The Fundamental Circuits Design for P Systems Implementation

Zeyi Shang

Southwest Jiaotong University

Université Paris Est Créteil Val de Marne

Motivation of Hardware Implementation

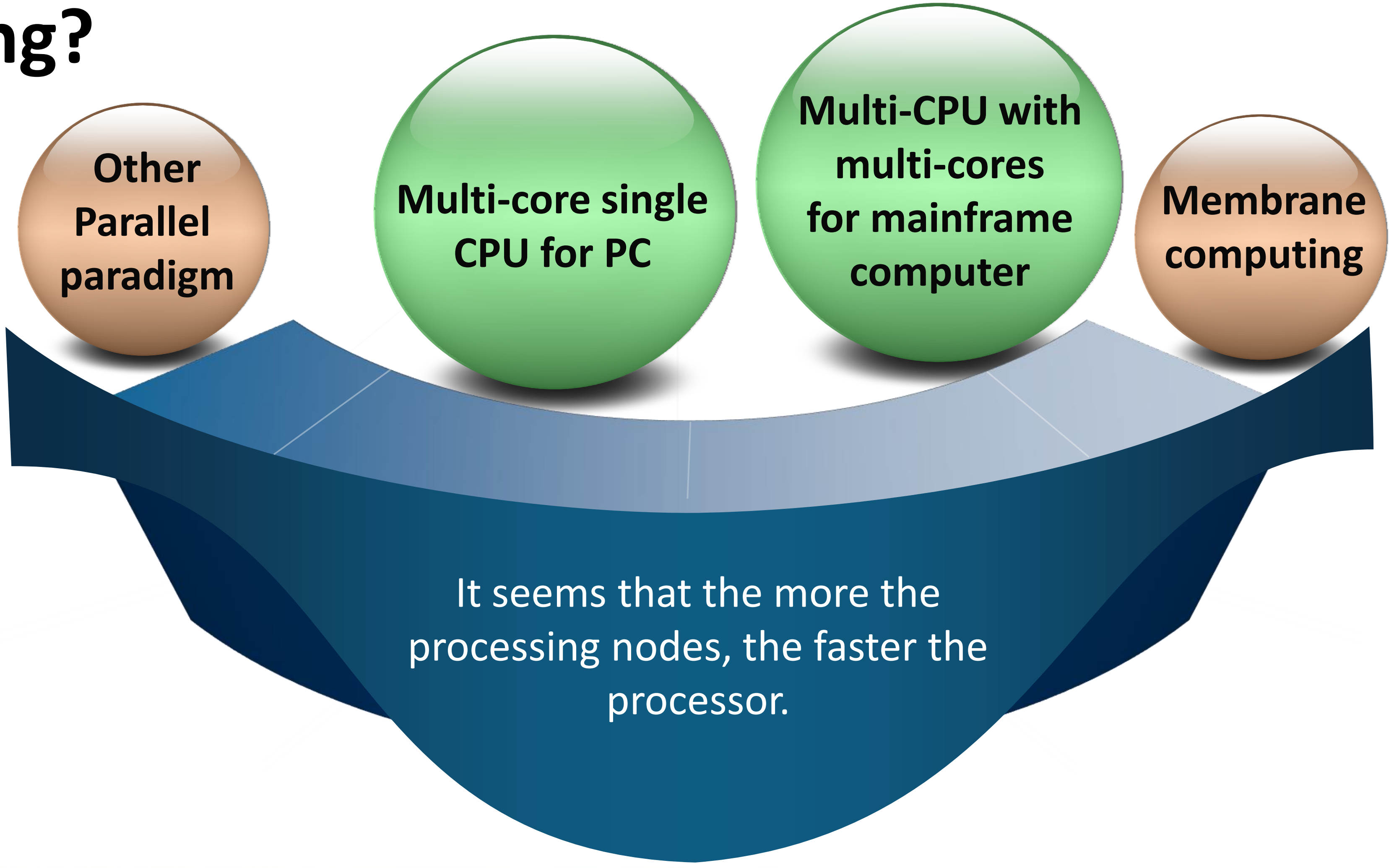
Why we need hardware implementation of P systems?

- Membrane computing is a newly parallel processing paradigm, which can be truly obtained on a parallel hardware, not just software simulation.

How to implement P systems on hardware?

- Design the parallel architectures and functional modules within it that conform to P systems' attributes in re-programmable hardware devices.

What are the differences between P system paradigm and the multi-core/CPU parallel processing?

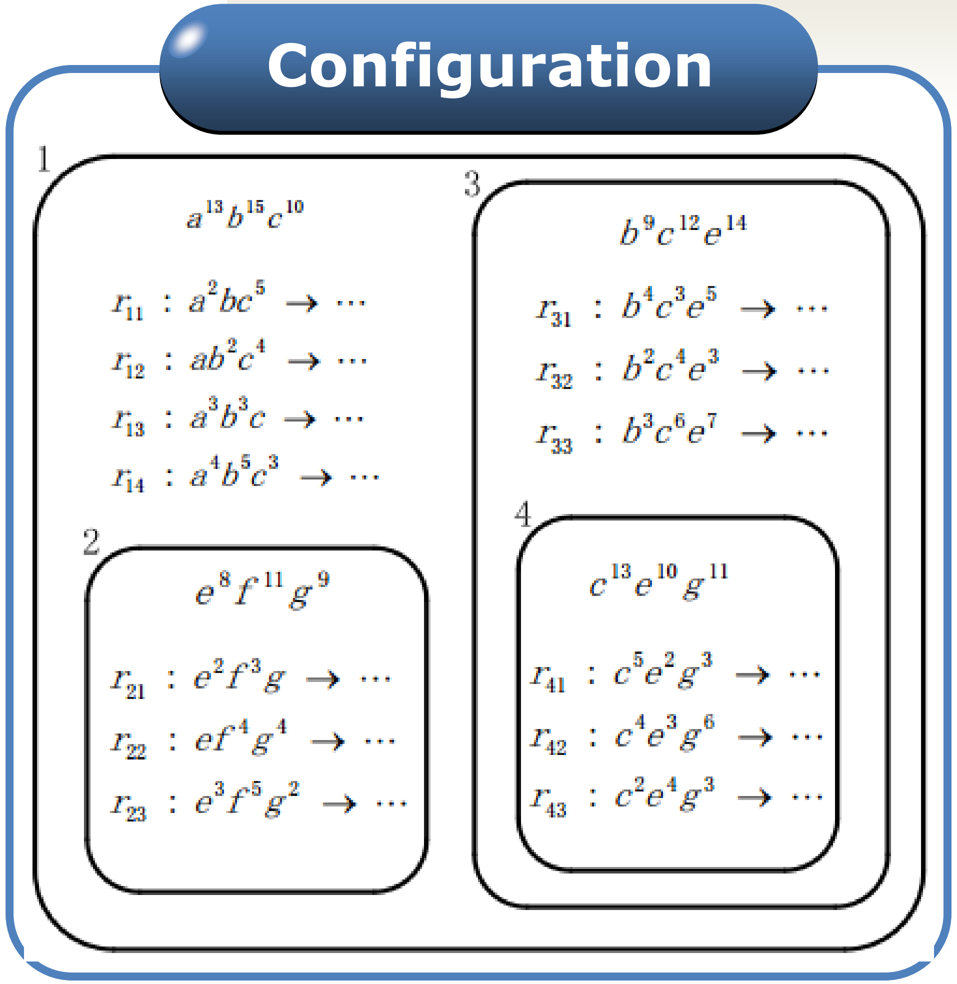

Other Parallel paradigm
Multi-core single CPU for PC
Multi-CPU with multi-cores for mainframe computer
Membrane computing

It seems that the more the processing nodes, the faster the processor.

The Arithmetic Operations in the Processing of P systems

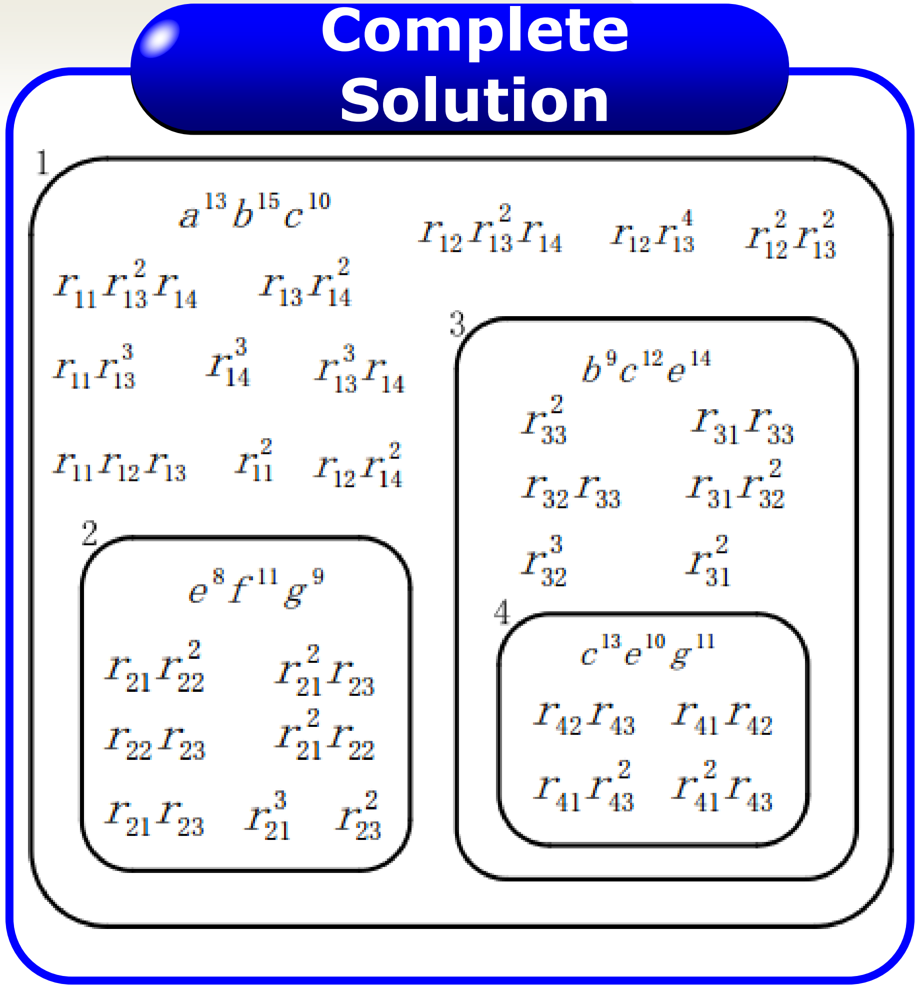
1. Division
2. Multiplication
3. Addition/ Subtraction

The analysis of parallelism and non-determinism


Configuration
1
$a^{13}b^{15}c^{10}$
$r_{11}: a^2bc^5 \to \cdots$
$r_{12}: ab^2c^4 \to \cdots$
$r_{13}: a^3b^3c \to \cdots$
$r_{14}: a^4b^5c^3 \to \cdots$
2
$e^8f^{11}g^9$
$r_{21}: e^2f^3g \to \cdots$
$r_{22}: ef^4g^4 \to \cdots$
$r_{23}: e^3f^5g^2 \to \cdots$
3
$b^9c^{12}e^{14}$
$r_{31}: b^4c^3e^5 \to \cdots$
$r_{32}: b^2c^4e^3 \to \cdots$
$r_{33}: b^3c^6e^7 \to \cdots$
4
$c^{13}e^{10}g^{11}$
$r_{41}: c^5e^2g^3 \to \cdots$
$r_{42}: c^4e^3g^6 \to \cdots$
$r_{43}: c^2e^4g^3 \to \cdots$

Derivation Mode:

Maximal Parallel



Division

Take a cell-like symbol objects P system evolving in maximal parallelism mode as an example. Calculating the maximal times of application (instances) of each rule in every region is the first computing step. This process comprises a serial of divisions and a logic MIN operation.

For each object type in a region, a division whose dividend is the number of that object type in the multiset of objects and the divisor is the number of objects in a given rule's left-hand-side is performed. The quotient of a division is the integer multiple (times) of the amount of that type in the multiset of objects comparing to the quantity of that object in the definition of the considered rule.

MIN(quotient1, quotient2, quotient3, …) gives the maximal instances of the rule.

Multiplication

After each maximal instance of rule is obtained, we should produce a multiset of rules with respect to the maximal parallelism evolving mode. The range of each number of the rules in the multiset of rules is 0 to the maximal instance. Assume that we elaborate a algorithm to generate the wanted multiset of rules and a multiset of rules selected to evolve the region configuration is $r_1^a r_2^b r_3^c \cdots$.

To compute the quantity of objects consumed and produced by the multiset of rules, we should perform a set of multiplications. The exponent of each object type in a definition of rule should multiply the exponent a, b, c, $\cdots$.

The product of objects in LHS is the number of objects consumed and the product in the RHS is the amount generated.

Addition/Subtraction

The next step carries out the update of multiset of objects which store in array of registers in terms of the products gained in the multiplications.

The LHS products should be subtracted from the relevant registers, whereas the RHS products should be added to the respective counterparts. Once all the additions and subtractions are executed, the multisets of objects are updated.

The arithmetic operations are indispensable for the implementation of membrane computing. If these operations are performed in parallel in hardware, we have a parallel processing device then.

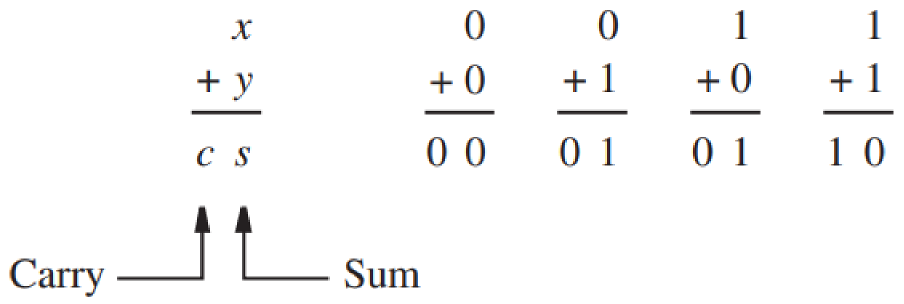
As we can anticipate, it is a common phenomenon that more than one rule update the same register, which cause conflicts. Appropriate strategies are needed to cope with this kind of conflict.

Example: Adder

Different with the software modeling, when we program with Hardware Description Language (HDL) such as Verilog or VHDL, we do not just program some algorithms but design digital circuits directly or indirectly. Mastering the grammar of the HDL is just the fundamental tool, the digital circuits design knowledge play the kernel role in elaborating a sophisticated circuits.

If we want to perform addition, we should design an adder first. An adder can be designed from different view of points. From a more intuitive way, a ripple-carry adder is easy to design, shown in next figure. The drawback of this kind of adder is that the delay is great. This fact will impair the performance of the adder when the bits number grow. We need a more efficient method to build a fast adder.

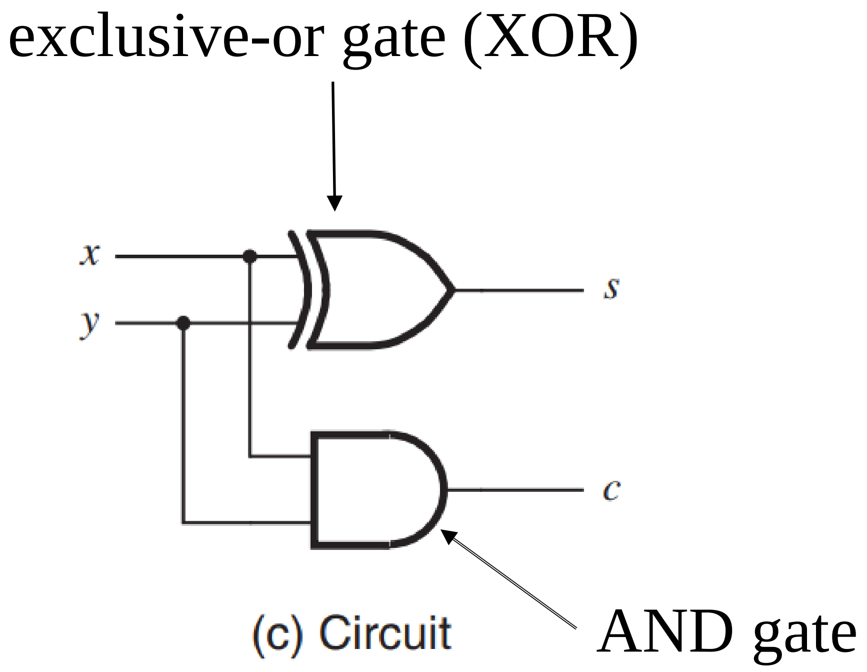
Half Adder



(a) The four possible cases

		Carry	Sum
x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(b) Truth table



(c) Circuit

There are no carry signals from other bit in the half adder because the total bit involved is only 1.

Ripple-carry adder

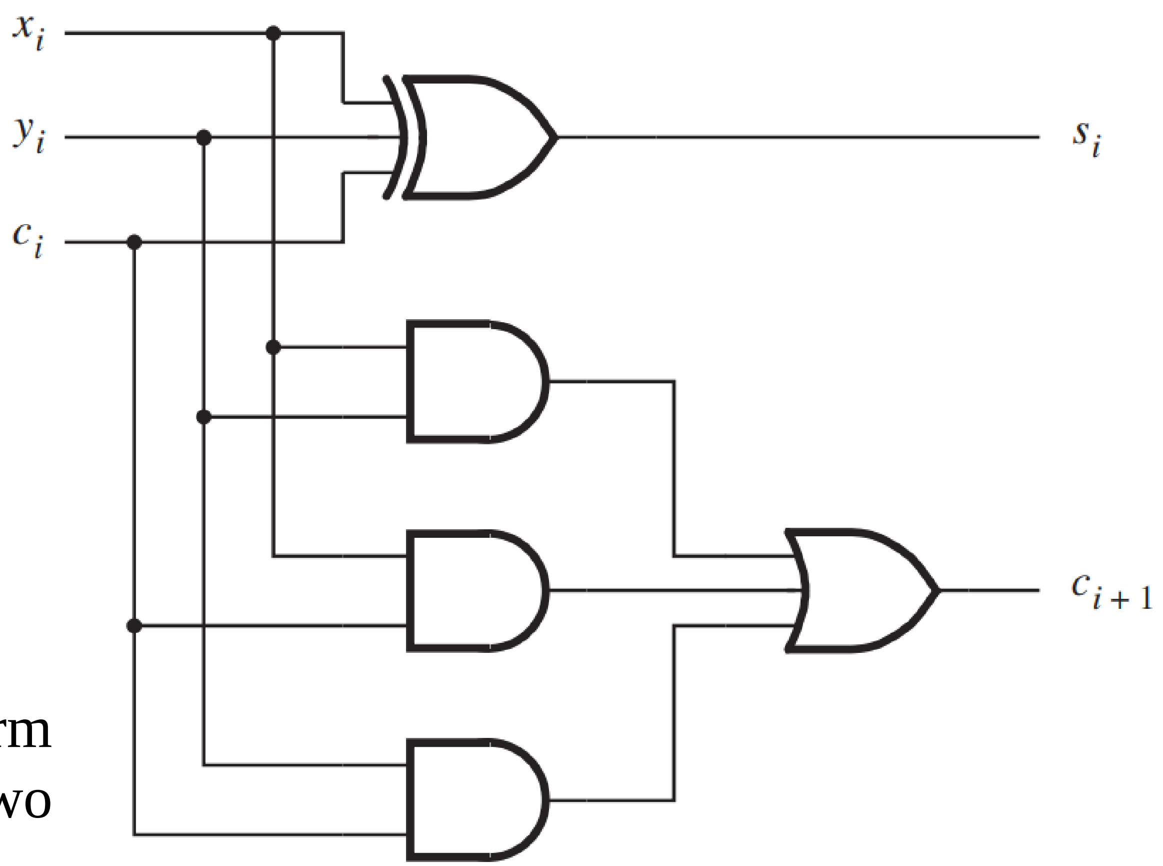
Generated carries $\longrightarrow$	1 1 1 0	
$X = x_4x_3x_2x_1x_0$	0 1 1 1 1	$(15)_{10}$
$+Y = y_4y_3y_2y_1y_0$	+ 0 1 0 1 0	$+ (10)_{10}$
$S = s_4s_3s_2s_1s_0$	1 1 0 0 1	$(25)_{10}$

(a) An example of addition

…	c_{i+1}	c_i	…
…	…	x_i	…
…	…	y_i	…
…	…	s_i	…

(b) Bit position i

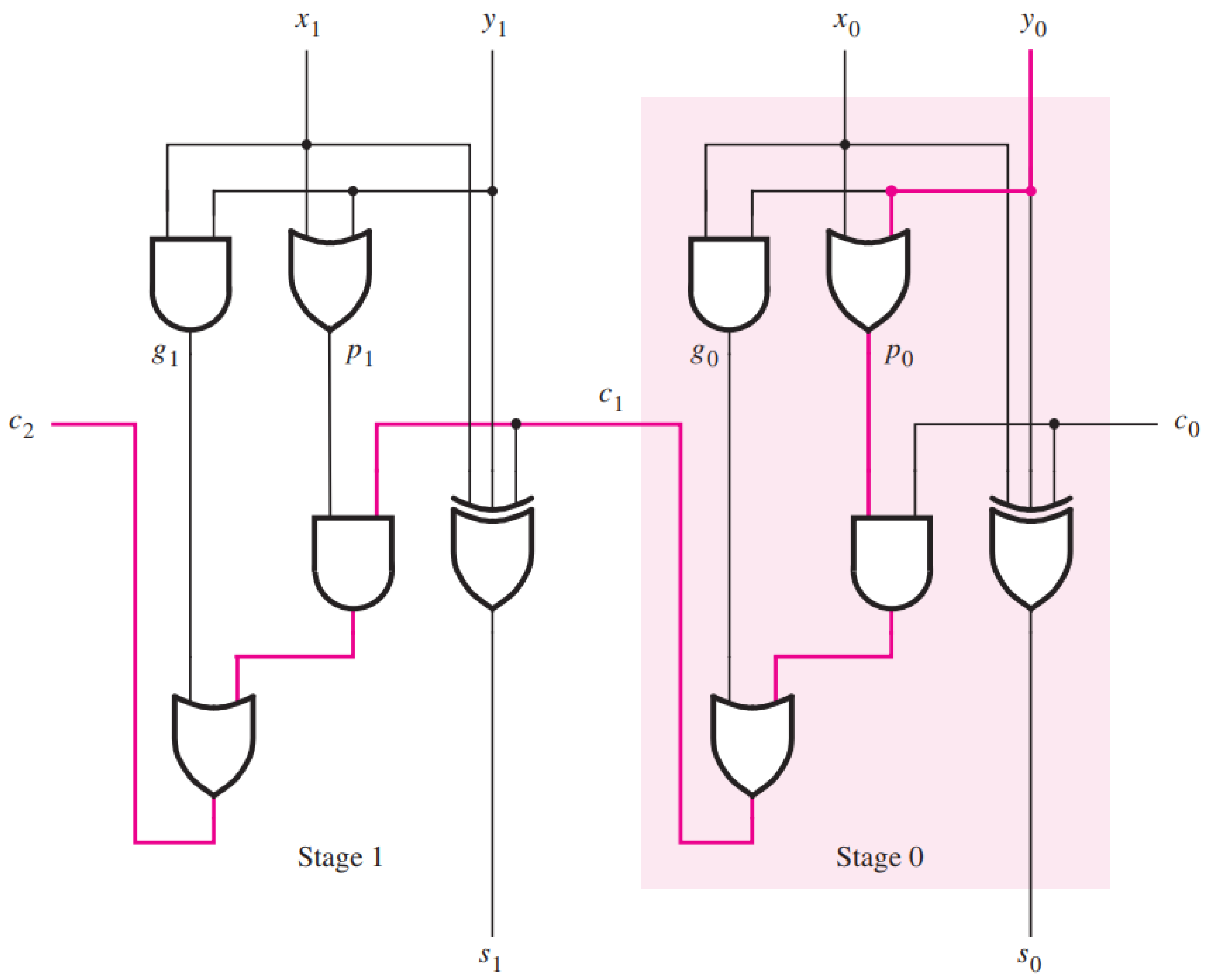
A full adder perform 1-bit addition of two multi-bit binary numbers. The total bits are larger than 1.



(c) Circuit

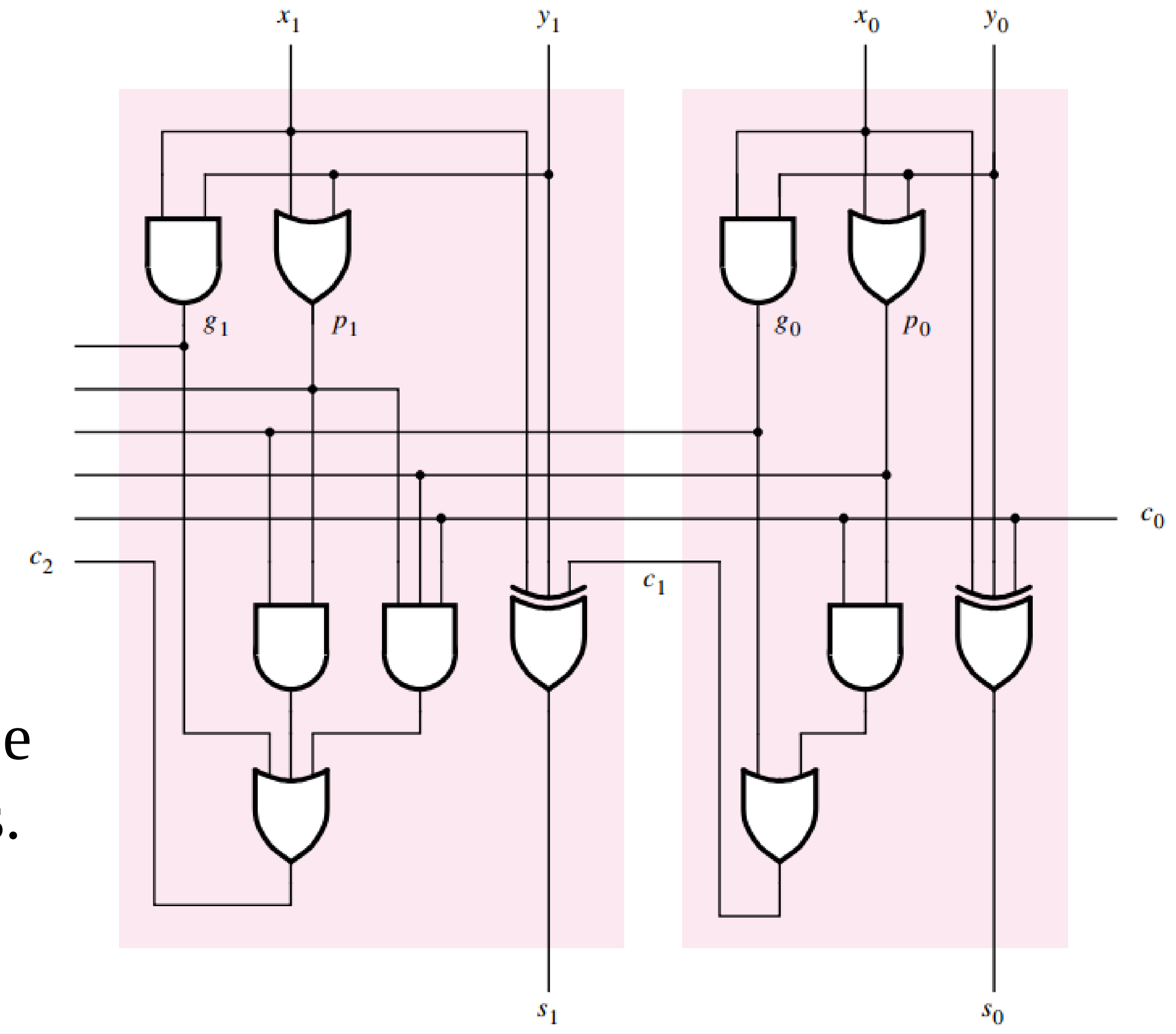
Ripple-carry adder

The right figure show the first 2 bit of a ripple-carry adder. The slow speed of the ripple-carry adder is caused by the long path along which a carry signal must propagate. The total number of gate delays along the critical path is 2n + 1. n is the bit number.



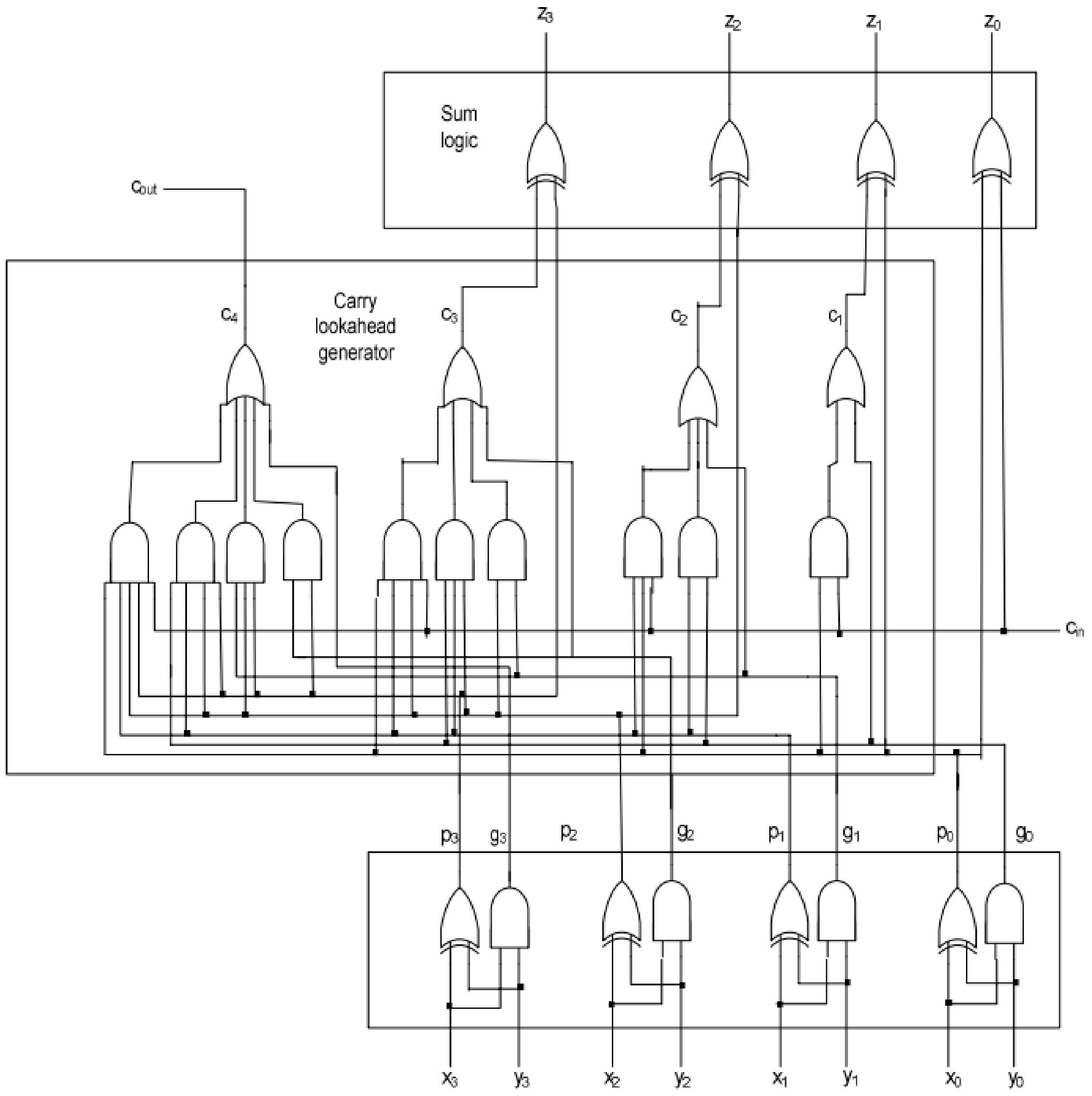
Carry-lookahead adder

For carry-lookahead adder, all carry signals are produced after three gate delays: one gate delay is needed to produce the generate and propagate signals *g*0, *g*1, *p*0, and *p*1, and two more gate delays are needed to produce *c*1 and *c*2 concurrently. Extending the circuit to n bits, the final carry-out signal cn would also be produced after only three gate delays.



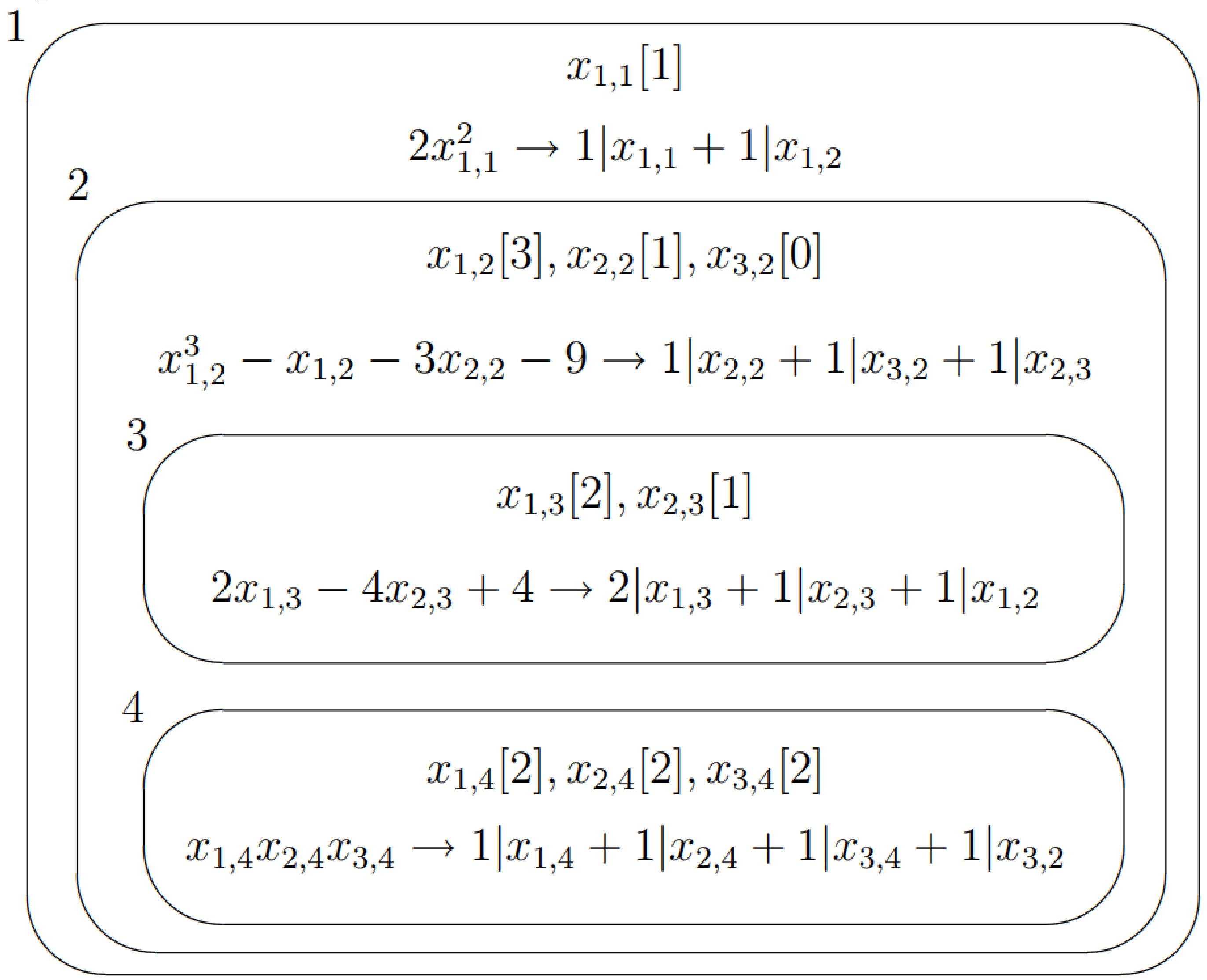
4-bit carry-lookahead adder

With the bit number rises, the complexity of carry-lookahead adder circuit increase dramatically. The fast computing speed comes at the price of complicated wire interconnection.

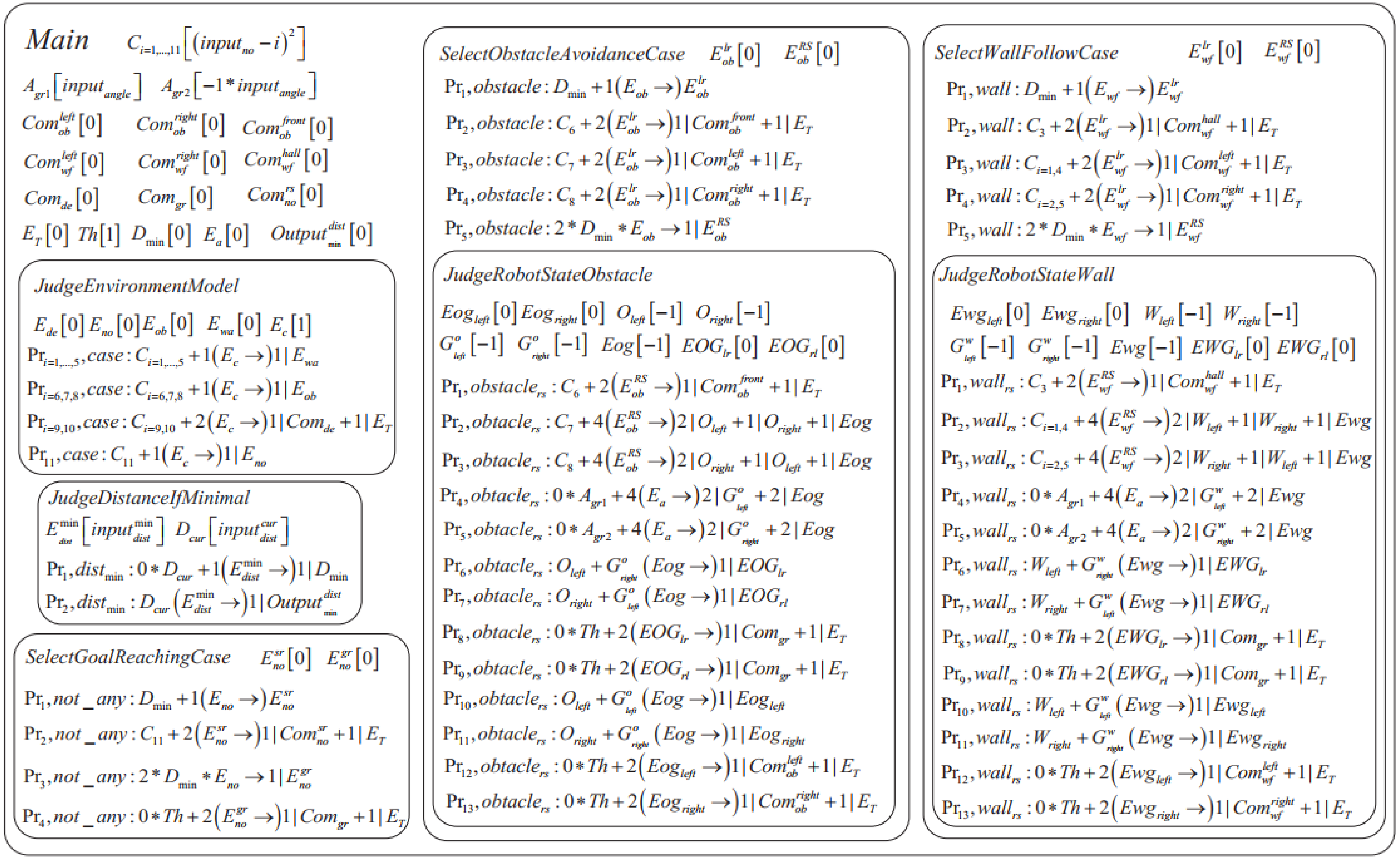


Numerical P system application

Numerical P systems (NPS) is the only one model so far that have been put into real-life applications, in the autonomous mobile robot control. The right figure shows a example of NPS. We can see that the all processes are algebraic operations. The core work is still handling arithmetic operations.


1
$x_{1,1}[1]$
$2x_{1,1}^2 \rightarrow 1|x_{1,1} + 1|x_{1,2}$
2
$x_{1,2}[3], x_{2,2}[1], x_{3,2}[0]$
$x_{1,2}^3 - x_{1,2} - 3x_{2,2} - 9 \rightarrow 1|x_{2,2} + 1|x_{3,2} + 1|x_{2,3}$
3
$x_{1,3}[2], x_{2,3}[1]$
$2x_{1,3} - 4x_{2,3} + 4 \rightarrow 2|x_{1,3} + 1|x_{2,3} + 1|x_{1,2}$
4
$x_{1,4}[2], x_{2,4}[2], x_{3,4}[2]$
$x_{1,4}x_{2,4}x_{3,4} \rightarrow 1|x_{1,4} + 1|x_{2,4} + 1|x_{3,4} + 1|x_{3,2}$

A NPS Controller



$Main$ $C_{i=1,\dots,11}\left[(input_{no}-i)^2\right]$

$A_{gr1}\left[input_{angle}\right]$ $A_{gr2}\left[-1*input_{angle}\right]$

$Com_{ob}^{left}[0]$ $Com_{ob}^{right}[0]$ $Com_{ob}^{front}[0]$

$Com_{wf}^{left}[0]$ $Com_{wf}^{right}[0]$ $Com_{wf}^{hall}[0]$

$Com_{de}[0]$ $Com_{gr}[0]$ $Com_{no}^{rs}[0]$

$E_T[0]$ $Th[1]$ $D_{\min}[0]$ $E_a[0]$ $Output_{\min}^{dist}[0]$

$JudgeEnvironmentModel$

$E_{de}[0]$ $E_{no}[0]$ $E_{ob}[0]$ $E_{wa}[0]$ $E_c[1]$

$\Pr_{i=1,\dots,5}, case: C_{i=1,\dots,5}+1(E_c\to)1|E_{wa}$

$\Pr_{i=6,7,8}, case: C_{i=6,7,8}+1(E_c\to)1|E_{ob}$

$\Pr_{i=9,10}, case: C_{i=9,10}+2(E_c\to)1|Com_{de}+1|E_T$

$\Pr_{11}, case: C_{11}+1(E_c\to)1|E_{no}$

$JudgeDistanceIfMinimal$

$E_{dist}^{\min}\left[input_{dist}^{\min}\right]$ $D_{cur}\left[input_{dist}^{cur}\right]$

$\Pr_1, dist_{\min}: 0*D_{cur}+1(E_{dist}^{\min}\to)1|D_{\min}$

$\Pr_2, dist_{\min}: D_{cur}(E_{dist}^{\min}\to)1|Output_{\min}^{dist}$

$SelectGoalReachingCase$ $E_{no}^{sr}[0]$ $E_{no}^{gr}[0]$

$\Pr_1, not\_any: D_{\min}+1(E_{no}\to)E_{no}^{sr}$

$\Pr_2, not\_any: C_{11}+2(E_{no}^{sr}\to)1|Com_{no}^{sr}+1|E_T$

$\Pr_3, not\_any: 2*D_{\min}*E_{no}\to 1|E_{no}^{gr}$

$\Pr_4, not\_any: 0*Th+2(E_{no}^{gr}\to)1|Com_{gr}+1|E_T$

$SelectObstacleAvoidanceCase$ $E_{ob}^{lr}[0]$ $E_{ob}^{RS}[0]$

$\Pr_1, obstacle: D_{\min}+1(E_{ob}\to)E_{ob}^{lr}$

$\Pr_2, obstacle: C_6+2(E_{ob}^{lr}\to)1|Com_{ob}^{front}+1|E_T$

$\Pr_3, obstacle: C_7+2(E_{ob}^{lr}\to)1|Com_{ob}^{left}+1|E_T$

$\Pr_4, obstacle: C_8+2(E_{ob}^{lr}\to)1|Com_{ob}^{right}+1|E_T$

$\Pr_5, obstacle: 2*D_{\min}*E_{ob}\to 1|E_{ob}^{RS}$

$JudgeRobotStateObstacle$

$Eog_{left}[0]$ $Eog_{right}[0]$ $O_{left}[-1]$ $O_{right}[-1]$

$G_{left}^{o}[-1]$ $G_{right}^{o}[-1]$ $Eog[-1]$ $EOG_{lr}[0]$ $EOG_{rl}[0]$

$\Pr_1, obstacle_{rs}: C_6+2(E_{ob}^{RS}\to)1|Com_{ob}^{front}+1|E_T$

$\Pr_2, obtacle_{rs}: C_7+4(E_{ob}^{RS}\to)2|O_{left}+1|O_{right}+1|Eog$

$\Pr_3, obtacle_{rs}: C_8+4(E_{ob}^{RS}\to)2|O_{right}+1|O_{left}+1|Eog$

$\Pr_4, obtacle_{rs}: 0*A_{gr1}+4(E_a\to)2|G_{left}^{o}+2|Eog$

$\Pr_5, obtacle_{rs}: 0*A_{gr2}+4(E_a\to)2|G_{right}^{o}+2|Eog$

$\Pr_6, obtacle_{rs}: O_{left}+G_{right}^{o}(Eog\to)1|EOG_{lr}$

$\Pr_7, obtacle_{rs}: O_{right}+G_{left}^{o}(Eog\to)1|EOG_{rl}$

$\Pr_8, obtacle_{rs}: 0*Th+2(EOG_{lr}\to)1|Com_{gr}+1|E_T$

$\Pr_9, obtacle_{rs}: 0*Th+2(EOG_{rl}\to)1|Com_{gr}+1|E_T$

$\Pr_{10}, obtacle_{rs}: O_{left}+G_{left}^{o}(Eog\to)1|Eog_{left}$

$\Pr_{11}, obtacle_{rs}: O_{right}+G_{right}^{o}(Eog\to)1|Eog_{right}$

$\Pr_{12}, obtacle_{rs}: 0*Th+2(Eog_{left}\to)1|Com_{ob}^{left}+1|E_T$

$\Pr_{13}, obtacle_{rs}: 0*Th+2(Eog_{right}\to)1|Com_{ob}^{right}+1|E_T$

$SelectWallFollowCase$ $E_{wf}^{lr}[0]$ $E_{wf}^{RS}[0]$

$\Pr_1, wall: D_{\min}+1(E_{wf}\to)E_{wf}^{lr}$

$\Pr_2, wall: C_3+2(E_{wf}^{lr}\to)1|Com_{wf}^{hall}+1|E_T$

$\Pr_3, wall: C_{i=1,4}+2(E_{wf}^{lr}\to)1|Com_{wf}^{left}+1|E_T$

$\Pr_4, wall: C_{i=2,5}+2(E_{wf}^{lr}\to)1|Com_{wf}^{right}+1|E_T$

$\Pr_5, wall: 2*D_{\min}*E_{wf}\to 1|E_{wf}^{RS}$

$JudgeRobotStateWall$

$Ewg_{left}[0]$ $Ewg_{right}[0]$ $W_{left}[-1]$ $W_{right}[-1]$

$G_{left}^{w}[-1]$ $G_{right}^{w}[-1]$ $Ewg[-1]$ $EWG_{lr}[0]$ $EWG_{rl}[0]$

$\Pr_1, wall_{rs}: C_3+2(E_{wf}^{RS}\to)1|Com_{wf}^{hall}+1|E_T$

$\Pr_2, wall_{rs}: C_{i=1,4}+4(E_{wf}^{RS}\to)2|W_{left}+1|W_{right}+1|Ewg$

$\Pr_3, wall_{rs}: C_{i=2,5}+4(E_{wf}^{RS}\to)2|W_{right}+1|W_{left}+1|Ewg$

$\Pr_4, wall_{rs}: 0*A_{gr1}+4(E_a\to)2|G_{left}^{w}+2|Ewg$

$\Pr_5, wall_{rs}: 0*A_{gr2}+4(E_a\to)2|G_{right}^{w}+2|Ewg$

$\Pr_6, wall_{rs}: W_{left}+G_{right}^{w}(Ewg\to)1|EWG_{lr}$

$\Pr_7, wall_{rs}: W_{right}+G_{left}^{w}(Ewg\to)1|EWG_{rl}$

$\Pr_8, wall_{rs}: 0*Th+2(EWG_{lr}\to)1|Com_{gr}+1|E_T$

$\Pr_9, wall_{rs}: 0*Th+2(EWG_{rl}\to)1|Com_{gr}+1|E_T$

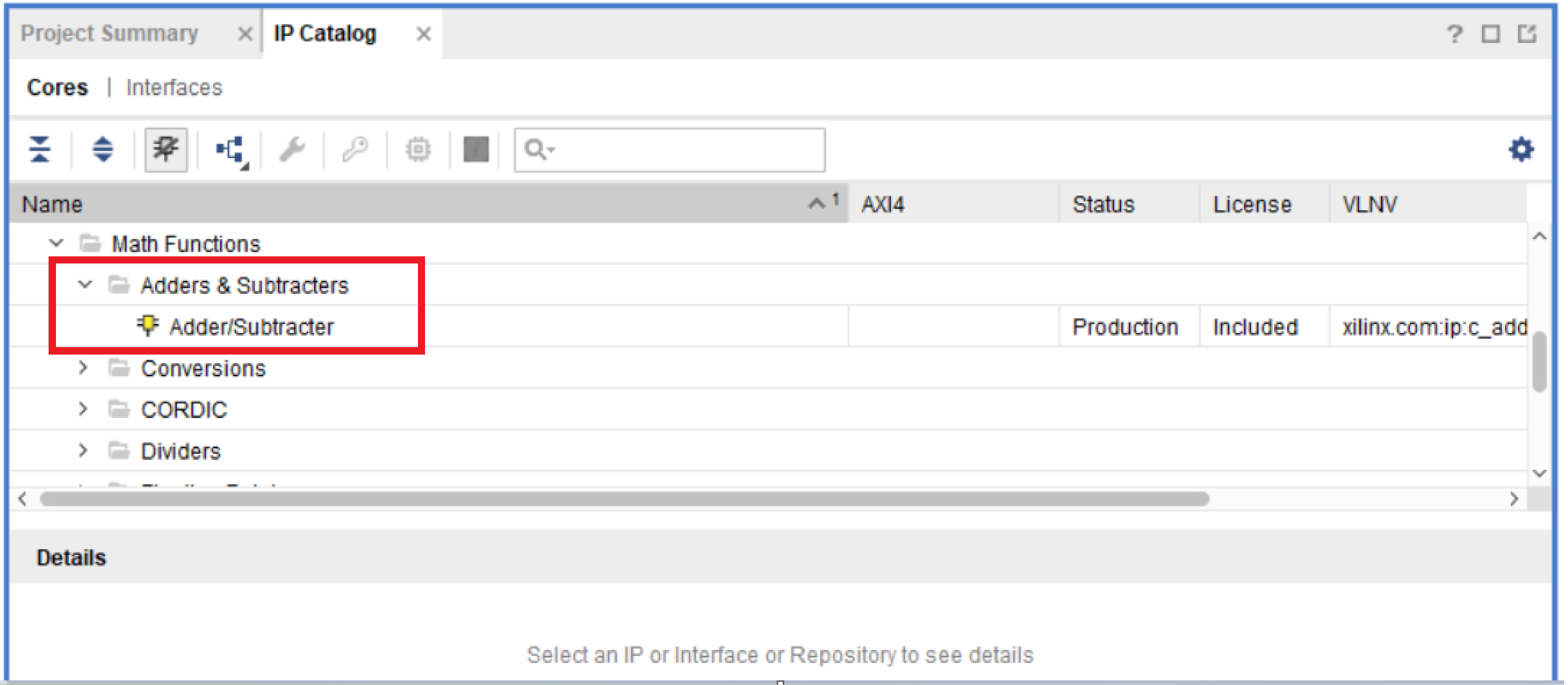
$\Pr_{10}, wall_{rs}: W_{left}+G_{left}^{w}(Ewg\to)1|Ewg_{left}$

$\Pr_{11}, wall_{rs}: W_{right}+G_{right}^{w}(Ewg\to)1|Ewg_{right}$

$\Pr_{12}, wall_{rs}: 0*Th+2(Ewg_{left}\to)1|Com_{wf}^{left}+1|E_T$

$\Pr_{13}, wall_{rs}: 0*Th+2(Ewg_{right}\to)1|Com_{wf}^{right}+1|E_T$

Problem: how to obtain the Vivado learning materials besides the User Guide?


Project Summary | IP Catalog

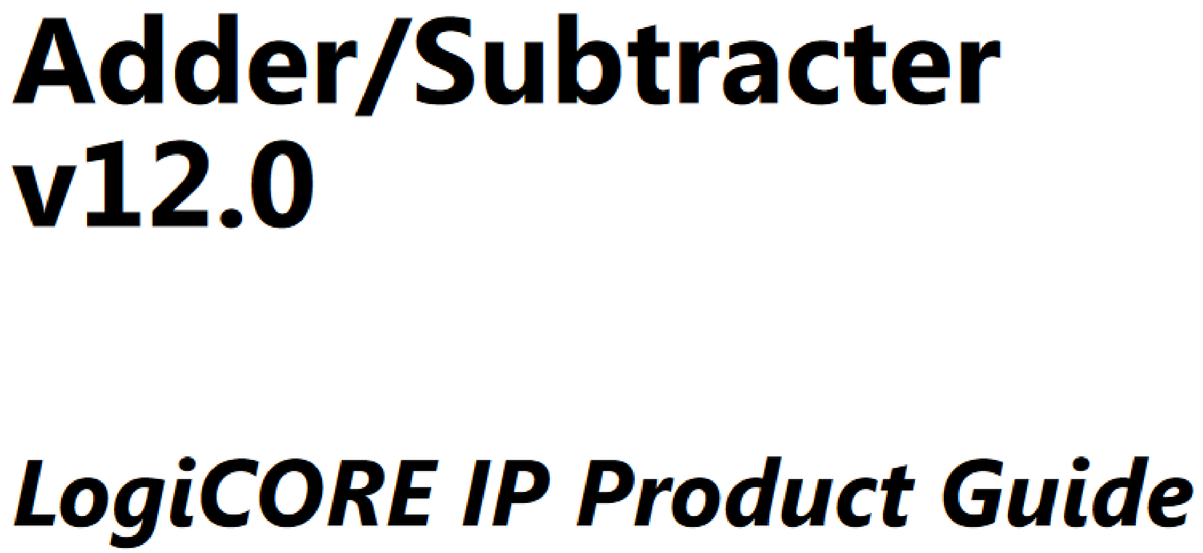
Cores | Interfaces

Name	AXI4	Status	License	VLNV
Math Functions				
Adders & Subtracters				
Adder/Subtracter		Production	Included	xilinx.com:ip:c_add
Conversions				
CORDIC				
Dividers				

Details

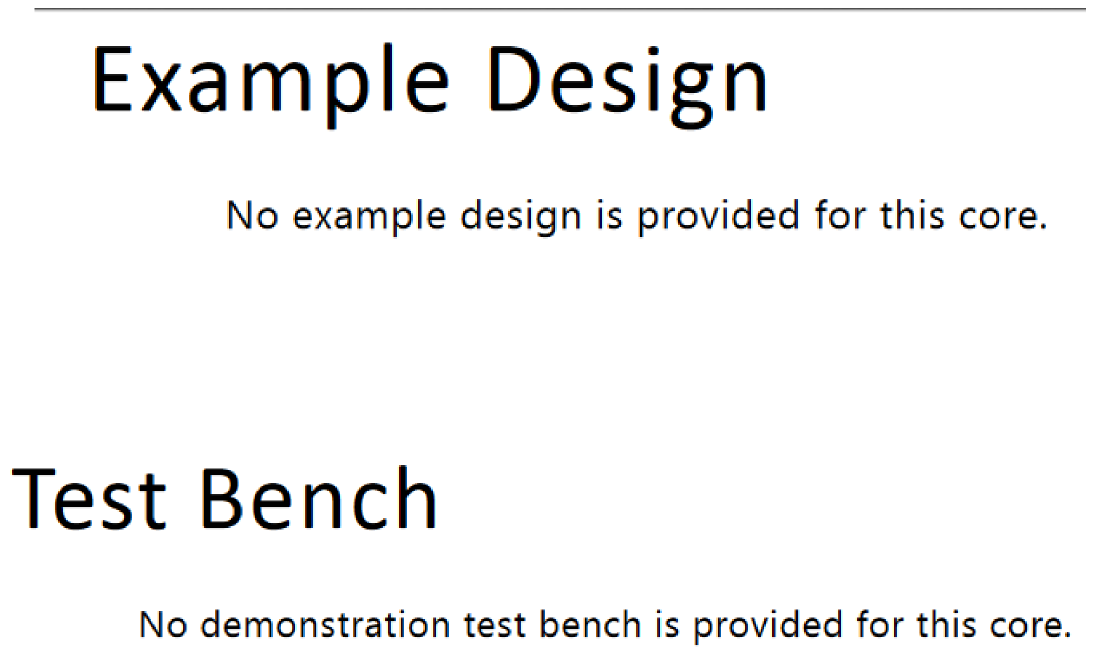
Select an IP or Interface or Repository to see details

Help me: how to obtain the Vivado learning materials besides the User Guide?



Adder/Subtracter v12.0

LogiCORE IP Product Guide



Example Design

No example design is provided for this core.

Test Bench

No demonstration test bench is provided for this core.

Thank You !