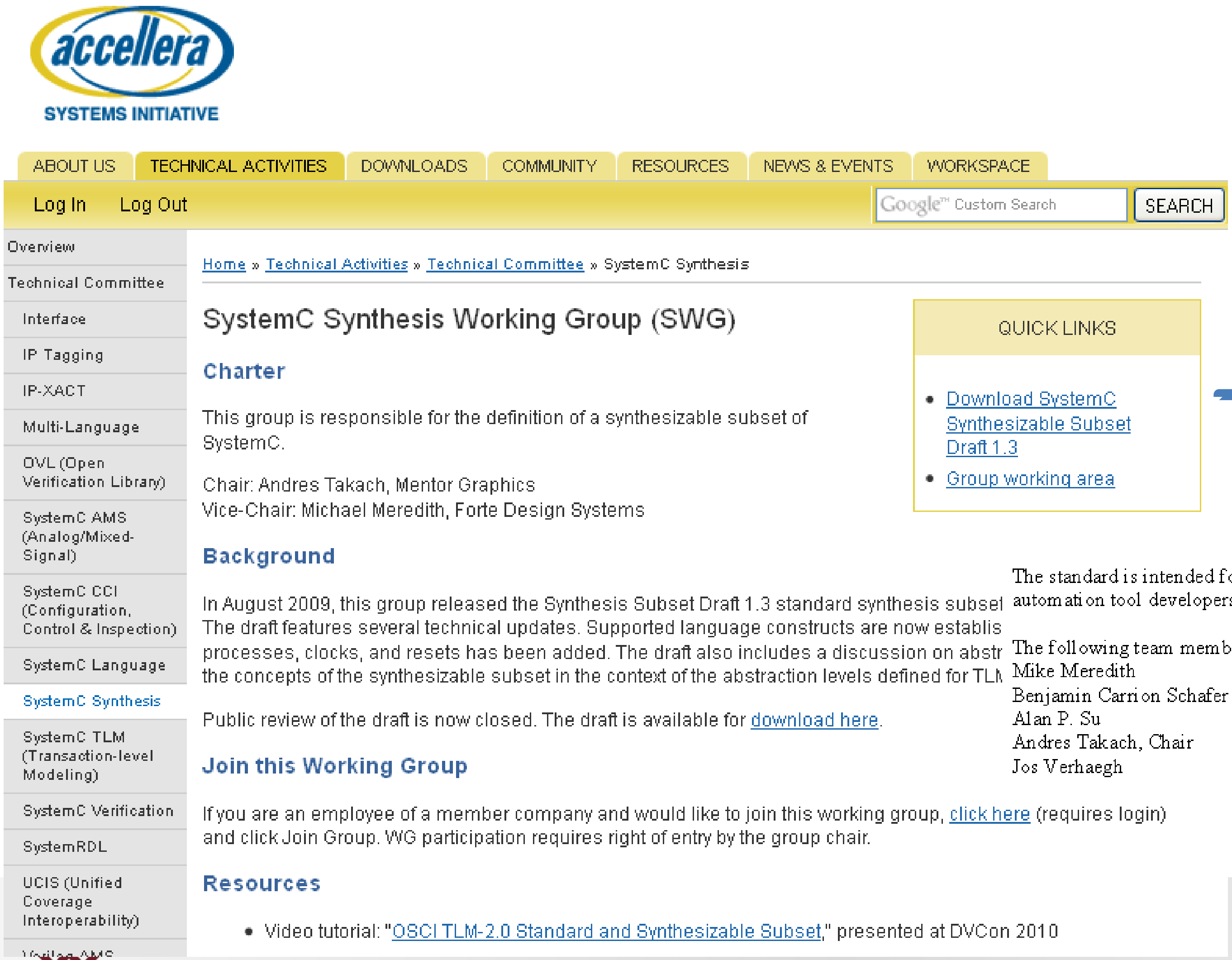
Industrial Experience

- Participated in Exhibitions
  - DATE
  - DAC
  - ASP-DAC

DAC 2008

DAC 2010

SystemC Synthesis Working Group



Home » Technical Activities » Technical Committee » SystemC Synthesis

SystemC Synthesis Working Group (SWG)

Charter

This group is responsible for the definition of a synthesizable subset of SystemC.

Chair: Andres Takach, Mentor Graphics
Vice-Chair: Michael Meredith, Forte Design Systems

QUICK LINKS

- Download SystemC Synthesizable Subset Draft 1.3
- Group working area

Background

In August 2009, this group released the Synthesis Subset Draft 1.3 standard synthesis subset The draft features several technical updates. Supported language constructs are now establis processes, clocks, and resets has been added. The draft also includes a discussion on abstr the concepts of the synthesizable subset in the context of the abstraction levels defined for TLM

Public review of the draft is now closed. The draft is available for download here.

Join this Working Group

If you are an employee of a member company and would like to join this working group, click here (requires login) and click Join Group. WG participation requires right of entry by the group chair.

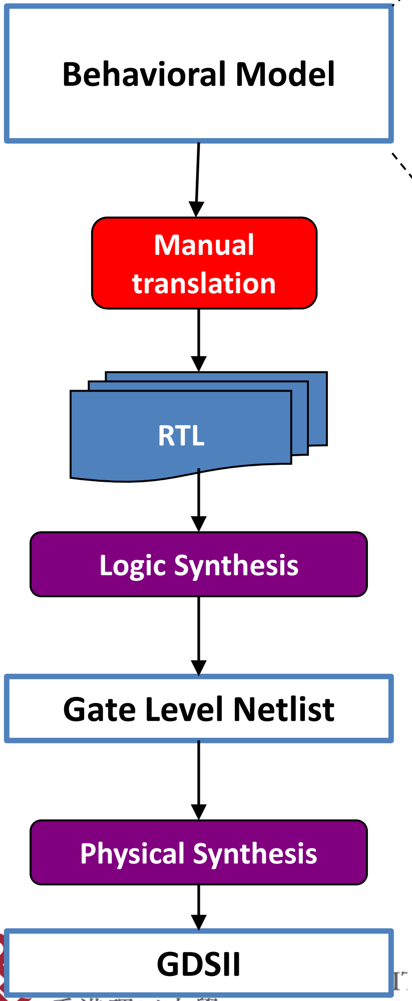
Resources

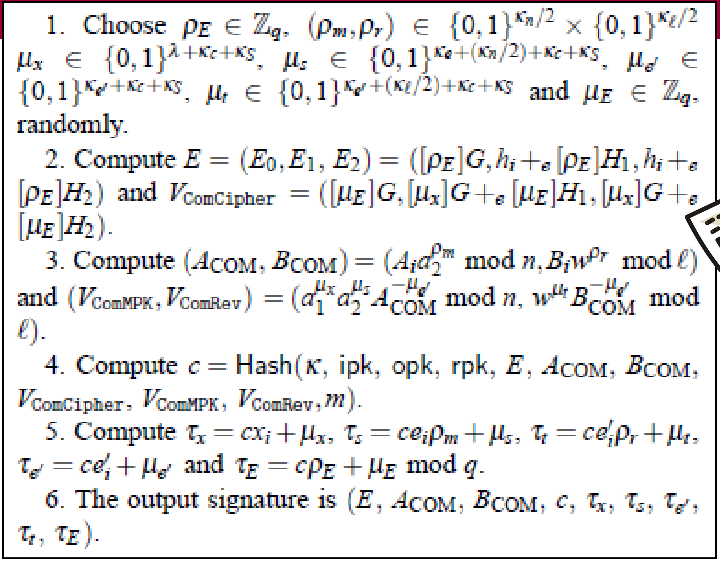
- Video tutorial: "OSCI TLM-2.0 Standard and Synthesizable Subset," presented at DVCon 2010

The standard is intended for use by logic designers, electronic engineers and design automation tool developers.

The following team members drove the Draft 2.0 effort:
Mike Meredith
Benjamin Carrion Schafer
Alan P. Su
Andres Takach, Chair
Jos Verhaegh

VLSI Flow Overview





1. Choose $\rho_E \in \mathbb{Z}_q$, $(\rho_m, \rho_r) \in \{0,1\}^{\kappa_n/2} \times \{0,1\}^{\kappa_\ell/2}$ $\mu_x \in \{0,1\}^{\lambda+\kappa_c+\kappa_S}$, $\mu_s \in \{0,1\}^{\kappa_e+(\kappa_n/2)+\kappa_c+\kappa_S}$, $\mu_{e'} \in \{0,1\}^{\kappa_{e'}+\kappa_c+\kappa_S}$, $\mu_r \in \{0,1\}^{\kappa_{e'}+(\kappa_\ell/2)+\kappa_c+\kappa_S}$ and $\mu_E \in \mathbb{Z}_q$, randomly.

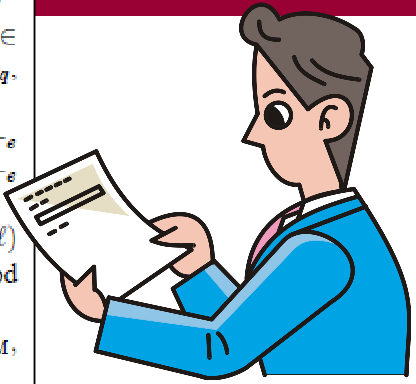
2. Compute $E = (E_0, E_1, E_2) = ([\rho_E]G, h_i +_e [\rho_E]H_1, h_i +_e [\rho_E]H_2)$ and $V_{\texttt{ComCipher}} = ([\mu_E]G, [\mu_x]G +_e [\mu_E]H_1, [\mu_x]G +_e [\mu_E]H_2)$.

3. Compute $(A_{\text{COM}}, B_{\text{COM}}) = (A_i a_2^{\rho_m} \bmod n, B_i w^{\rho_r} \bmod \ell)$ and $(V_{\texttt{ComMPK}}, V_{\texttt{ComRev}}) = (a_1^{\mu_x} a_2^{\mu_s} A_{\text{COM}}^{-\mu_{e'}} \bmod n,\ w^{\mu_r} B_{\text{COM}}^{-\mu_{e'}} \bmod \ell)$.

4. Compute $c = \mathsf{Hash}(\kappa, \mathsf{ipk}, \mathsf{opk}, \mathsf{rpk}, E, A_{\text{COM}}, B_{\text{COM}}, V_{\texttt{ComCipher}}, V_{\texttt{ComMPK}}, V_{\texttt{ComRev}}, m)$.

5. Compute $\tau_x = cx_i + \mu_x$, $\tau_s = ce_i\rho_m + \mu_s$, $\tau_r = ce_i'\rho_r + \mu_r$, $\tau_{e'} = ce_i' + \mu_{e'}$ and $\tau_E = c\rho_E + \mu_E \bmod q$.

6. The output signature is $(E, A_{\text{COM}}, B_{\text{COM}}, c, \tau_x, \tau_s, \tau_{e'}, \tau_r, \tau_E)$.



Software Algorithm Designer

Hardware Designer

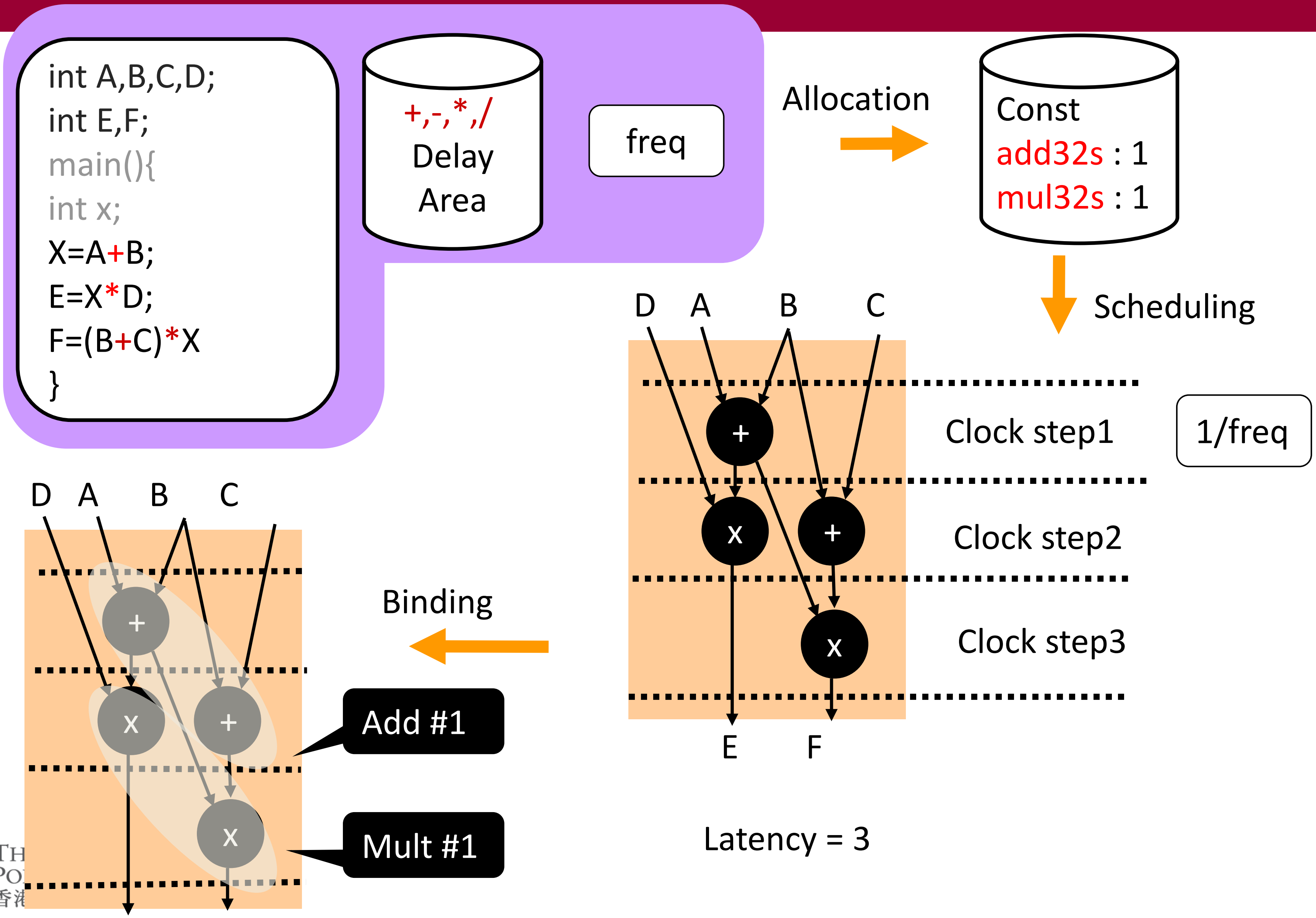
High Level Synthesis

- Definition

  "Automatic conversion of behavioral, untimed descriptions into hardware that efficiently implements that behavior"

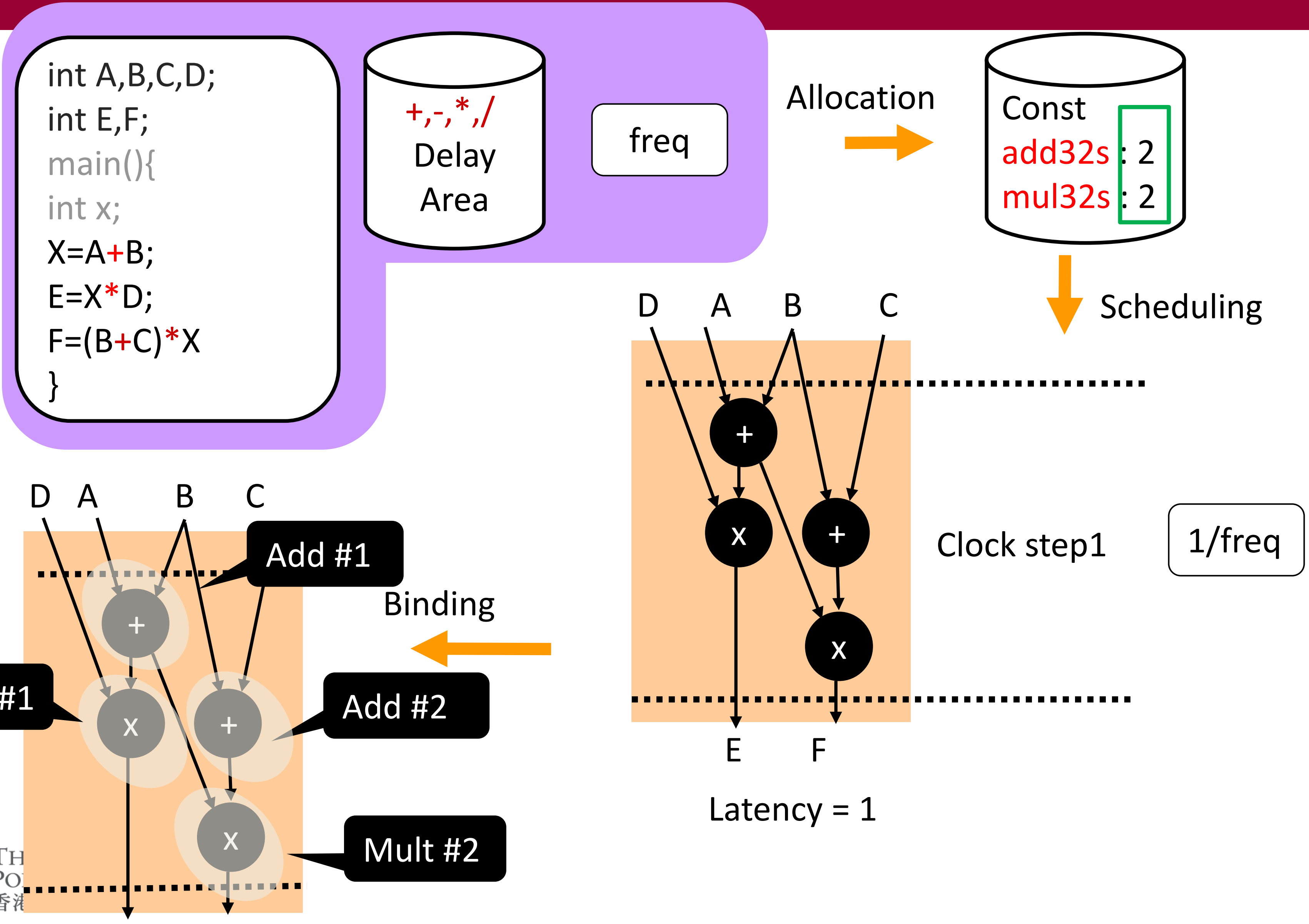
- Main Steps
  - Allocation
    - Specify the hardware resources that will be necessary
  - Scheduling
    - Determine for each operation the time at which it should be performed such that no precedence contraint is violated
  - Binding
    - Provide a mapping from each operation to a specific functional unit and from each variable to a register

High Level Synthesis Overview



High Level Synthesis Overview cont.

```
int A,B,C,D;
int E,F;
main(){
int x;
X=A+B;
E=X*D;
F=(B+C)*X
}
```

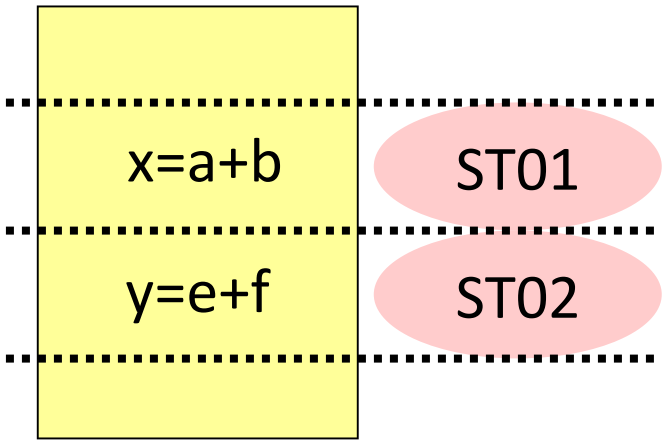


HLS Resources Constraints

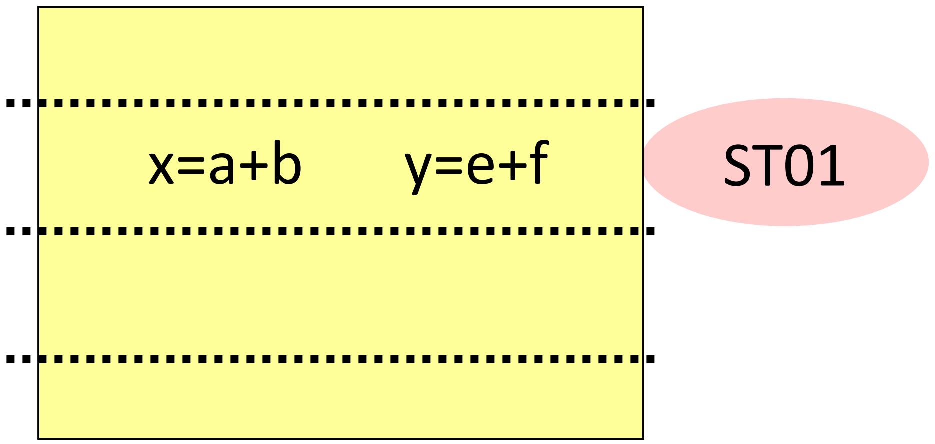
- Functional Unit Constraint file specifies how many FUs can be instantiated → Impacts the synthesized architecture

```
int main(){
  x = a+b;
  y = e+f;
}
```

1 Adder

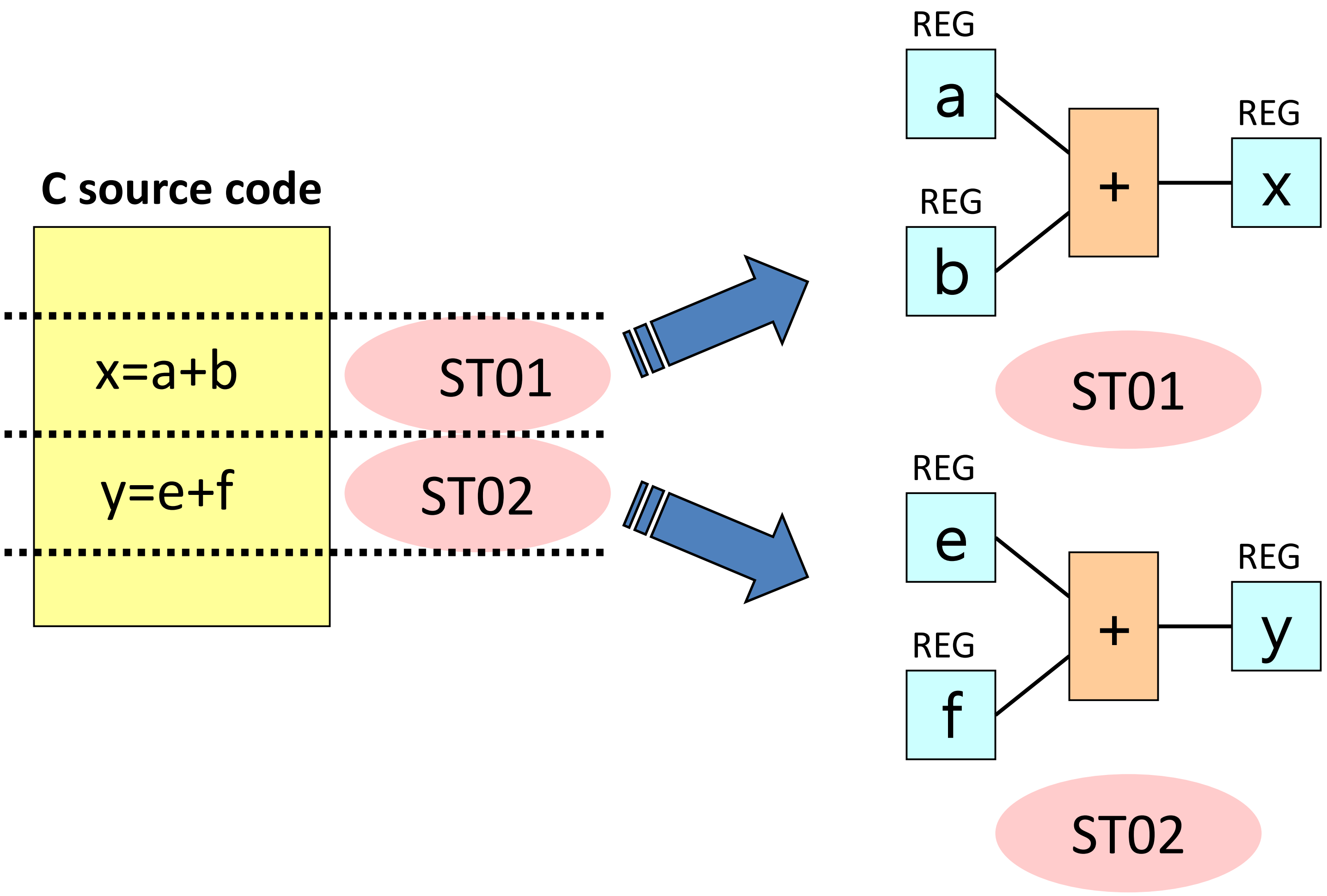


2 Adders

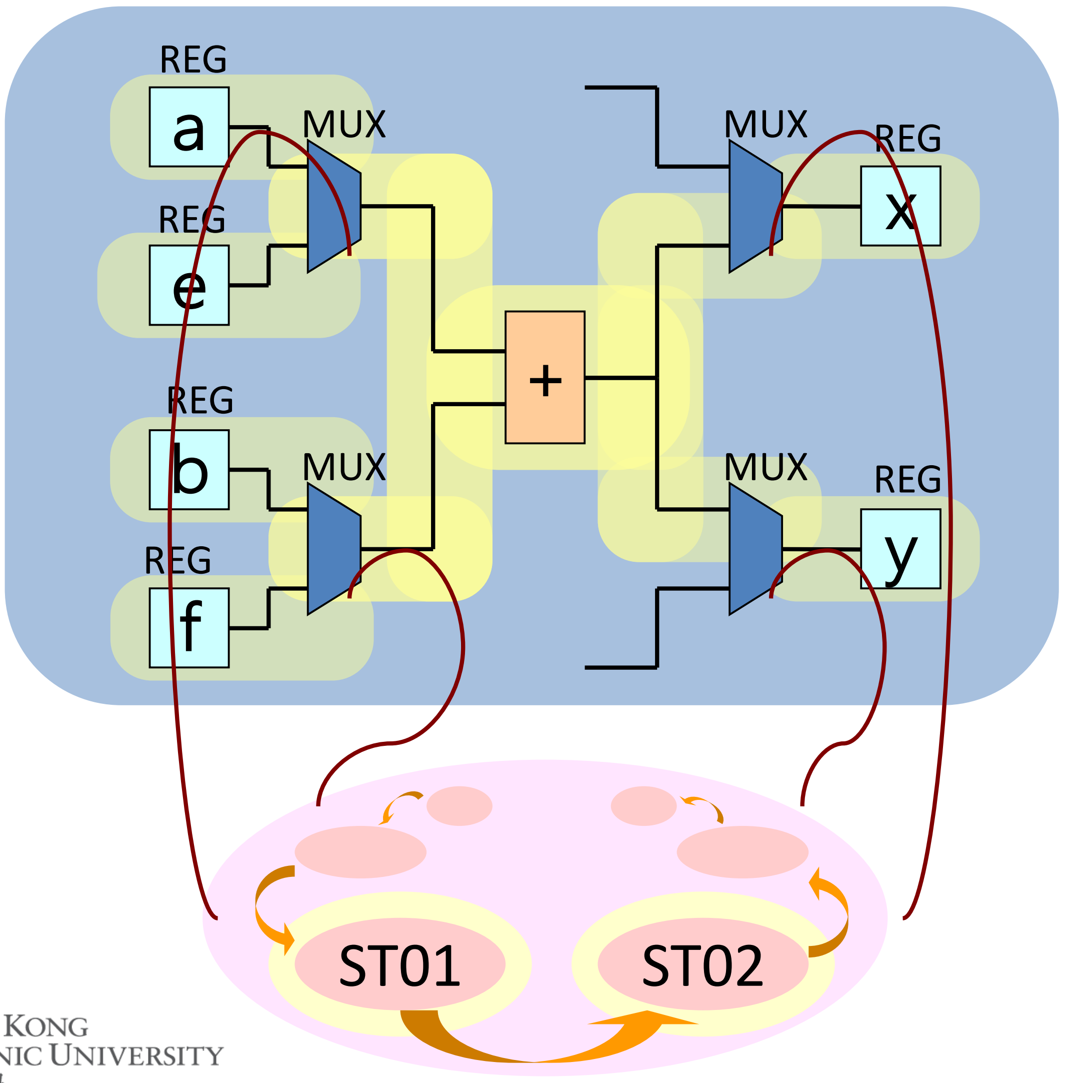


THE HONG KONG POLYTECHNIC UNIVERSITY 香港理工大學

Resource Constraint: Min FUs (Resource Sharing)



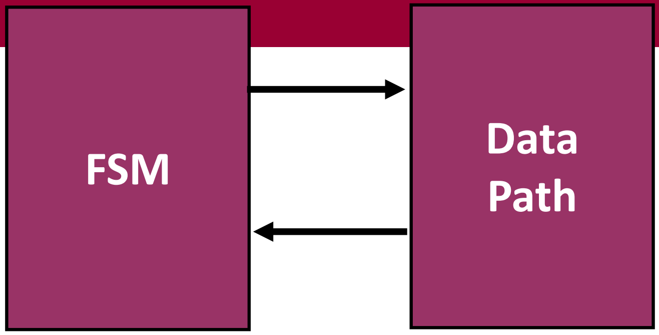
High Level Synthesis – Min FUs (Resource Sharing)



Data Path

FSM

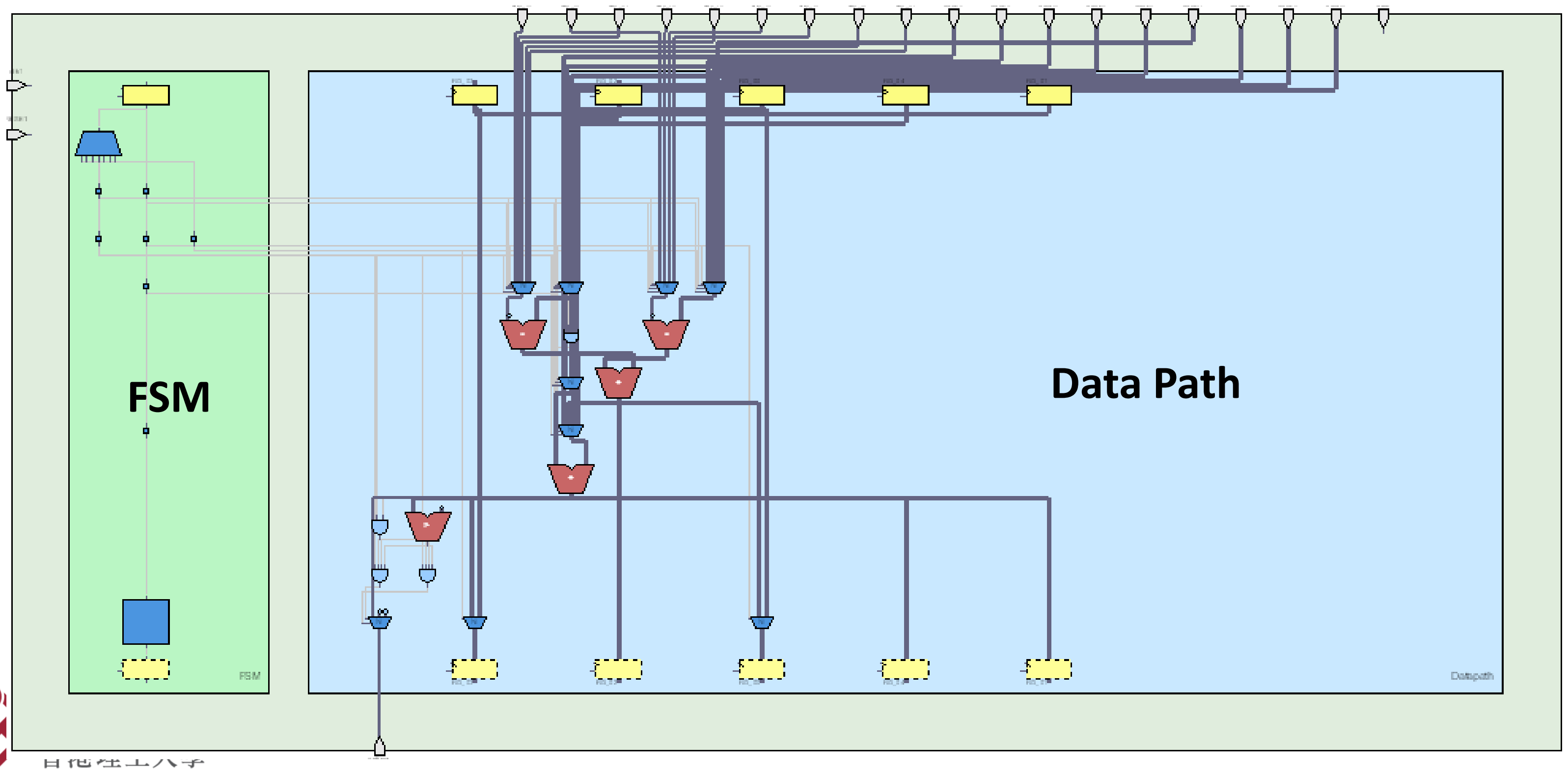
Target Architecture



Finite State Machine with Data Path

FIR Filter example

```
for(i=0;i<9;i++)
        sum += ary[i] * coeff[i] ;
```



Fmax (delay) vs. FUs delay

- Very important for good synthesis result
- Latency will change

```
int main(){
  x = a+b;
  y = e+f;
}
```

F = 100Mhz = 10ns

Delay adder =5ns

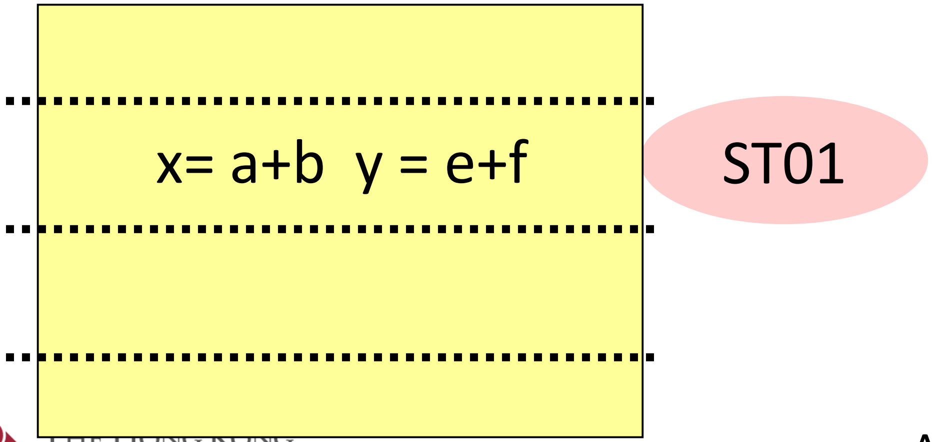
FCNT = 2 adders

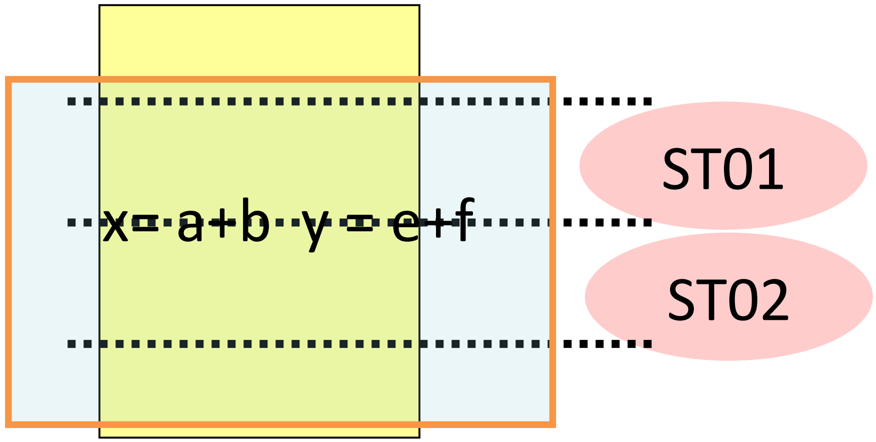
```
int main(){
  x = a+b;
  y = e+f;
}
```

F = 200Mhz = 5ns

Delay adder =6ns

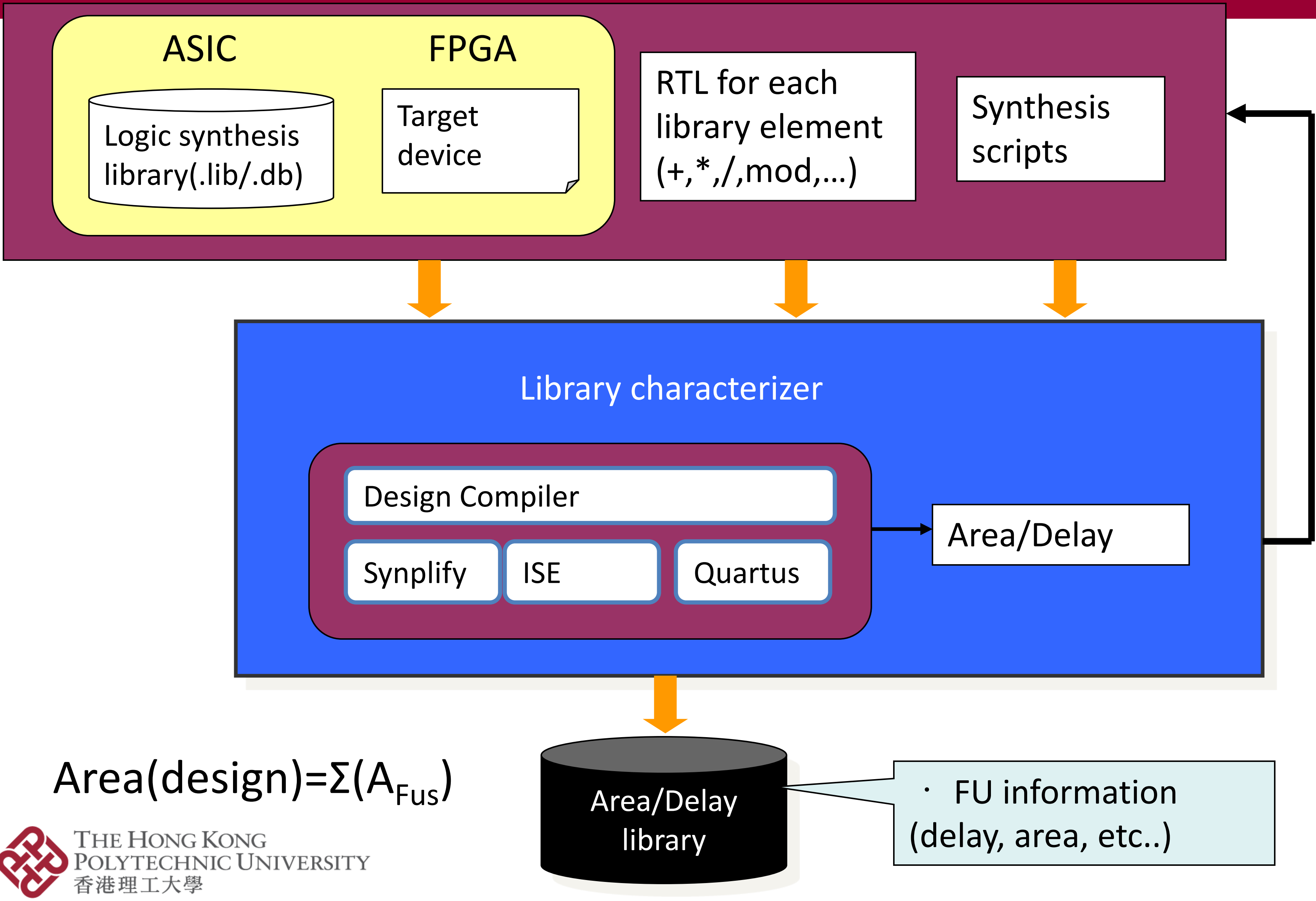
FCNT = 2 adders





Achieved: (1) Multi-cycle operation or (2) pipelining

Library Characterizer



$$Area(design)=\Sigma(A_{Fus})$$

THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

Library Format

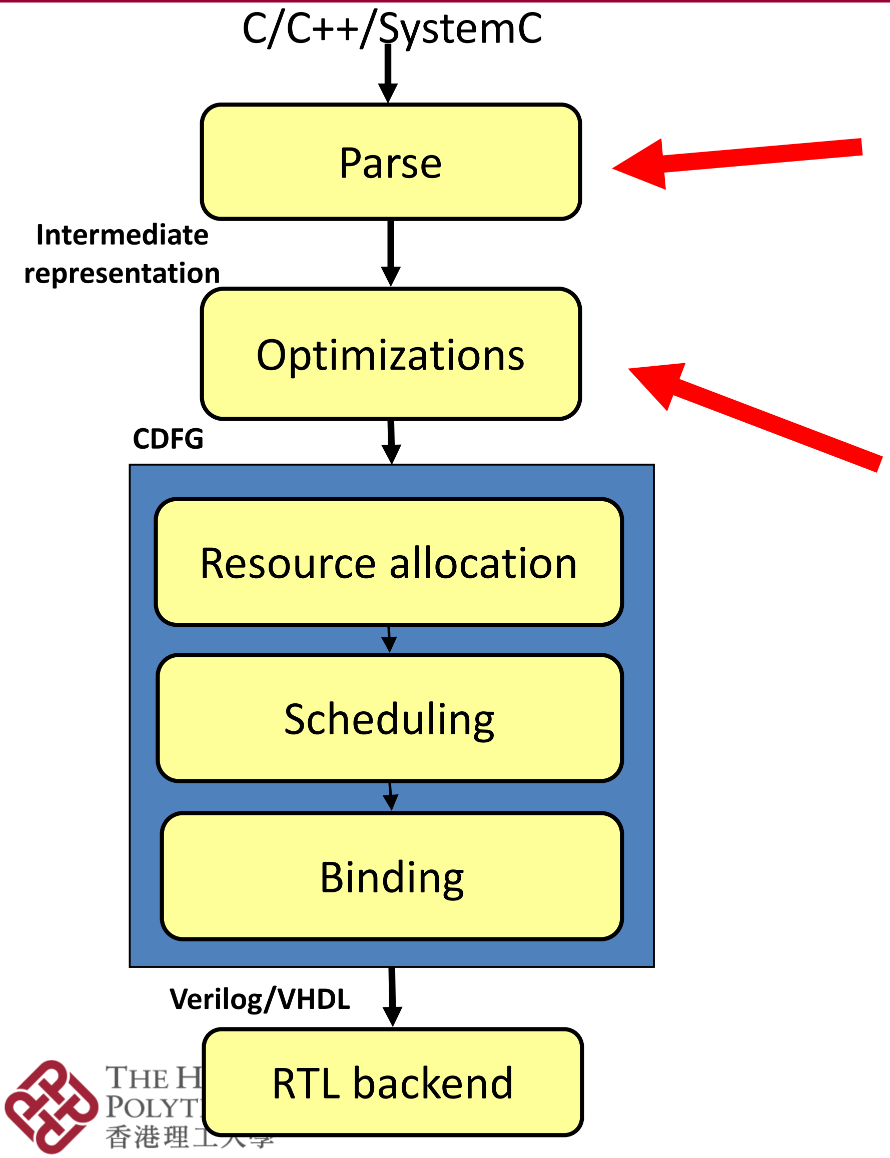
- Vendor dependent. No standard format like in logic synthesis (.lib, .db).
- E.g.:

```
@FLIB {
   NAME       mul4s
   KIND       *
   BITWIDTH   4
   DELAY      264
   SIGN       SIGNED
   SYN_TOOL   ISE
   LSTAGE     1
}
```

```
@FLIB {
   NAME       mul8u
   KIND       *
   BITWIDTH   8
   DELAY      362
   SIGN       UNSIGNED
   SYN_TOOL   ISE
   LSTAGE     1
}
```

THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

HLS Flow Overview



Compiler Front End
(lex and parse)

Compiler optimizations

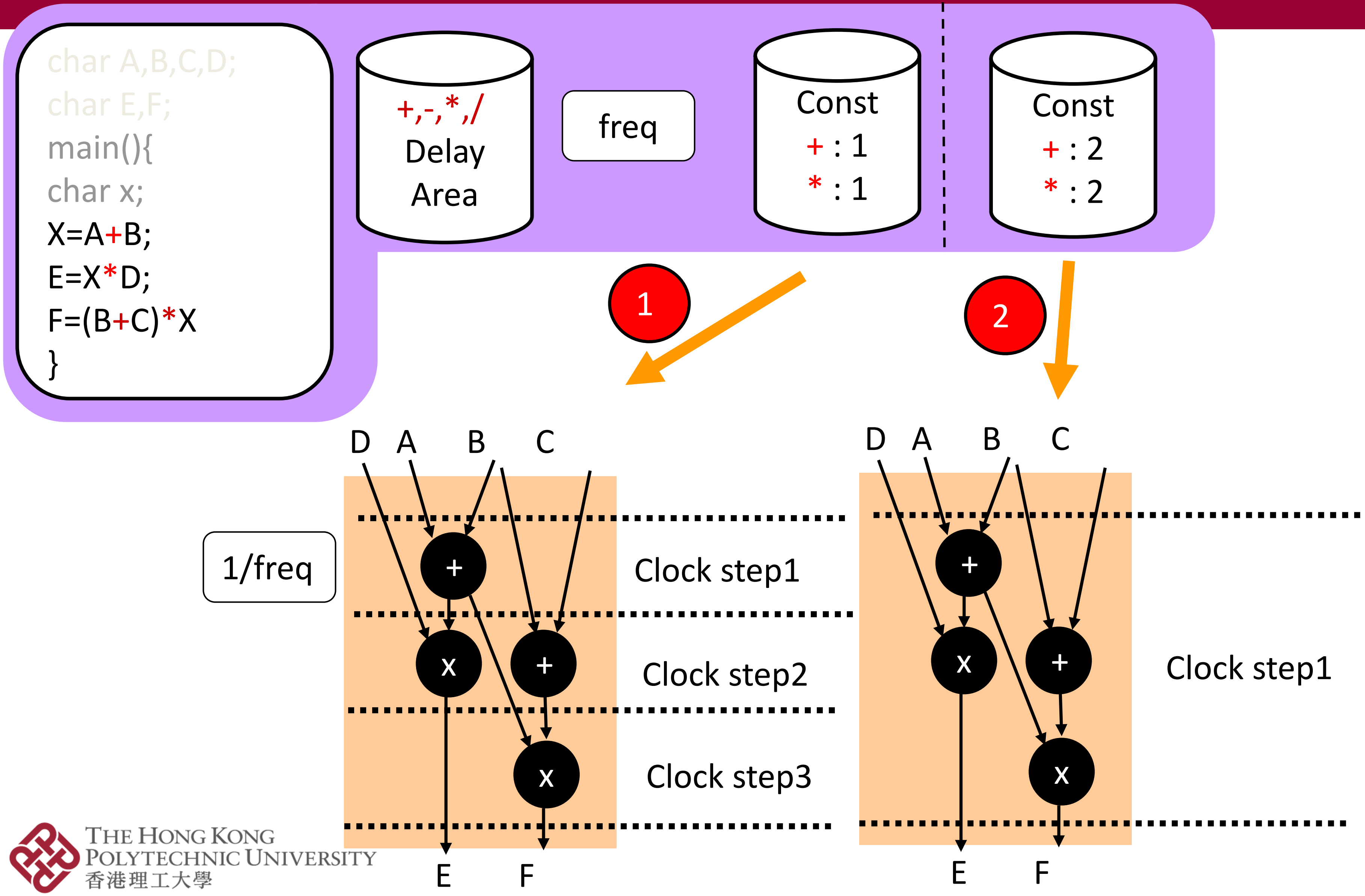
Constant/Variable propagation, common sub-expression elimination, dead-code elimination, etc..

Synthesis transformations

Loop unrolling, array expansion, speculation, conditional expansions, tree high reduction

---

Benefits of HLS: (1) Automatic Alternative Architecture Generation



```
char A,B,C,D;
char E,F;
main(){
char x;
X=A+B;
E=X*D;
F=(B+C)*X
}
```

Benefits of HLS: (1) Automatic Alternative Architecture Generation

Behavioral Description in C

```
char A,B,C,D;
char E,F;
main(){
char X;
X = A + B;
E = X * D;
F = (B + C) * X;
}
```

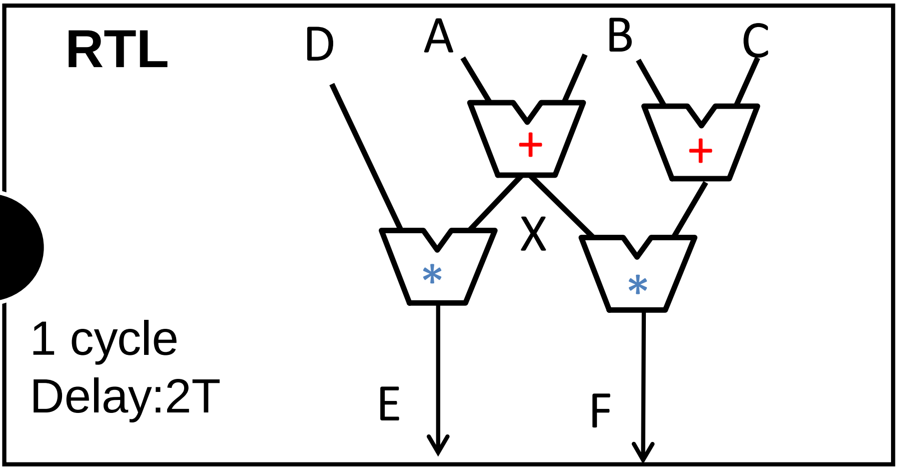
+ : 2
* : 2

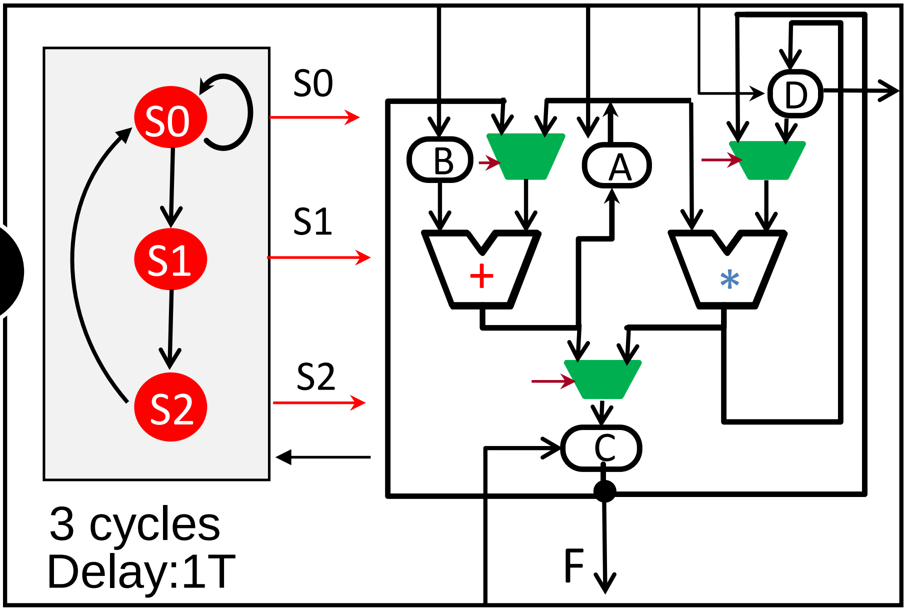
1

FU constraints

+ : 1
* : 1

2



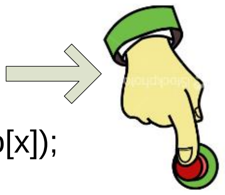


THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

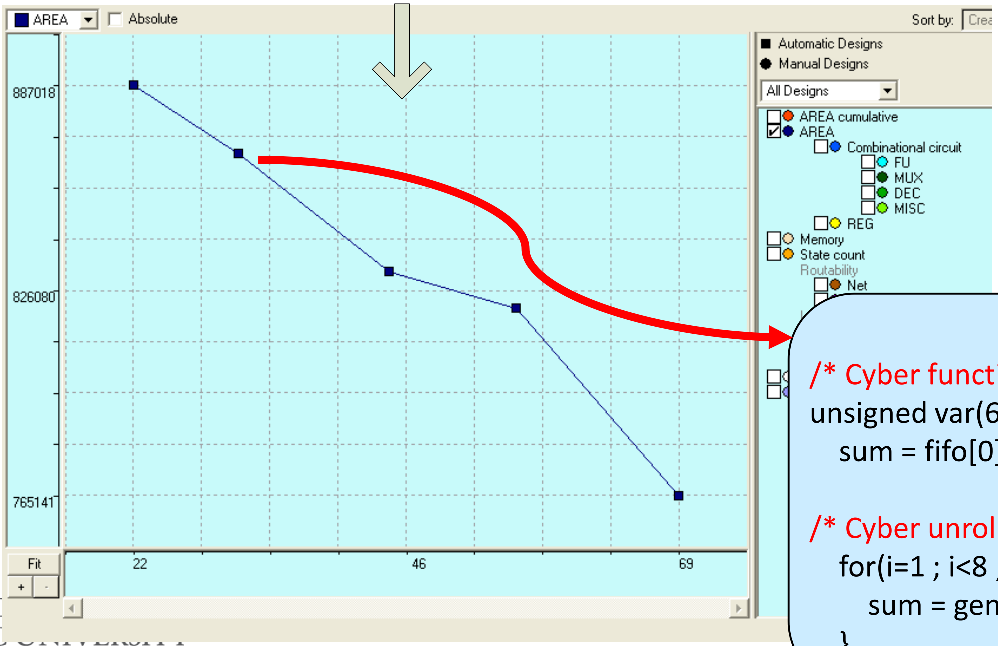
Benefits of HLS: (2) Architectural Design Space Exploration

C code:

```
unsigned var(63..0) add{
   sum = fifo[0];

   for(i=1;i<8;i++){
          sum = gen_sum(I,fifo[x]);
   }
```



- Resource sharing control (number of FUs)
- Synthesis directives (pragmas)
- Global synthesis options



```
/* Cyber function = inline */
unsigned var(63..0) add{
   sum = fifo[0];

/* Cyber unroll_times = all */
   for(i=1 ; i<8 ; i++){
      sum = gen_sum(I, fifo[x]):
   }
```

Pragma-based Design Space Exploration

- Synthesis directives inserted as comments in the source code (pragmas)
  - Loops (unroll, pipeline)
  - Arrays (register, RAM, ROM, logic, expand)
  - Functions (inline or not)

```
process main(){
  in  ter(0:8) in0 ;
  out ter(0:8) out0 ;
  var(0:8) sum;
  var (0:8) fifo[8] /* Cyber array=reg */{ 0, 0, 0, 0, 0, 0, 0, 0 } ;
  var(0:4) i ;

   /* Cyber unroll_times=0 */
   for( i = 7 ; i > 0 ; i-- )
     fifo[i] = fifo[i-1] ;

   fifo[0] = in0;

   /* Cyber unroll_times=0 */
   for(i=1;i<8;i++){
            /* Cyber func=inline */
             sum = addition(sum, fifo[i]);
}
   out0 = sum / 8;
}
```

Explore 1

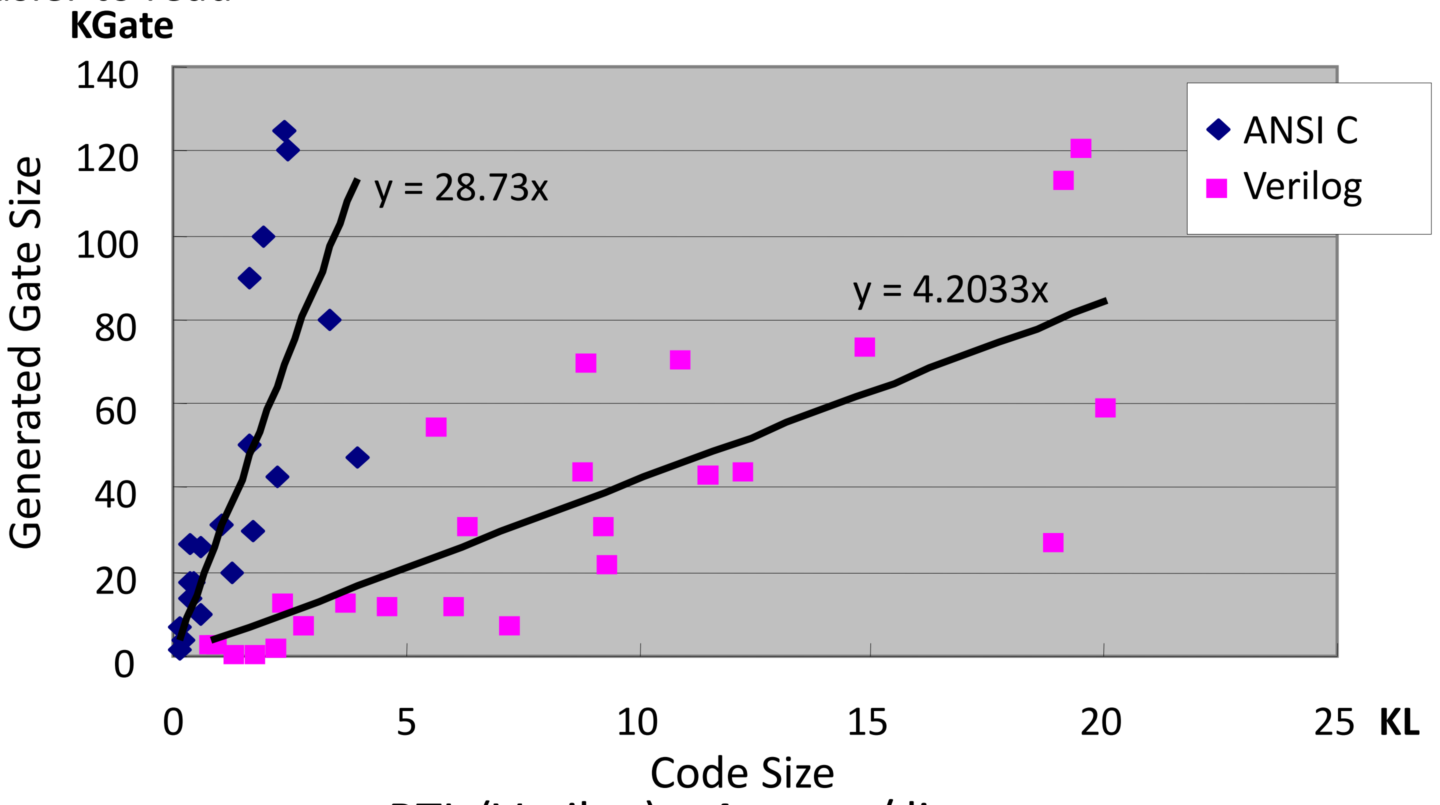
Explore 2

Explore 3

array	loop1	loop2	function	Area	Latency
ram	0	0	goto	1362	24
ram	all	all	inline	2684	19
reg	0	all	inline	2915	6
reg	0	0	inline	3544	5
reg	all	all	inline	4352	1

Benefits of HLS: (3) Increase Productivity

- Less details needed ➔ Faster to design and verify
- Less bugs
- Easier to maintain source code
- Easier to read



RTL (Verilog) 4 gates / line

Behavioral C 28 gates / line

➔ **7x more productive**

THE HONG KONG POLYTECHNIC UNIVERSITY 香港理工大學

Benefits of HLS: (4) Less Bugs

$$\frac{\text{Bugs}}{\text{Chip}} = \frac{\text{Logic Transistors}}{\text{Chip}} \times \frac{\text{Lines of code}}{\text{Logic Transistors}} \times \frac{\text{Bugs}}{\text{Lines of code}}$$

Example:

$$\frac{\text{100 Bugs}}{\text{Chip}} = \frac{\text{10,000,000 trans}}{\text{Chip}} \times \frac{1}{10} \times \frac{1}{10,000}$$

Moore's Law

Level of abstraction

→Moore's Law implies more bugs

Resource Allocation

- Specifies the hardware resources that will be necessary

```
int A,B,C,D;
int E,F;
main(){
int x;
X=A+B;
E=X*D;
F=(B+C)*X
}
```

Allocation ➔

Const
add32s : 1
mul32s : 1

```
char A,B,C,D;
char E,F;
main(){
char x;
X=A+B;
E=X*D;
F=(B+C)*X
}
```

Allocation ➔

Const
add8s : 1
mul8s : 1

➔ How to represent variable bitwidths in ANSI-C?
(e.g. 12 bits, 17bits)

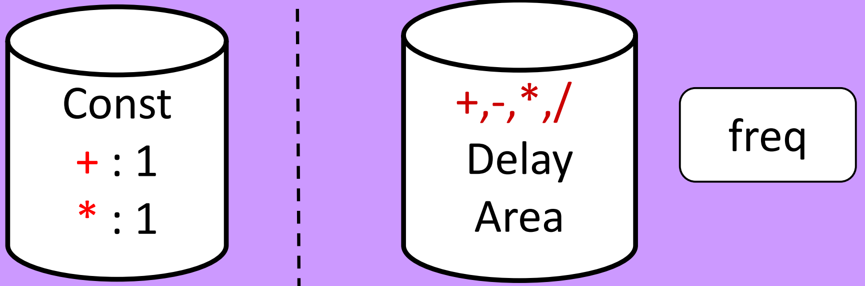
Scheduling : Constraint (Resources, Time)

- Resource constraint:
  - Given a set *O* of operations with a partial ordering, a set *K* of functional unit types, a type function, $\sigma: O \rightarrow K$, to map the operations into the functional unit types, and resource constraints m_k for each functional unit type.
  - Find a (optimal) schedule for the set of operations that obeys the partial ordering (Minimize latency)
    - ➔ ASAP, ALAP, List scheduling

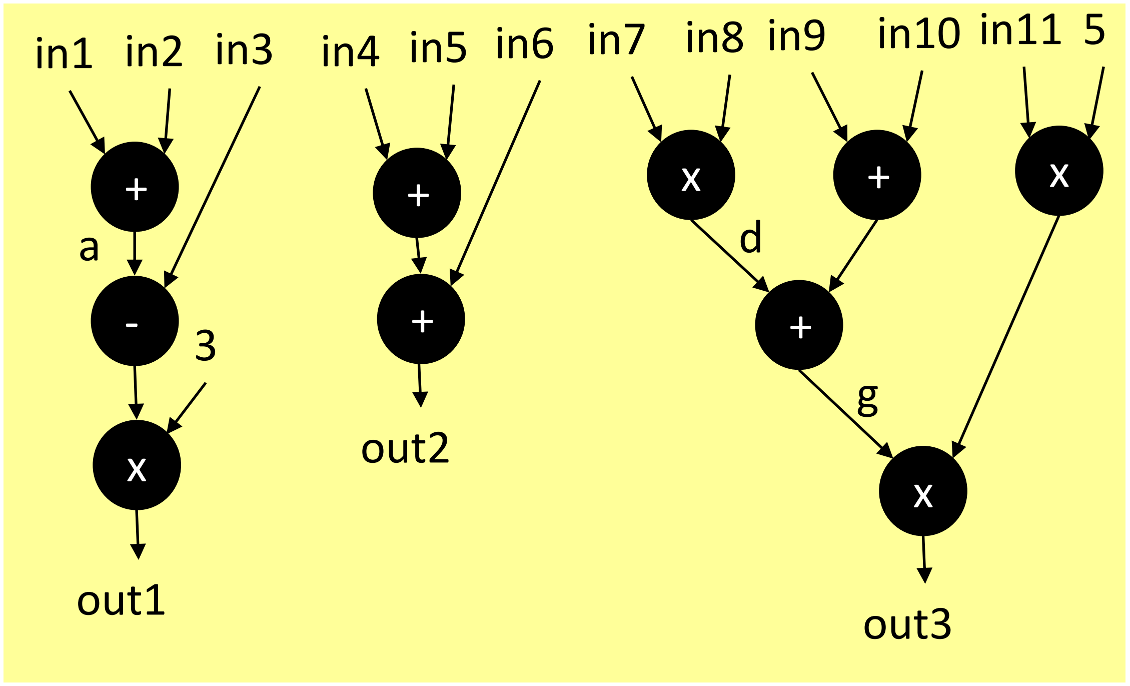
- Timing constraints
  - Minimize the resources given a fixed timing
    - ➔ Force directed scheduling

Example

```
int main(){
int in1, in2, in3, in4,…,in11;
int a, d, g;
int out1, out2, out3;
a= in1+in2;
out1 = (a-in3) * 3;
out2 = in4 + in5 + in6;
d = in7 * in8;
g = d + in9 + in10;
out3 = in11 * 5 * g;
}
```



CDFG

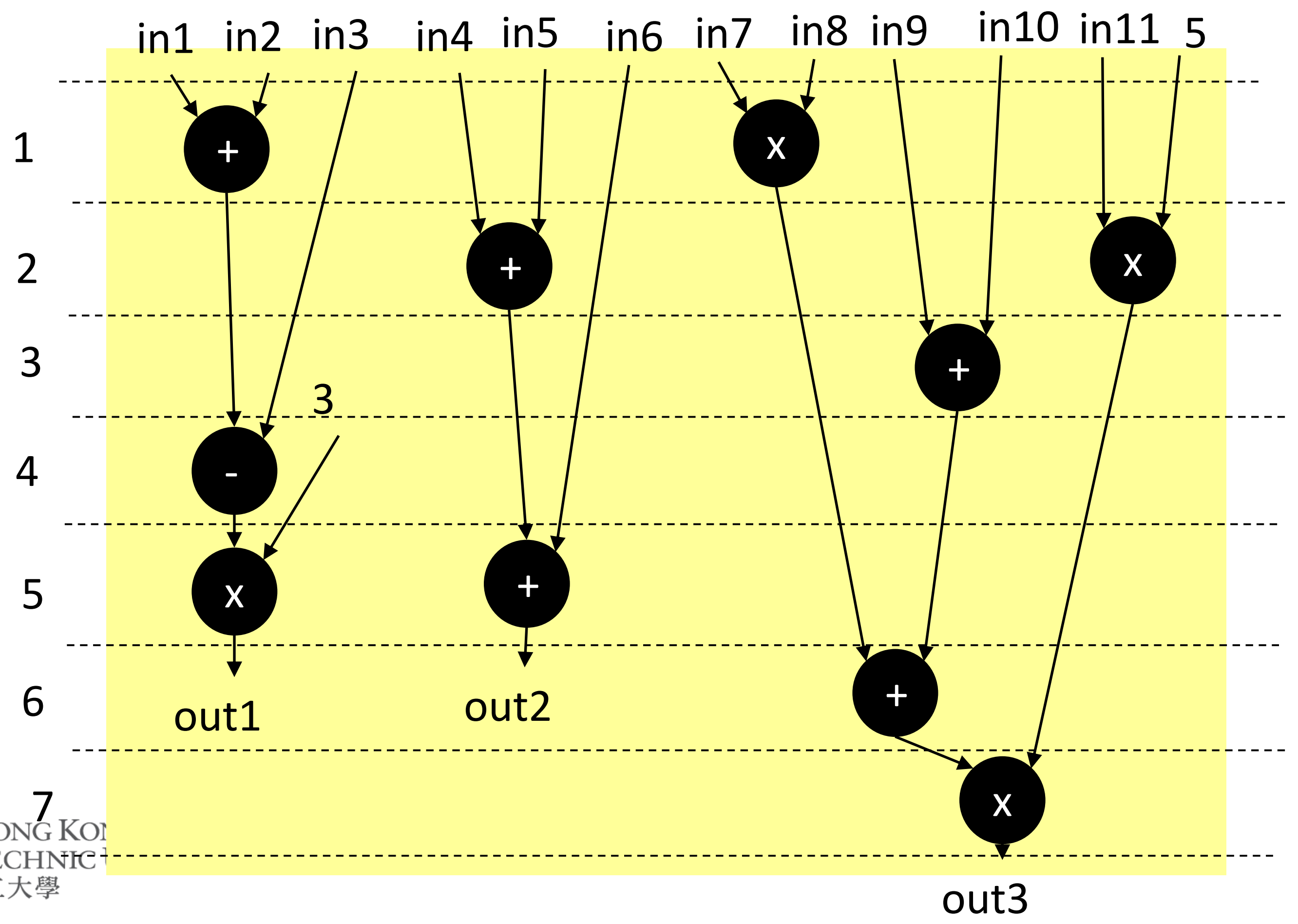


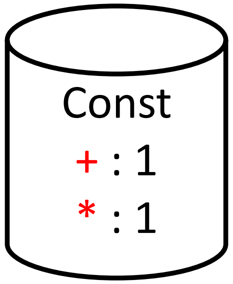
Scheduling (I): ASAP

- Map operations to their *earliest* possible start time not violating the precedence constraints
- Easy and fast to compute
  - Find longest path in a directed acyclic graph
  - No attemp to optimize ressource cost
- Gives the fastest possible schedule if unlimited amount of resources are available
- Gives an upper bound on execution speed

ASAP Scheduling

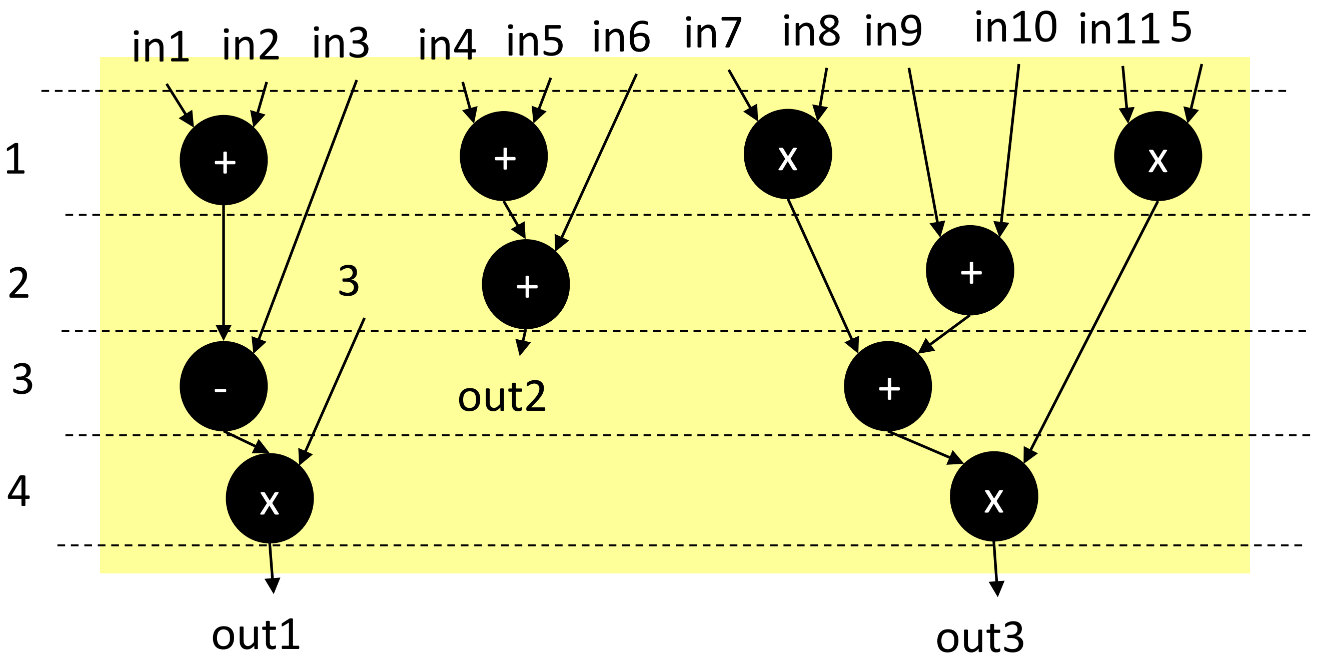
- Sort operation topologically according to their dependence
- Schedule operations in sorted order by placing them in the <u>earliest</u> possible control step

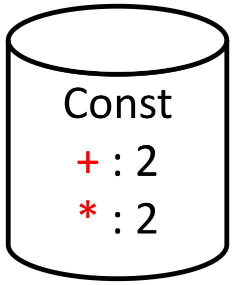




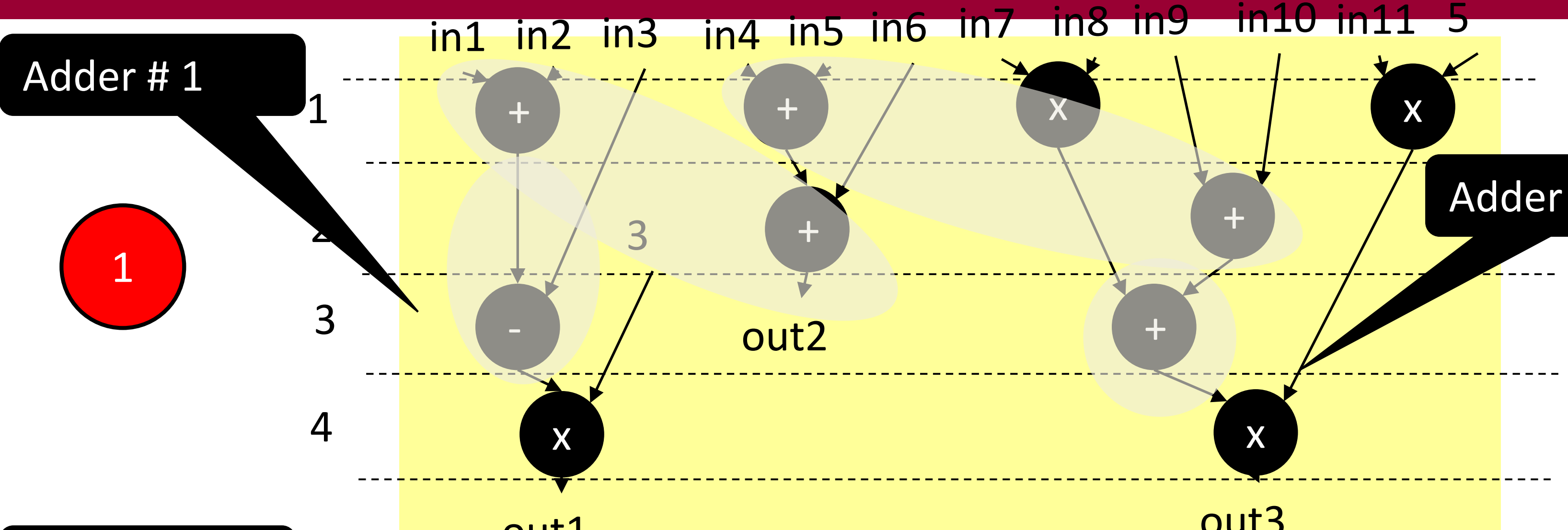
Binding for ASAP Example with new FU Constraint

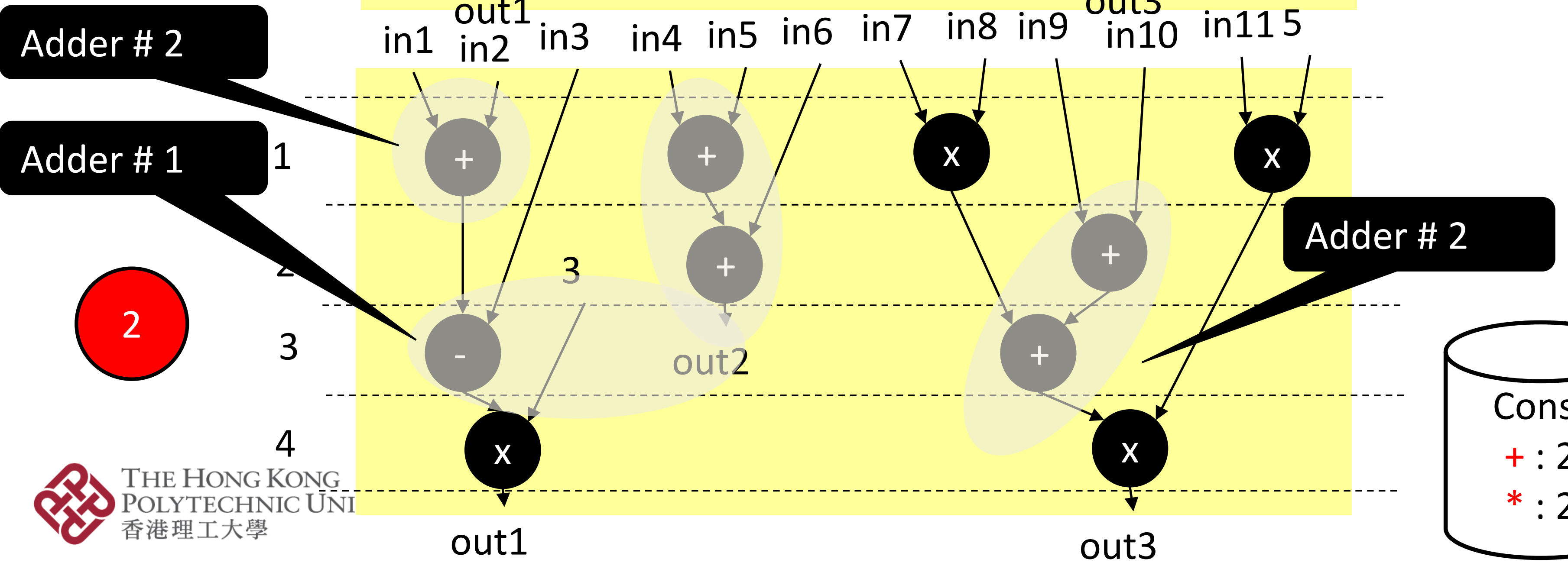
- Assign Operators to particular FUs
- ASAP with 2 adders and 2 multipliers





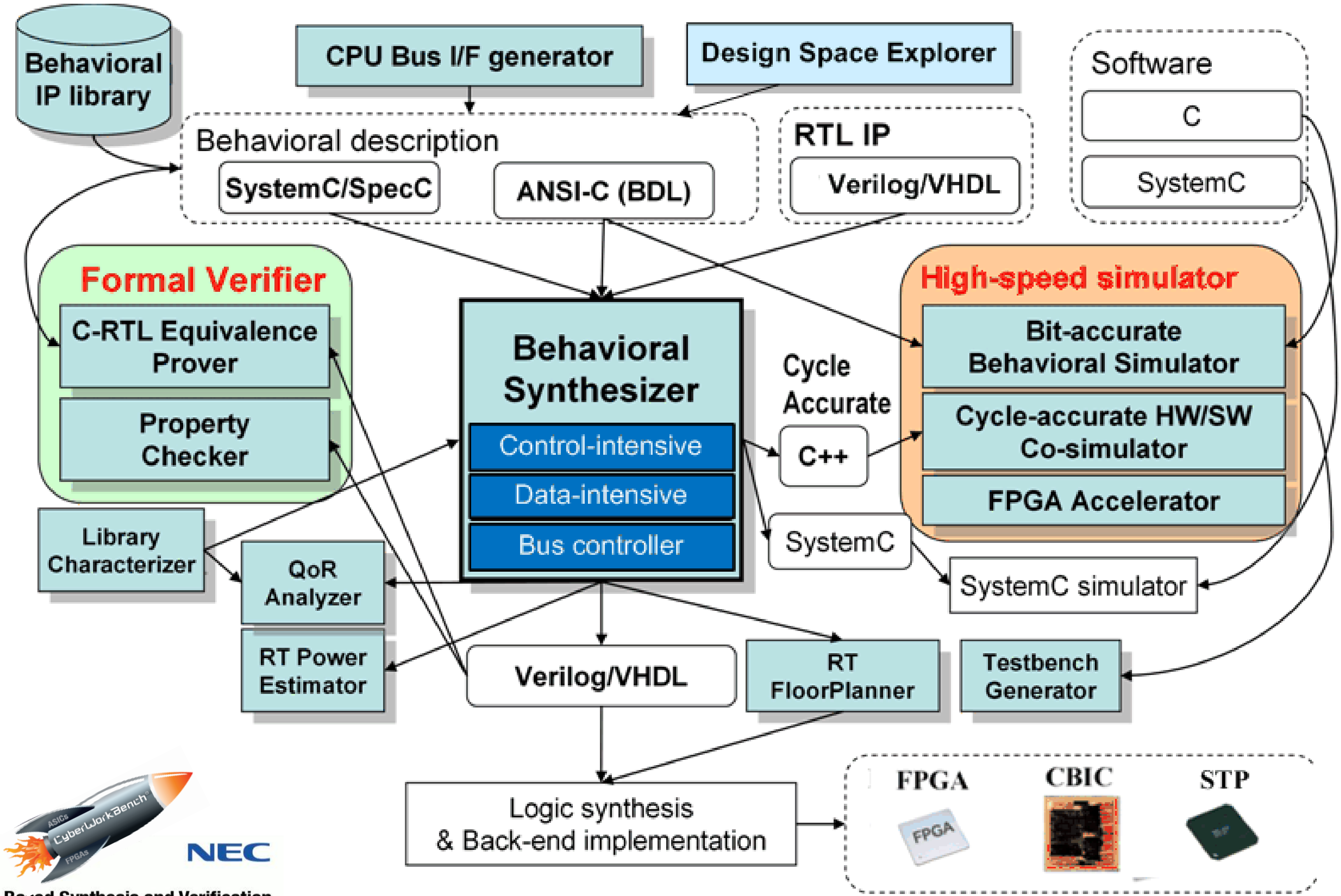
Binding Example con't





Const
+ : 2
* : 2

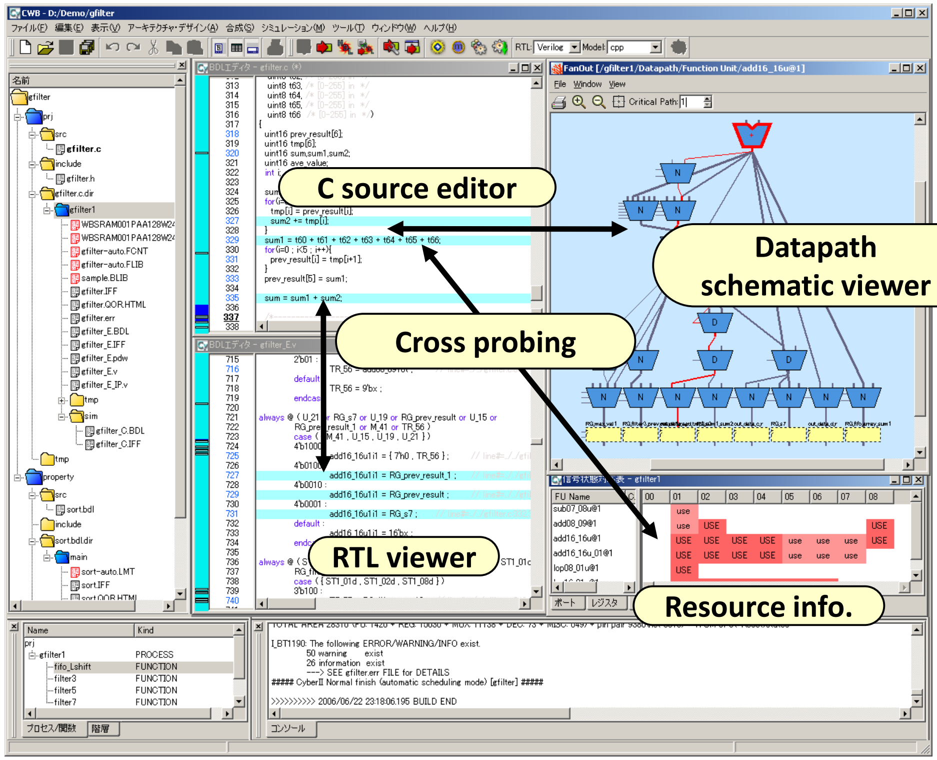
NEC - CyberWorkBench



CyberWorkBench
ASICs
FPGAs
NEC
Behavioral C-Based Synthesis and Verification
with CyberWorkBench
Taping out commercial chips since 1993
developed All-in-C

CyberWorkBench's IDE

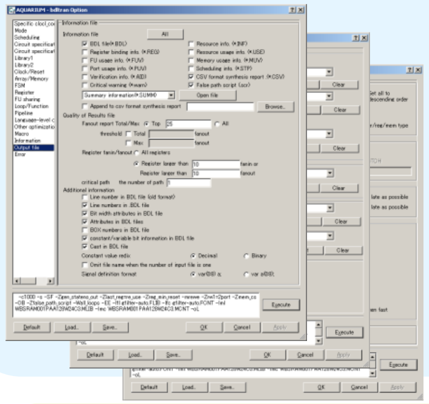
Integrated GUI


CWB - D:/Demo/gfilter
ファイル(F) 編集(E) 表示(V) アーキテクチャ・デザイン(A) 合成(S) シミュレーション(M) ツール(T) ウィンドウ(W) ヘルプ(H)
RTL: Verilog
Model: cpp
BDLエディタ - gfilter.c (*)
BDLエディタ - gfilter_E.v
FanOut [/gfilter1/Datapath/Function Unit/add16_16u@1]
Critical Path
C source editor
Datapath schematic viewer
Cross probing
RTL viewer
Resource info.
ポート
レジスタ
コンソール

```
 uint8 t63, /* [0-255] in */
 uint8 t64, /* [0-255] in */
 uint8 t65, /* [0-255] in */
 uint8 t66  /* [0-255] in */)
{
  uint16 prev_result[6];
  uint16 tmp[6];
  uint16 sum,sum1,sum2;
  uint16 ave_value;
  int i;

  sum
  for(i=
    tmp[i] = prev_result[i];
    sum2 += tmp[i];
  }
  sum1 = t60 + t61 + t62 + t63 + t64 + t65 + t66;
  for(i=0 ; i<5 ; i++){
    prev_result[i] = tmp[i+1];
  }
  prev_result[5] = sum1;

  sum = sum1 + sum2;
```

```
I_BT1190: The following ERROR/WARNING/INFO exist.
          50 warning     exist
          26 information  exist
          ---> SEE gfilter.err FILE for DETAILS
##### CyberII Normal finish (automatic scheduling mode) [gfilter] #####

>>>>>>>>>> 2006/06/22 23:18:06.195 BUILD END
```

Synthesis Controllability



Rich synthesis function
- Parallelization
- Pipelining
- Behavioral IP reuse
- Controller synthesis
- Script generation
- Various circuit support
- Synchronous/asynchronous/ pipelined memory
- Synchronous/asynchronous reset
- Multiple clock, gated clock
- RTL style selection

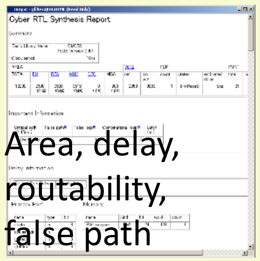
Useful feedback to the designers

QoR report

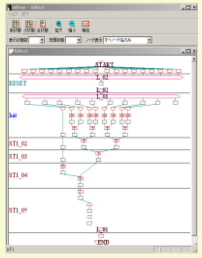
Dataflow diagram

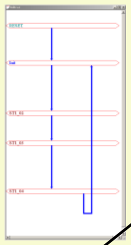
State transition

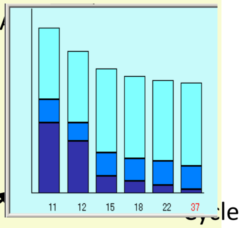
Tradeoff chart



Area, delay, routability, false path







Automatic exploration

C for Hardware

- ANSI-C does not have custom data types. How to specify 12 or 17 bits?
- How to specify I/Os? (entity)

  ➔Need HW extensions
- CWB calls it BDL (Behavioral Description Language)
- Same as ANSI-C but need:
  – Rename synthesizable function as “process”
  – Declare Inputs and outputs

Pre-compiler Directives

- Used to separate different versions of the same program

```
#ifdef WIN32
        system("dir *");
#elseif LINUX
        system("ls –ltra");
#endif
```

- Variables WIN32/LINUX can be specified at source code or Makefile or gcc command line
  - Source code: #define WIN32
  - Makefile : CFLAGS = DWIN32
  - Gcc command line : gcc foo.c –DWIN32

BDL Data types

- 3 different HW specific data types:
- Terminal (ter) – does not hold values across the clock cycle boundaries
  - Register (reg) – holds values across the clock cycle boundaries
  - Var (var) – wire or register (decided by HLS tool)
- Standard data types bitwidths

int	32 bits	char	8 bits
short	16 bits	long	32 bits
long long	64 bits	bool	1 bit

- Custom bitwidth specification
  - type(7..0) or type(0:8).E.g:

```
var(0:4)  a;    /* Ascending order:  start bit 0, bit width 4 */
reg(7..0) b;    /* Descending order: start bit 7, end bit 0 */
ter(0..3) c;    /* Ascending order:  start bit 0, end bit 3 */
```

THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

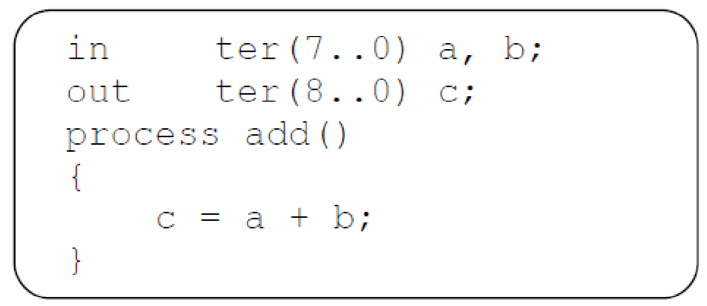
BDL Inputs/Outputs

- Specify inputs and outputs using in/out keywords

```
in     var(0:4)  a;  /* "in" variable of var type */
out    reg(7..0) b;  /* "out" variable of reg type */
inout  ter(0..3) c;  /* "I/O" variable of ter type */
```

BDL - Process

- In C language, the function called "main”. In BDL it is called “process”



```
in      ter(7..0)  a,  b;
out     ter(8..0)  c;
process  add()
{
        c  =  a  +  b;
}
```

```
in ter(7..0) a, b;
out ter(8..0) c;
process add(){
      int a_in, b_in;      // declare reg
      a_in=a;b_in=b;   // read inputs
      c=a_in+b_in;}
```

- In order to avoid having to declare intermediate variables → use input()/output functions (declare in/out as var type)
- Every BDL Program requires :
  - IO declarations
  - Process declaration

```
in var(7..0) a, b;
out ter(8..0) c;
process add(){
input(a);
input(b);   // read inputs
      c=a+b;}
```

Bitwidth operations

- Concatenation

```
o(0:16) = a(0:8)::b(0:8);
```

- Or Reduction

```
f(0:1) = ^>a(0:4);
```

- Constants

```
x(0:4) = 0b1010; /* "10" as 4-bit value */
x(0:6) = 017;    /* "15" as 6-bit value */
x(0:12) = 0x0ff; /* "255" as 12-bit value */
```

Typical Applications

Traditionally Fit for HLS

Data Intensive

Arithmetic operations
Simple algorithm

-FIR, FFT, ...

-secret key Encryption

-simple ECC, EDC

-graphic decoding

Traditionally **NOT** fit for HLS

Control Intensive

Arithmetic operation in
Complex control

- Video Voice recognition
- Data compression
- Complex CODEC
- DRM
- Turbo ECC
- Public Key Encryption

Traditionally **NOT** fit for HLS

Controller

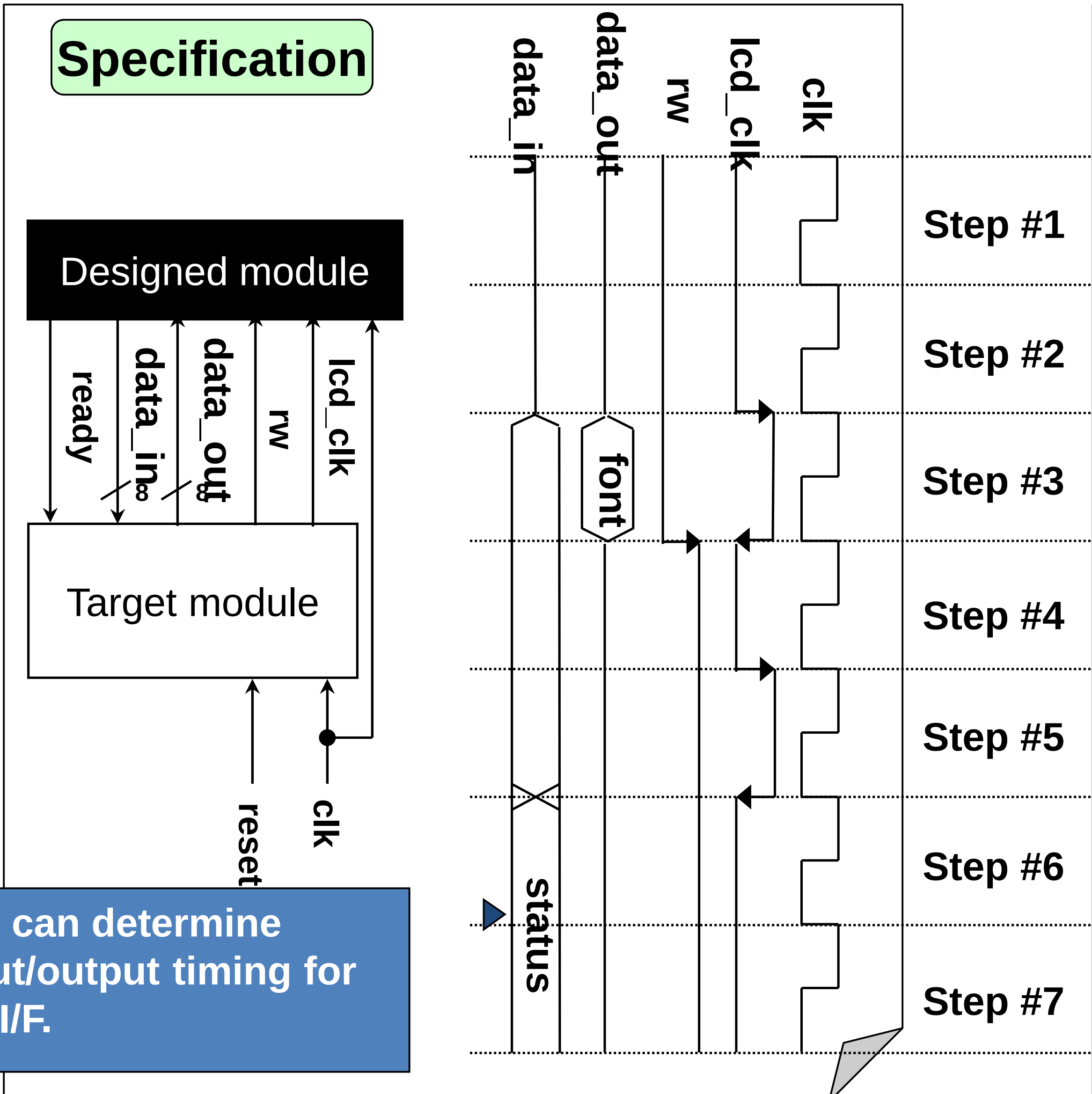
Sequencers

-USB I/F, ATA, UART,...
-PCI bus I/F, AMBA bus.
-DMA, TIMER,…
-SDRAM I/F, NAND flash,...

- Multiple internal synthesis Engines
- Multiple Synthesis directives (local and global)
- C syntax extension (implementation description)

Manual Scheduling - LCD Controller Example

Specification



You can determine input/output timing for the I/F.

C for HW

```
lcd_clk = 0 ;
$
lcd_clk = 1 ;
rw = 0 ;
data_out = font ;
$
lcd_clk = 0 ;
rw = 1 ;
$
do{
   lcd_clk = 1 ;
$
   lcd_clk = 0 ;
$
   status = data_in ;
$
} while ( status = = 1 )
```

You can control the timing for all signals with the symbol ‘$’.

Scheduling blocks

- Scheduling_block enables the use of timed descriptions inside an untimed description.
  - Clock boundary for a clock specifier “$”
  - No extra clock boundaries

```
process main(){
        ....
  /* Cyber scheduling_block */
  for(i=0; i<16; i++) {
    ary1[i] = in0 + i;
    $
    ary2[i] = in1 + i;
    $
  }
        ....
}
```

Specify clock boundaries for timing-critical loops

```
process main(){
  /* Cyber scheduling_block */
  {
    while(!en) $;
    data = data_in;
    ack = 1;
  }

  /* body of algorithm */
        ....
}
```

Describe interfaces in an algorithmic description.

CWB – Testbench Generator

- CWB comes with an automatic testbench generator
- The testbench reads untimed inputs/outputs from a text file with extension .clv or .tlv
- clv vs. tlv files
  - Clv: Cycle-level vectors
  - Tlv : Transaction-level vectors

CLV/TLV files

- Every IO has to have a .clv or .tlv file
  - <io_name>.clv .e.g.
  - in ter(7..0) a; → a.clv or a.tlv
- In case that IO is an array. E.g.
  - In ter(0:16) data[8];
  - data_a00.clv, data_a001.clv, etc..

a.clv./tlv

```
3
4
5
1
14
65
34
```

BDL Example

```
int fifo[8]  = {0, 0, 0, 0, 0, 0, 0, 0};




int ave8(int in0){
 int, sum,  i, out0;

 /* Shift data to accommodate new input to be read */
      for (i = 7; i > 0; i--) {
          fifo[i] = fifo[i- 1];
       }

  /* Read new input into fifo */
    fifo[0] = in0;

  /* Set first element of sum to compute the average => can save 1 loop iteration*/
    sum= fifo[0];

   /* Add up all the numbers in the fifo */
     for (i= 1; i< 8; i++) {
          sum += fifo[i];
       }

 /* Compute the avg by dividing by 8 -> In HW  a divide by 8 (/8) = shift 3 times */
       out= sum / 8;

  /* Output the newly computed average to the output port */
    return (out0);
}
```

```
int fifo[8]  = {0, 0, 0, 0, 0, 0, 0, 0};

#ifdef BDL
            in ter(0:8) in0_bdl;
            out ter(7..0) out0_bdl;
            process ave8(){
#else
           int ave8(int in0){
#endif
int, sum,  i, out0;

#ifdef BDL
    in0 = in0_bdl;
#endif

 /* Shift data to accommodate new input to be read */
      for (i = 7; i > 0; i--) {
          fifo[i] = fifo[i- 1];
       }

:
:
:

 /*
Output the newly computed average to the output port */
#ifdef BDL
 out0_bdl = out0;
#else
 return (out0);
#end if;
}
```

Synthesis directives (pragmas) Example

```
in  ter(0:8)  in0;
out ter(0:8)  out0;
var(0:8)  fifo[8] /* Cyber array = REG */= {0, 0, 0, 0, 0, 0, 0, 0};
process ave(){
int  out0_v, sum,  i;
/* Cyber unroll_times =all */
 for (i = 7; i > 0; i--) {
      fifo[i] = fifo[i- 1];
}
  fifo[0] = in0;
  sum= fifo[0];
/* Cyber unroll_times =0 */
   for (i= 1; i< 8; i++) {
      sum += fifo[i];  }
    out0_v= sum / 8;
    out0 = out0_v;}
```

Synthesis directives:

- Arrays : Registers or Memory
- Loops: Unroll or not

HLS example – Average of 8 numbers

```
in  ter(0:8)  in0;
out ter(0:8)  out0;
var(0:8)  fifo[8] = {0, 0, 0, 0, 0, 0, 0, 0};
process ave8(){
int  out0_v, sum,  i;
  for (i = 7; i > 0; i--) {
            fifo[i] = fifo[i- 1];
}
    fifo[0] = in0;
    sum= fifo[0];
      for (i= 1; i< 8; i++) {
            sum += fifo[i];
        }
        out0_v= sum / 8;
        out0 = out0_v;}
```

THE HONG KONG
POLYTECHNIC UNIVERSITY
香港理工大學

HLS Example – Resource Allocation

- What type and how many number of FUs are needed to fully unroll the loops ?

```
@FCNT{
            NAME      add8u
            LIMIT     4
#           COMMENT
}
@FCNT{
            NAME      add12u
            LIMIT     3
#           COMMENT
}
```

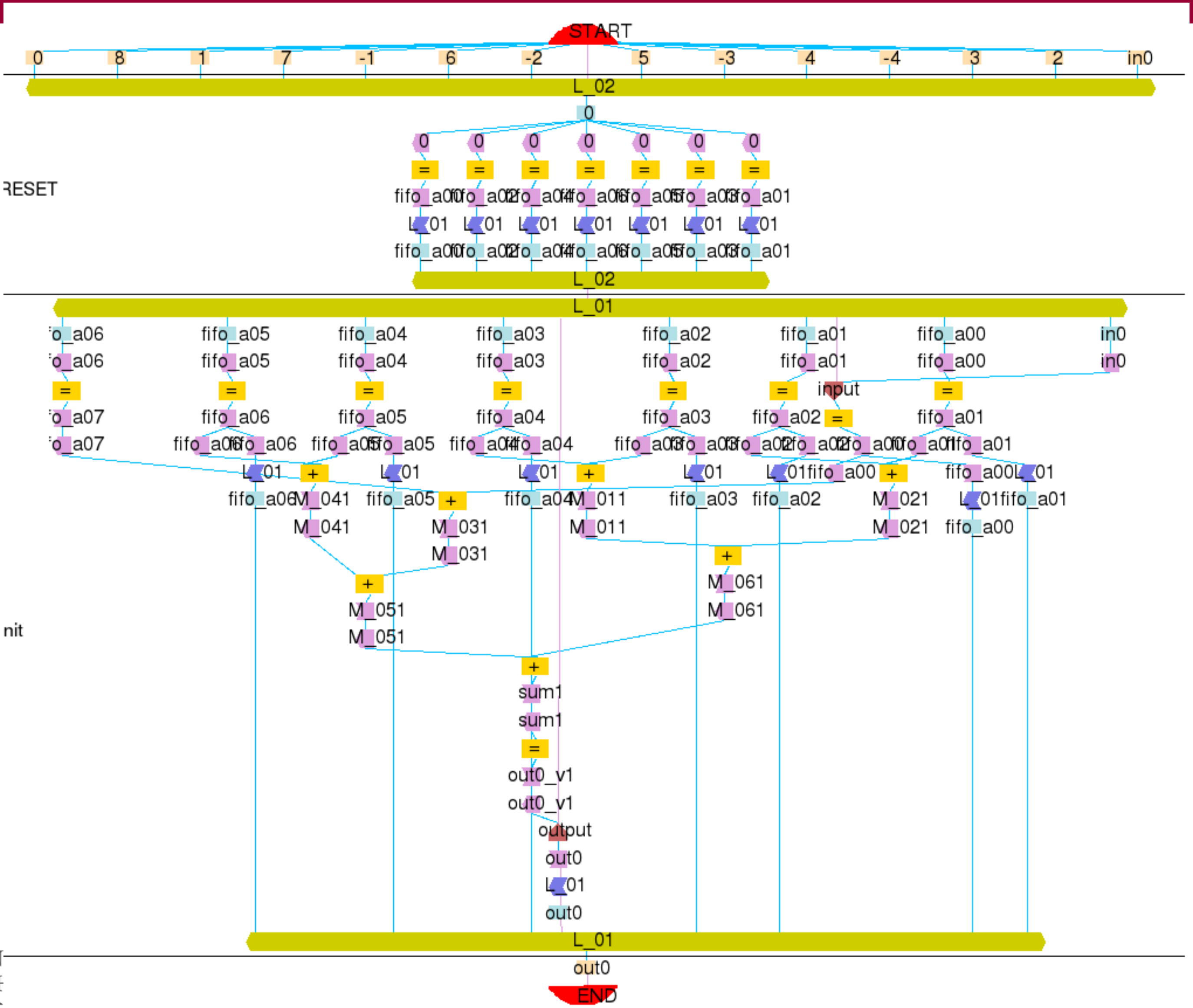
HLS Example - Scheduling

- Schedule the code manually given:
  - the following constraint and delay files
  - Target frequency of 100 MHz (delay = 10ns) 1000 x 1/100ns [unit]

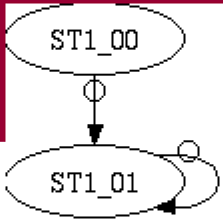
```
@FCNT{
          NAME     add8u
          LIMIT    4
}
@FCNT{
          NAME     add12u
          LIMIT    3
}
```

```
DELAY_UNIT 1/100ns
@FLIB{
          NAME     add8u
          DELAY    52
          AREA     312
}
@FCNT{
          NAME     add12u
          DELAY     61
          AREA       507
}
```

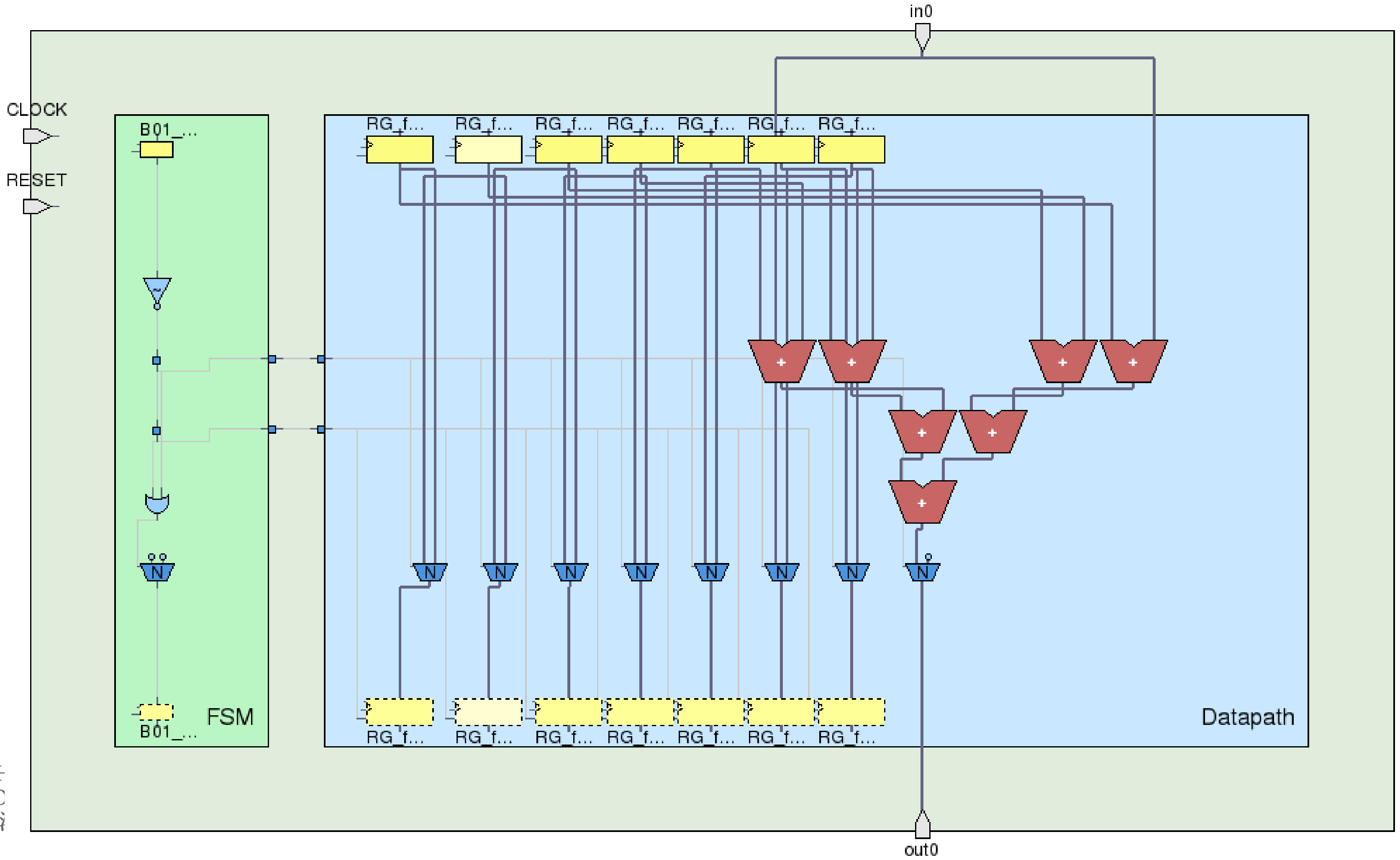
HLS - Scheduling



HLS Example- Binding

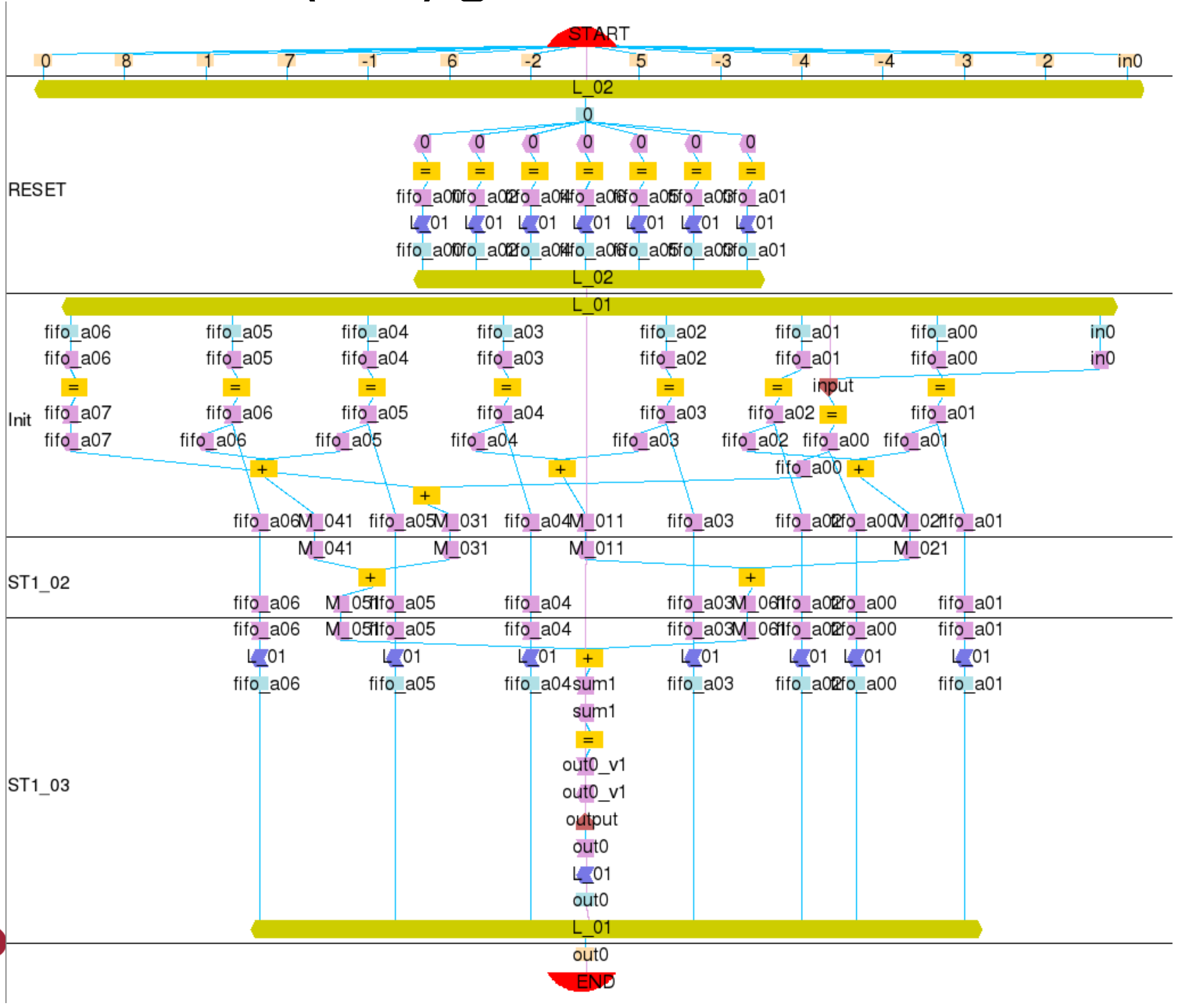


- Bind the scheduled CDFG and report the total number of states of the FSM controller

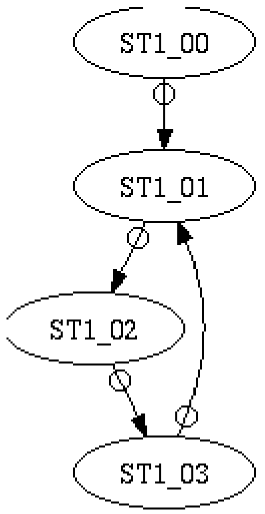


HLS - Example

- What would happen if the target frequency increased to 1 GHz (1ns) given the same constraints ?

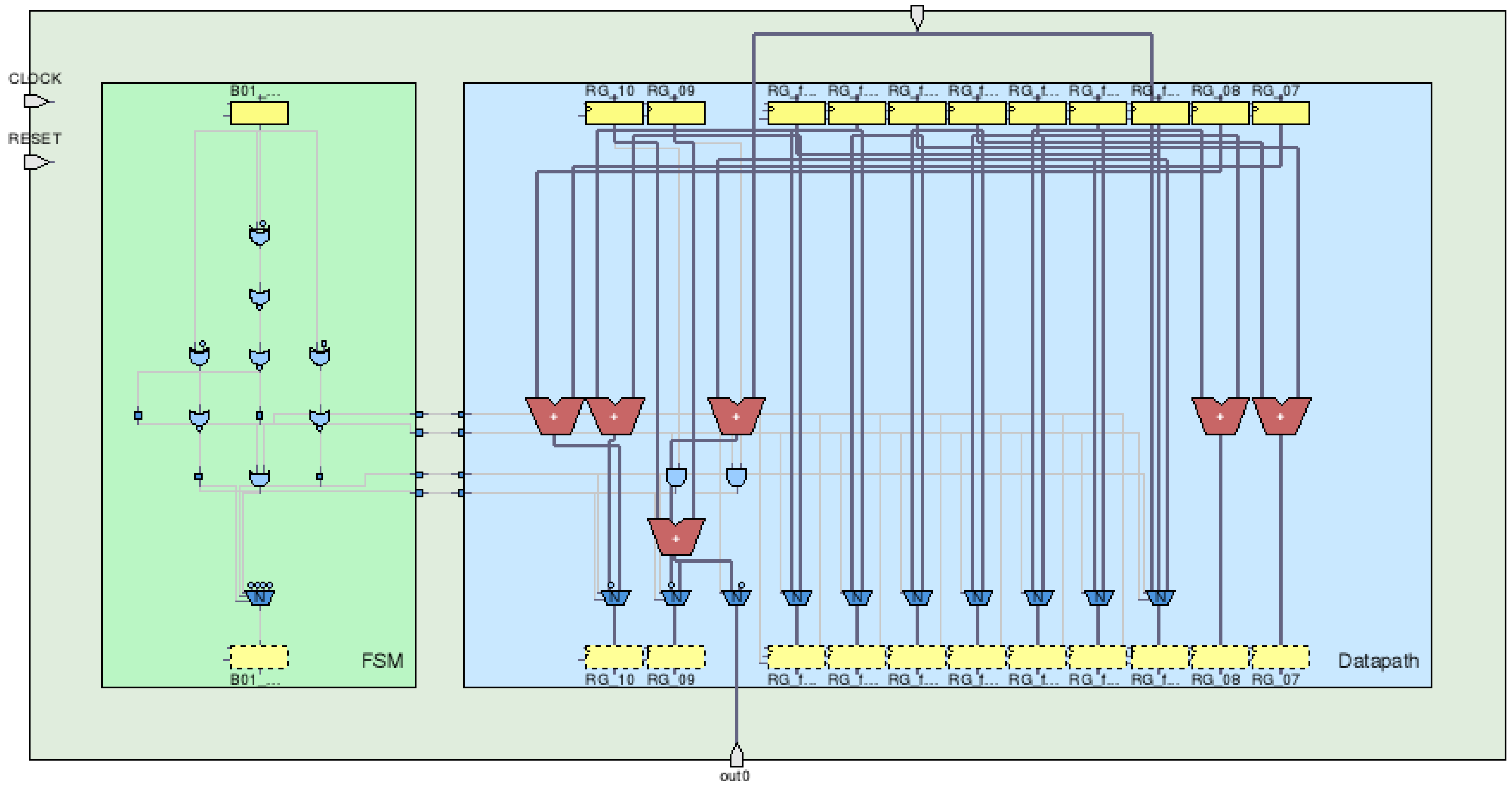


- FUs operations cannot be scheduled together in 1 clock cycle ➔ Need 3 cycles (states to execute)



HLS - Binding

- FSM needed to steer data through 3 states
- Less FUs needed because cannot execute all operations on the same stage



Commercial HLS Tool

- Cadence – Stratus (C,C++, SystemC)
- Mentor Graphics – Catapult (C++, SystemC)
- NEC – CyberWorkBench (C, SystemC)
- Synopsys – Synphony (C, SystemC)
- Xilinx – VivadoHLS (C,C++, SystemC)

HLS Adoption Problems

- Input language
  - ANSI C? (subsets), SystemC?
- New design methodology. Needs time to be adopted ➔ Still not being taught at each school
- Current RTL designers need to adopt it, but they don't 'trust' the tool
- QoR compared to hand-coded RTL

Input Language - ANSI-C

Pros:
  - Large user base
  - Lots of legacy code
  - Easy to learn
- Cons:
  - Does not have HW specific constructs. E.g. custom bitwidth data types, parallel statements ➔ each vendor developed its own subset
  - ANSI-C subsets bind users to the EDA vendor. High switching cost
  - ANSI-C subsets cannot be compiled with ‘gcc’ (need a dedicated compiler)

SystemC

- Open Source C++ class library for HW description
- IEEE standard (IEEE 1666)
- Freely available at www.accellera.org
- Simulation is fast
- Can develop and co-simulate HW and SW
- Promote interoperability between tools
- Detail SystemC information available at: http://videos.accellera.org/tlm20andsubset/index.html

Why was SystemC needed?

- In ANSI C/C++ you can not:
  - **Express Concurrency**: HW system operate by nature in parallel
  - **Custom Data types**: e.g. specify any bitwidth, fixed-point data types,
  - **Model communications**: Signals, protocols
  - Notion of time: Clock cycles

S2CBench : Synthesizable SystemC Benchmark Suite for High-Level Synthesis

- 12+1 SystemC Benchmarks which comply with latest SystemC synthesizable subset draft (12 synthesizable+1 non-synthesizable)
- Freely available at:
  - www.s2cbench.org
  - http://sourceforge.net/projects/s2cbench/
- From different domains
  - Automotive, Security, Telecommunication, Consumer
- Control dominated and Data dominant designs
- Each test unique HLS features
  1. Tool language support (e.g. templates, structures, fixed point data types)
  2. Synthesis optimizations (e.g. loop unrolling, pipelining, function inlining, array synthesis)
  3. Tool performance (e.g. running time, accuracy of synthesis report)

Motivation for S2CBench

- HLS tool evaluation cycle is typically very long (multiple tools are evaluated using multiple designs)
  - Companies don't have the expertise in HLS
  - Companies don't have ANSI-C, C++ or SystemC models for their RTL designs in order to compare the QoR of the HLS tools
- C/C++ supported by most vendors include vendor-specific constructs (e.g. data types, port declarations)
- SystemC only language supported by all major HLS vendors

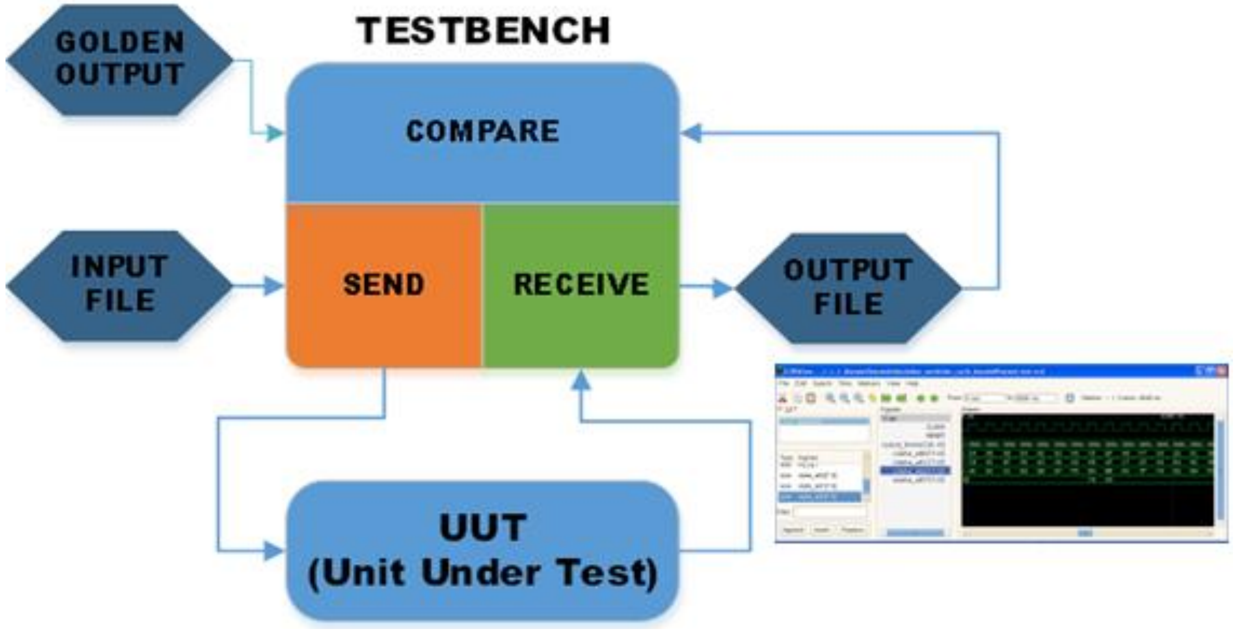
Vendor	Tool Name	Supported Languages
Cadence (Forte)	Cynthesizer	SystemC
Cadence	C-to-Silicon	C, C++, SystemC
Calypto	CatapultC	C++, SystemC
NEC	CyberWorkBench	C, SystemC
Xilinx	Vivado HLS	C, C++, SystemC

S2CBench : 15+1 designs

Design	Type	Domain	Optimizations Tested
qsort	dd	Auto/Ind	Loops, arrays, functions pointers
sobel	dd	Auto/Ind	Loops, functions, IO array expansion, multi-dimensional arrays expansion, fixed arrays (ROM, logic)
aes cipher	dd	Security	IO array expansion, multi-dimensional arrays expansion , large fixed arrays
kasumi	dd	Security	Multi-processes, delay report accuracy
md5c	dd	Security	#define macros, delay report accuracy
snow3G	dd	Security	Templates, delay report accuracy, function synthesis
adpcm	Cd	Telecom	Structure synthesis
FFT	dd	Telecom	Floating point, trigonometric functions
FIR	dd	Consu	IO array expansion, arrays, loops, functions, sum of products
Decimation	dd	Consum	Resource sharing across loops, fixed point data types
Interp	dd	Consum	Polynomial decomposition, fixed point data types, sum or products
IDCT	dd	Consum	#include statement to initialize arrays, loops, functions,
Disparity	cd/dd	Consum	Hierarchical design, multi-dimensional array expansion, synthesis running time

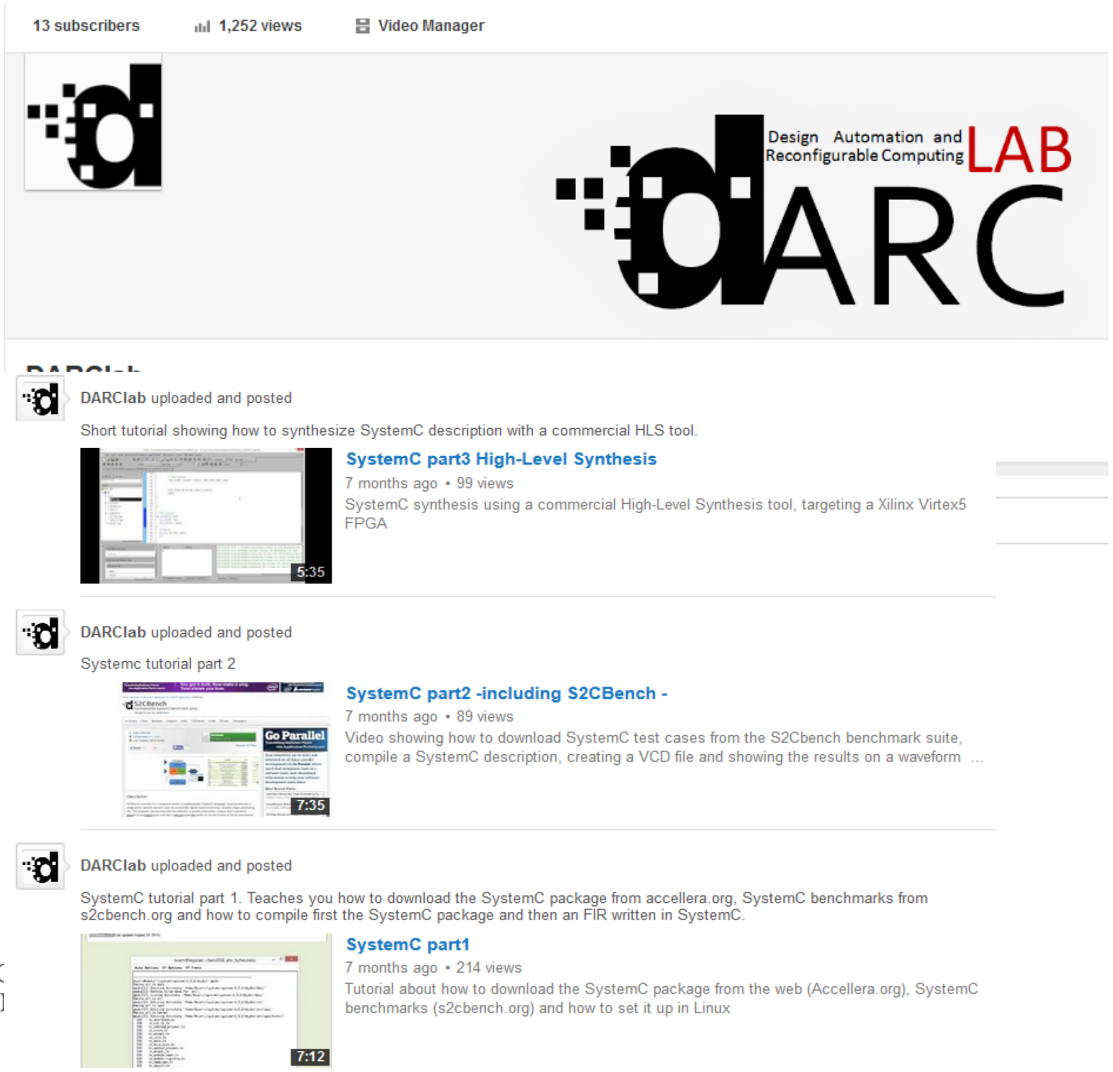
Benchmark Block Diagram

- TB sends stimuli data stored in files (editable) to UUT
- TB receives the data and compares it against golden output (stored in file)
- TB reports if results match or not
- Option to dump VCD file



HLS YouTube Channel

- https://www.youtube.com/user/DARClabify



13 subscribers 1,252 views Video Manager

DARClab uploaded and posted

Short tutorial showing how to synthesize SystemC description with a commercial HLS tool.

SystemC part3 High-Level Synthesis

7 months ago • 99 views

SystemC synthesis using a commercial High-Level Synthesis tool, targeting a Xilinx Virtex5 FPGA

5:35

DARClab uploaded and posted

Systemc tutorial part 2

SystemC part2 -including S2CBench -

7 months ago • 89 views

Video showing how to download SystemC test cases from the S2Cbench benchmark suite, compile a SystemC description, creating a VCD file and showing the results on a waveform ...

7:35

DARClab uploaded and posted

SystemC tutorial part 1. Teaches you how to download the SystemC package from accellera.org, SystemC benchmarks from s2cbench.org and how to compile first the SystemC package and then an FIR written in SystemC.

SystemC part1

7 months ago • 214 views

Tutorial about how to download the SystemC package from the web (Accellera.org), SystemC benchmarks (s2cbench.org) and how to set it up in Linux

7:12

THE HONG KONG POLYTECHNIC U
香港理工大學

What Do I want from you

- New ideas for my research → the more challenging and “crazy” the better
- Internships and possible “jobs” for my students
- Financial support for a common project (university could match the funds)

Next Steps

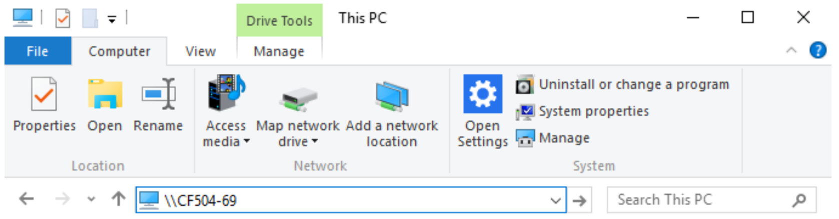
- First step:
  - Re-design some of the available RTL in C/SystemC
  - Compare results
  - Create new designs from C
- Second step:
  - Possible financial support to hire a student for a joint project(s)

For more information :

b.carrionschafer@polyu.edu.hk
www.eie.polyu.edu.hk\~schaferb

Tutorial

1. Log in to your computer booting in Windows7
   – Login: CF105-<num>
   – Password : CF105-<num>
2. Copy the files in the shared folder to your machine
   – In file explorer \\CF504-69



3. Open CyberWorkBench

THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

FAQ1) "How Do You Compare With Hand-written RTL?"

- Fixed Architecture (e.g. Processor):

Configurable processor	C	RTL	Ratio
#lines	1.3KL	9.2KL	7.6X
Sim. speed	61Kc/s	0.3K cycles/sec	203X
Size	19KG	18KG	5%+

- HW design from sequential algorithm (resource sharing)

Viterbi decoder	C	RTL	Ratio
Size	5 KG	50KG	1/10

Constraint length: 9, Data rate: 1/2, word 8bit, ACB 4 parallel

Decode: Cycles 10,882 clocks , speed 0.68ms (at 16MHz clock)

THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學