Languages and Tools for Formal Design of Cyber-Physical Systems

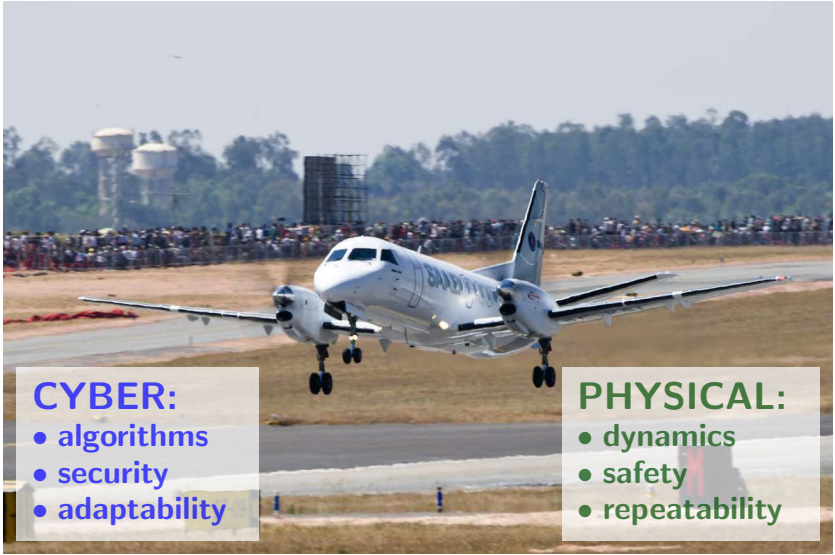
George Ungureanu[1] *Timmy Sundström*[2] *Ingo Sander*[1] *Ingemar Söderquist*[2]

[1]School of EECS, KTH Royal Institute of Technology, Stockholm, Sweden
[2]Business Area Aeronautics, Saab AB, Linköping, Sweden

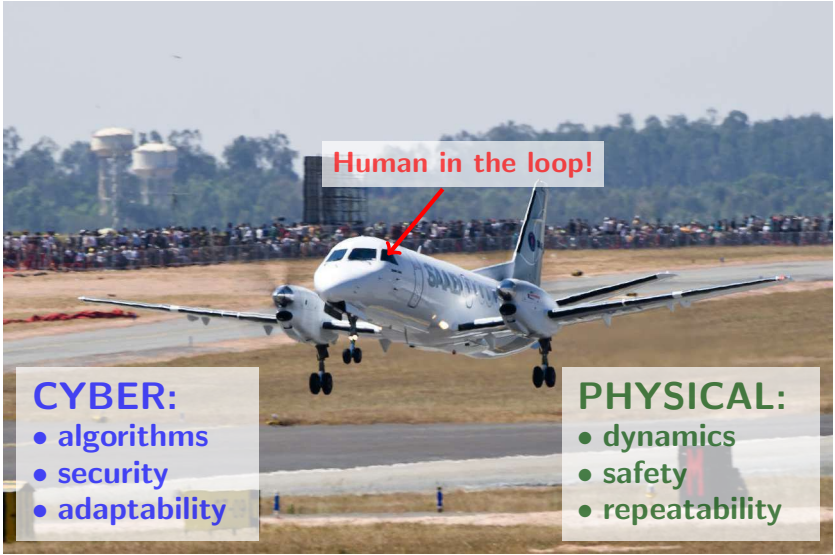
October 8, 2019

Cyber-Physical Systems

Cyber-Physical Systems

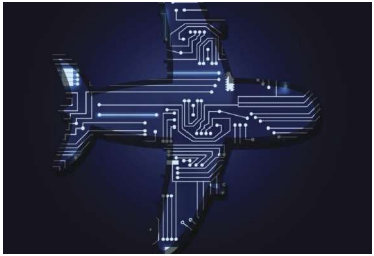


Cyber-Physical Systems



Cyber Physical System Design

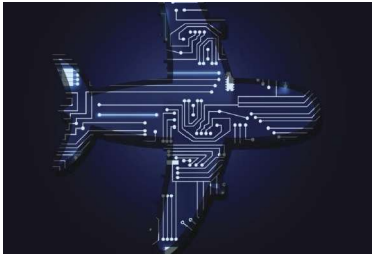
- combine many *mature* disciplines
- tools to design and simulate behaviors
- design flows from specification to implementation



$\Downarrow$

Cyber Physical System Design ... Challenges

- combine many *mature* disciplines
  - *incompatible* abstractions
  - *fundamental* issues with interpretations of *time*
- tools to design and simulate behaviors

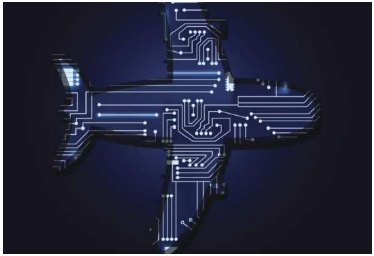


$\Downarrow$

- design flows from specification to implementation

Cyber Physical System Design ... Challenges

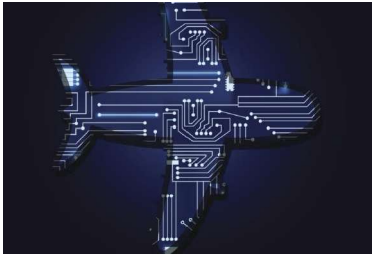
- combine many *mature* disciplines
  - *incompatible* abstractions
  - *fundamental* issues with interpretations of *time*
- tools to design and simulate behaviors
  - text-based specification documents
  - no formal semantics $\Rightarrow$ low correlation between simulation model and design artifact
- design flows from specification to implementation



$\Downarrow$

Cyber Physical System Design ... Challenges

- combine many *mature* disciplines
  - *incompatible* abstractions
  - *fundamental* issues with interpretations of *time*
- tools to design and simulate behaviors
  - text-based specification documents
  - no formal semantics $\Rightarrow$ low correlation between simulation model and design artifact
- design flows from specification to implementation
  - refinements are *ad-hoc*, based on designer experience
  - heavily dependent on prototype testing across design stages

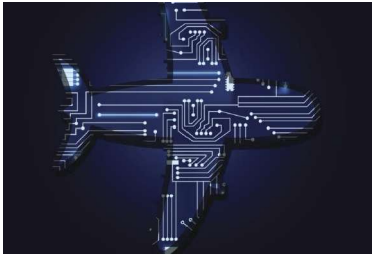


$\Downarrow$

Cyber Physical System Design ... Challenges

- combine many *mature* disciplines
  - *incompatible* abstractions
  - *fundamental* issues with interpretations of *time*
- tools to design and simulate behaviors
  - text-based specification documents
  - no formal semantics $\Rightarrow$ low correlation between simulation model and design artifact
- design flows from specification to implementation
  - refinements are *ad-hoc*, based on designer experience
  - heavily dependent on prototype testing across design stages

HUGE COSTS!



$\Downarrow$

Just Implement It...

```c
#include <msp430.h>

int main(void) {
    WDTCTL = WDTPW | WDTHOLD;   // Stop watchdog timer

    //Configure output ports
    P1DIR |= BIT0;              // P1.0 = LED

    //Configure clock system
    UCSCTL0 = 0x0000;           // Set lowest possible DCOx, MODx
                                // These are controlled by FLL
    UCSCTL1 = DCORSEL_5;        // Select DCO range to 16MHz
    UCSCTL2 = 249;              // (N+1) * 32.768kHz = 16MHz
    UCSCTL3 = SELREF_2;         // Set DCO FLL reference = REFO
    UCSCTL4 = SELA_0 | SELS_4 | SELM_3 ;   // Set ACLK = XT1,
                                // SMCLK = DCOCLKDIV, MCLK = DCOCLK
    __bis_status_register(SCG0 | SCG1 | OSCOFF);   // Disable FLL, XT1 osc
    __nop();

    //Configure timer A0
    TA0CTL = TASSEL_1 | MC_1 | TACLR;   // SMCLK = 32768Hz, Up Mode
    TA0CCR0 = 32767;                    // 1 second period

    while(1){
        P1OUT ^= BIT0;

        while((TA0CTL & TAIFG) == 0){
            }
            TA0CTL &= ~(TAIFG);
    }

    return 0;
}
```

7M lines of code[1]

[1] Boeing 787 Dreamliners *avionics* software. Source `https://informationisbeautiful.net`

Just Implement It...

```c
#include <msp430.h>

int main(void) {
    WDTCTL = WDTPW | WDTHOLD;   // Stop watchdog timer

    //Configure output ports
    P1DIR |= BIT0;              // P1.0 = LED

    //Configure clock system
    UCSCTL0 = 0x0000;           // Set lowest possible DCOx, MODx
                                // These are controlled by FLL
    UCSCTL1 = DCORSEL_5;        // Select DCO range to 16MHz
    UCSCTL2 |= 243;             // (N+1) * 32.768kHz = 16MHz
    UCSCTL3 = SELREF_2;         // Set DCO FLL reference = REFO
    UCSCTL4 = SELA_0 | SELS_4 | SELM_3 ;   // Set ACLK = XT1,
                                // SMCLK = DCOCLKDIV, MCLK = DCOCLK
    __bis_status_register(SCG0 | SMCLKOFF | FLLUNLOCK_0 | OSCOFF);  // Disable FLL,
    //Loop until XT1,XT2 & DCO stabilizes - in this case only DCO has to stabilize
    do{
        UCSCTL7 &= ~(XT2OFFG | XT1LFOFFG | DCOFFG);
        SFRIFG1 &= ~OFIFG;
    }while((SFRIFG1 & OFIFG));                          // Test oscillator fault flag
    __bis_status_register(SCG0 | SCG1 | OSCOFF);       // Restore FLL, XT2, XT1LF, DCO
    //Configure Timer A0 to 1 second period
    TA0CCTL0 = CCIE;            // Enable interrupt
    TA0CCR0 = 32768;            // Set period to 32768 (1 second)
    TA0CTL = TASSEL_1 | MC_1 | TACLR;   // SMCLK = 32768Hz, Up Mode
                                        // 1 second period

    __bis_SR_register(LPM0_bits | GIE);     // Enter LPM0, interrupts enabled
    while(1);
}

#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A0_ISR(void){
    P1OUT ^= BIT0;              // Toggle P1.0
}
```

7M lines of code[1]

[1] Boeing 787 Dreamliners *avionics* software. Source `https://informationisbeautiful.net`

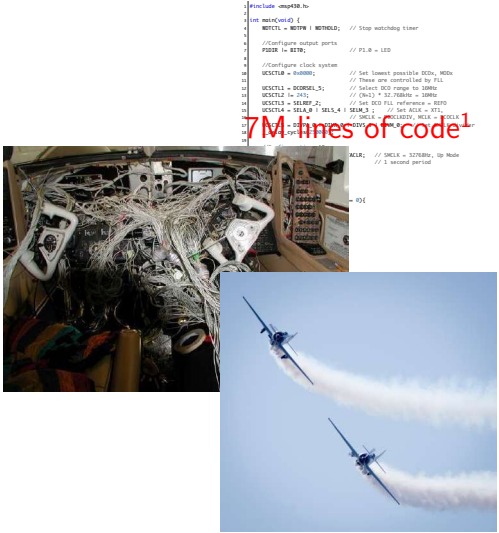
Just Implement It...

```
#include <msp430.h>

int main(void) {
    WDTCTL = WDTPW | WDTHOLD;   // Stop watchdog timer

    //Configure output ports
    P1DIR |= BIT0;              // P1.0 = LED

    //Configure clock system
    UCSCTL0 = 0x0000;           // Set lowest possible DCOx, MODx
                                // These are controlled by FLL
    UCSCTL1 = DCORSEL_5;        // Select DCO range to 16MHz
    UCSCTL2 |= 243;             // (N+1) * 32.768kHz = 16MHz
    UCSCTL3 = SELREF_2;         // Set DCO FLL reference = REFO
    UCSCTL4 = SELA_0 | SELS_4 | SELM_3 ;   // Set ACLK = XT1,
                                // SMCLK = DCOCLKDIV, MCLK = DCOCLK
    UCSCTL5 |= 0x0000;          // No divider for ACLK, SMCLK, MCLK
    UCSCTL6 &= ~XT1OFF;         // Turn on XT1
                                // Wait for XT1 to stabilize
                                // Clear XT1,DCO fault flags
                                // Clear OFIFG fault flag
                                // Wait for fault flags to settle
                                // Set ACLK = XT1
                                // Set SMCLK = DCOCLKDIV
                                // Set MCLK = DCOCLK
    TA0CCTL0 = CCIE;            // Enable interrupt
    TA0CCR0 = 32768;            // Set ACLK = 32768Hz, Up Mode
                                // 1 second period

    __bis_SR_register(LPM0 | GIE);   // Enter LPM0, interrupts enabled
    while (1) {
        __bis_SR_register(LPM0);     // Enter LPM0
        P1OUT ^= 0x01;               // Toggle P1.0 using exclusive-OR
    }
}

// Timer A0 interrupt service routine
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void) {
    P1OUT ^= 0x01;                   // Toggle P1.0 using exclusive-OR
}
```

7M lines of code[1]

[1] Boeing 787 Dreamliners *avionics* software. Source `https://informationisbeautiful.net`

Just Implement It...

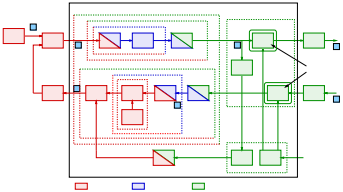


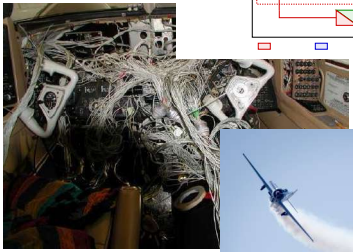
Learn from VLSI design

- unprecedented complexity
- *well-founded*, systematic abstractions
- *clear & rigorous* path from abstract behavior to silicon

[1] Boeing 787 Dreamliners *avionics* software. Source `https://informationisbeautiful.net`

Just Implement It...

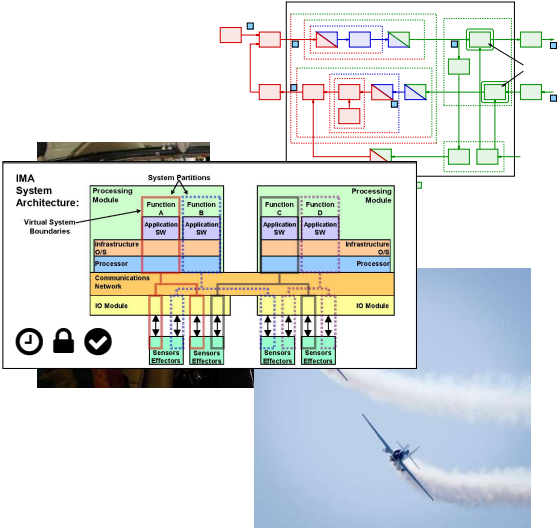




Learn from VLSI design

- unprecedented complexity
- *well-founded*, systematic abstractions
- *clear & rigorous* path from abstract behavior to silicon

Just Implement It...

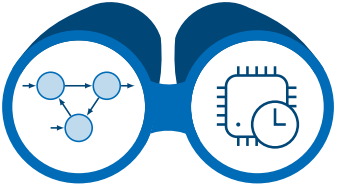


Learn from VLSI design

- unprecedented complexity
- *well-founded*, systematic abstractions
- *clear & rigorous* path from abstract behavior to silicon

[0] C. B. Watkins. "Integrated Modular Avionics: Managing the Allocation of Shared Intersystem Resources". In: *2006 ieee/aiaa 25TH Digital Avionics Systems Conference* (2006), pp. 1–12.

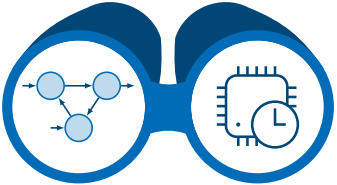
ForSyDe



ForSyDe : **For**mal **Sy**stem **De**sign
ForSyDe ≈ Foresight;

- Definition of foresight (Merriam-Webster)
  1. an act or the power of foreseeing : prescience
     Through foresight she could tell what the outcome would be.
  2. provident care : prudence
     had the foresight to invest his money wisely
  3. an act of looking forward; also : a view forward

ForSyDe

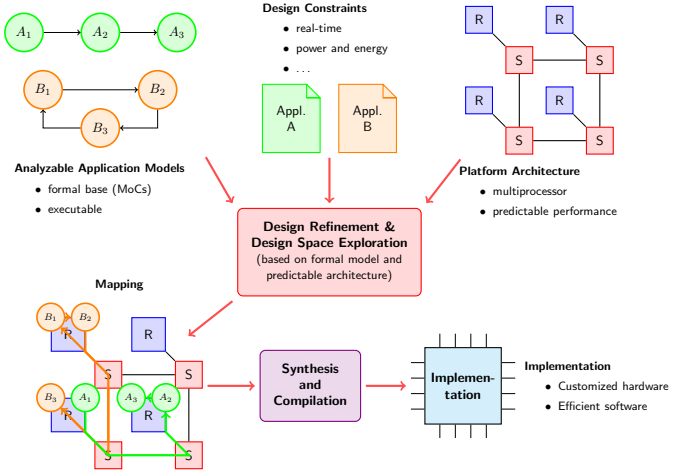


ForSyDe : **For**mal **Sy**stem **De**sign
ForSyDe ≈ Foresight;

- Definition of foresight (Merriam-Webster)
  1. an act or the power of foreseeing : prescience
     Through foresight she could tell what the outcome would be.
  2. provident care : prudence
     had the foresight to invest his money wisely
  3. an act of looking forward; also : a view forward

- high level of abstraction
- formal semantics
- clear separation between modeled aspects
- rigorous design flow
- multiple, heterogeneous predictable targets

ForSyDe



- high level of abstraction
- formal semantics
- clear separation between modeled aspects
- rigorous design flow
- multiple, heterogeneous predictable targets

Behaviors, not Functions

Not only numbers but

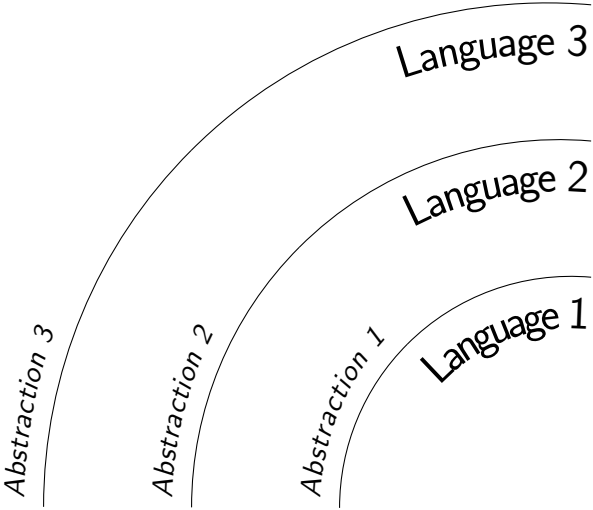
- interaction
- dynamics
- synchronization
- timing
- concurrency/parallelism
- protocol/modes
- communication
- probabilistic distribution
- security primitives
- other non-functional properties...

Interacting aspects!

Behaviors, not Functions

Not only numbers but

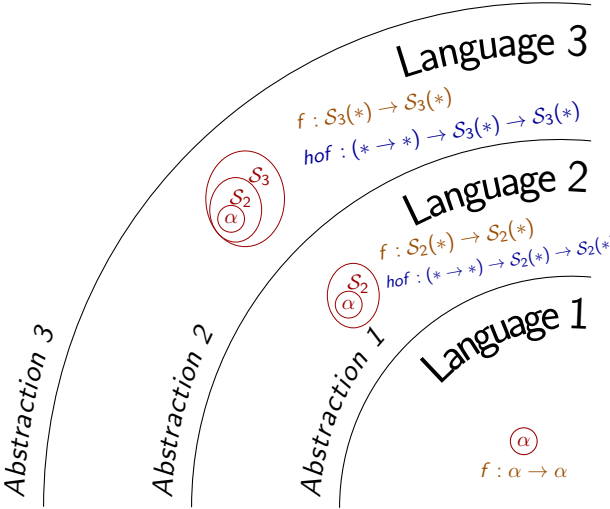
- interaction
- dynamics
- synchronization
- timing
- concurrency/parallelism
- protocol/modes
- communication
- probabilistic distribution
- security primitives
- other non-functional properties...



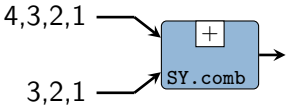
Layered Languages

Each layer consists of:

- a set of structured types
  = *encode properties*
- a set of functions over these types
  = *transformations, rules*
- a set of higher order functions (HOF)
  = *conduits between layers*



A Language for Timed Behavior



SY process

Signals

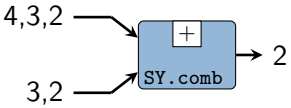
- encode temporal information
- define *tag systems*[1]

Processes

- act according to MoC semantics
- created with *process constructors* (HOF)

[1] E.A. Lee and A. Sangiovanni-Vincentelli. *A framework for comparing models of computation.* Dec. 1998.

A Language for Timed Behavior



SY process

Signals

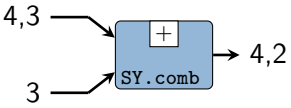
- encode temporal information
- define *tag systems*[1]

Processes

- act according to MoC semantics
- created with *process constructors* (HOF)

[1] Lee and Sangiovanni-Vincentelli, *A framework for comparing models of computation*.

A Language for Timed Behavior



SY process

Signals

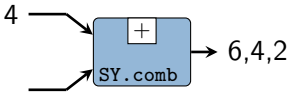
- encode temporal information
- define *tag systems*[1]

Processes

- act according to MoC semantics
- created with *process constructors* (HOF)

[1] Lee and Sangiovanni-Vincentelli, *A framework for comparing models of computation*.

A Language for Timed Behavior



SY process

Signals

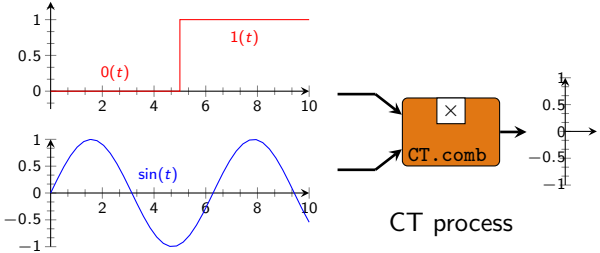
- encode temporal information
- define *tag systems*[1]

Processes

- act according to MoC semantics
- created with *process constructors* (HOF)

[1] Lee and Sangiovanni-Vincentelli, *A framework for comparing models of computation*.

A Language for Timed Behavior



Signals

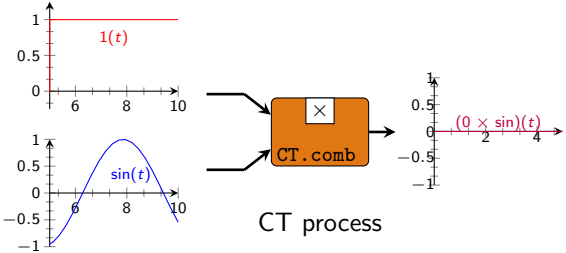
- encode temporal information
- define *tag systems*[1]

Processes

- act according to MoC semantics
- created with *process constructors* (HOF)

[1] Lee and Sangiovanni-Vincentelli, *A framework for comparing models of computation*.

A Language for Timed Behavior



Signals

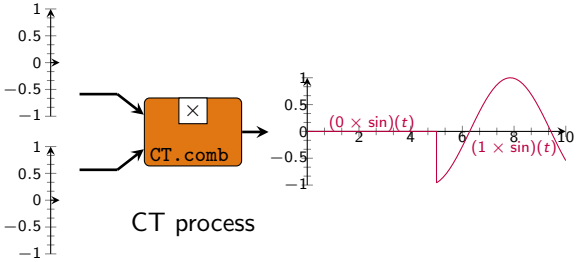
- encode temporal information
- define *tag systems*[1]

Processes

- act according to MoC semantics
- created with *process constructors* (HOF)

[1] Lee and Sangiovanni-Vincentelli, *A framework for comparing models of computation*.

A Language for Timed Behavior



Signals

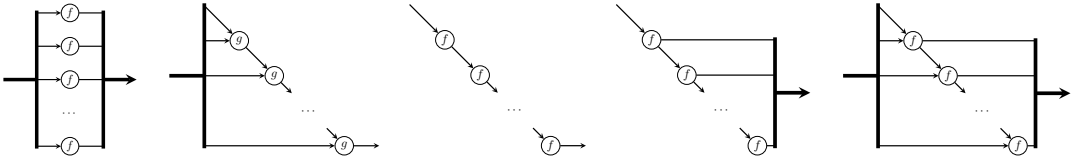
- encode temporal information
- define *tag systems*[1]

Processes

- act according to MoC semantics
- created with *process constructors* (HOF)

[1] Lee and Sangiovanni-Vincentelli, *A framework for comparing models of computation*.

A Language for Structured Parallelism



(a) `farm` (b) `reduce` (c) `pipe` (d) `recur` (e) `prefix`

Regular Structures (e.g. Vectors)

- encode spatial information
- enable *catamorphisms*[2]

Parallel Patterns

- functional relations between elements
- potential for parallel distribution
- created with *skeletons* (HOF)

[2] David B Skillicorn. *Foundations of parallel programming*. 6. Cambridge University Press, 2005.

A Language for Testing Properties

$$\textit{pre-condition} \Rightarrow \textit{statement}$$

Example:

$$\forall a \in List(\alpha) \Rightarrow \texttt{reverse}(\texttt{reverse}\ a) = a$$

Generators

- express pre-conditions
- abstract random data generators
- "smartened" by algebraic properties[3]

Properties & Combinators

- test truth statements
- systematic composition of generators

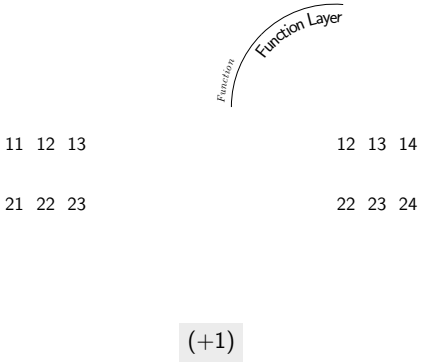
[3] John Hughes. "QuickCheck testing for fun and profit". In: *International Symposium on Practical Aspects of Declarative Languages*. Springer. 2007.

Layered Languages in Tandem

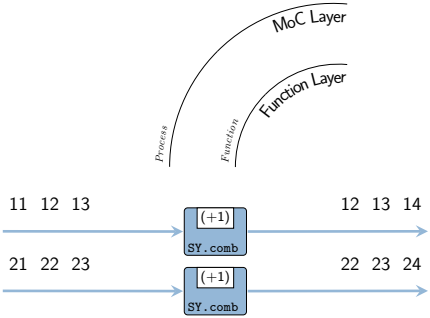
11 12 13

21 22 23

Layered Languages in Tandem

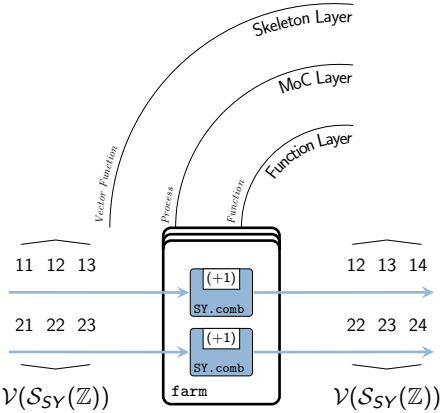


Layered Languages in Tandem



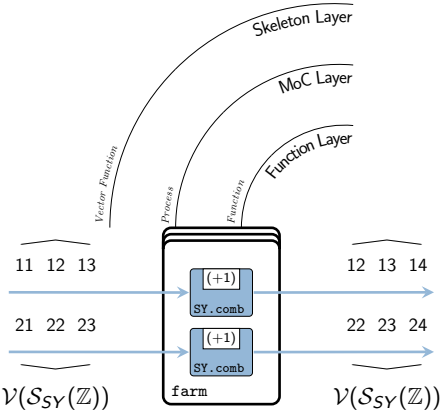
MoC.comb(+1)

Layered Languages in Tandem



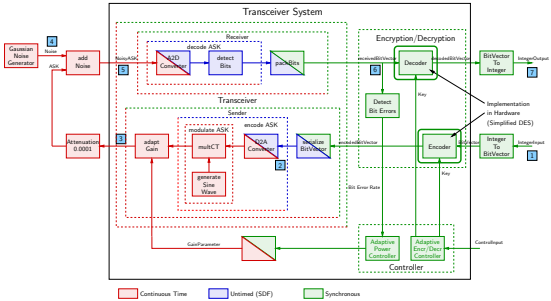

```
Skeleton.farm(MoC.comb(+1))
```

Layered Languages in Tandem

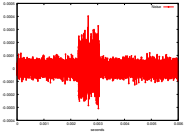


$$\forall s \in \mathcal{S}_{SY}(\mathcal{V}(\alpha)) \Rightarrow \texttt{len(}\ \texttt{Skeleton.farm(MoC.comb(}+1\texttt{))}\ (s)\texttt{)} = \texttt{len}(s)$$

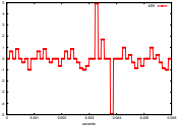
Simulation, Validation, Verification



1 `in = {0,1,2,3,4,5}`



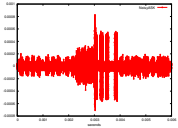
4



3

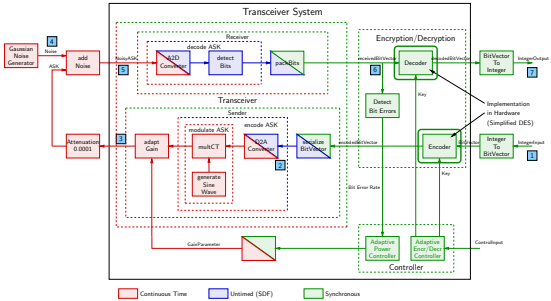
7 `out =`

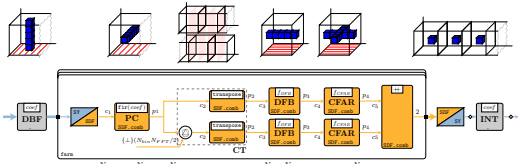
`{0,1,10,3,4,5}`



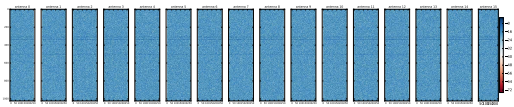
5

Simulation, Validation, Verification

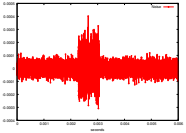




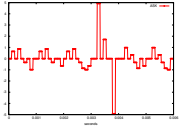
$c_1 = p_1 = N_b;\ c_2 = N_{FFT} \times N_b;\ p_2 = c_4 = p_4 = c_5 = N_b \times N_{FFT};\ c_3 = p_3 = N_{FFT};$



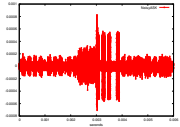
1 `in = {0,1,2,3,4,5}`



4



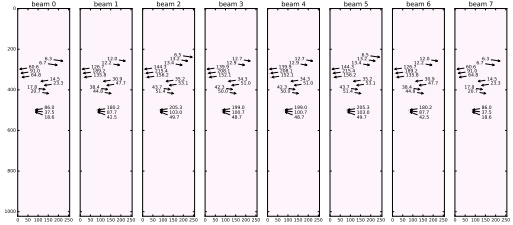
3



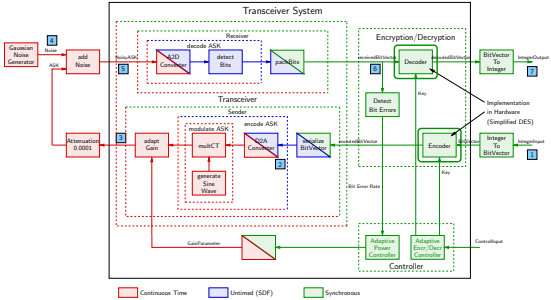
5

7 `out =`

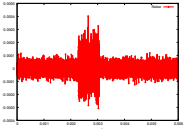
`{0,1,10,3,4,5}`



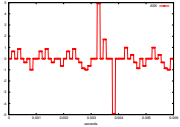
Simulation, Validation, Verification



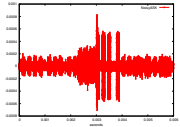
1 `in = {0,1,2,3,4,5}`



4



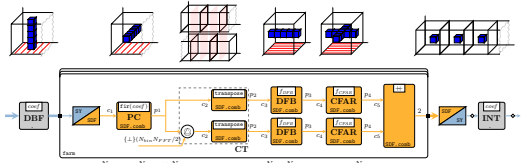
3

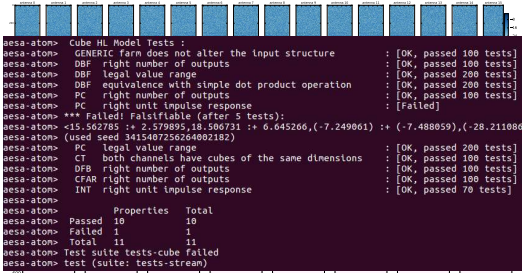


5

7 `out =`

`{0,1,10,3,4,5}`

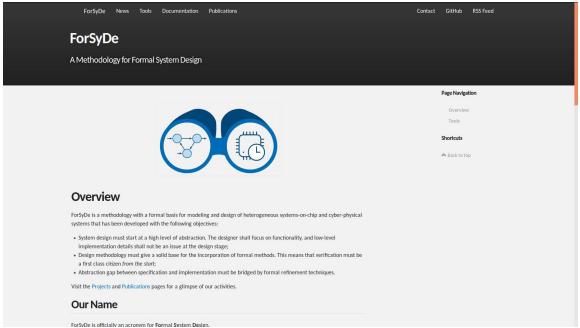




The ForSyDe Modeling Frameworks

- **ForSyDe-Shallow**
  The original modeling framework
- **ForSyDe-Deep**
  Synthesis towards VHDL.
- **ForSyDe-SystemC**
  Closer to imperative targets
- **ForSyDe-Atom**
  Sandbox for novel modeling concepts
- **ForSyDe-Eclipse**
  Eclipse-based GUI frontend

Open source tools available online:



https://forsyde.github.io/

Thank you!