C++ support for better hardware/software co-design in C# with SME

Kenneth Skovhede
Niels Bohr Institute
University of Copenhagen

FSP 2017
2017-09-07
Belgium

Compared to the current solutions, I want something that is:

[✔] Faster

[✔] Less bugs

[✔] Easy to use

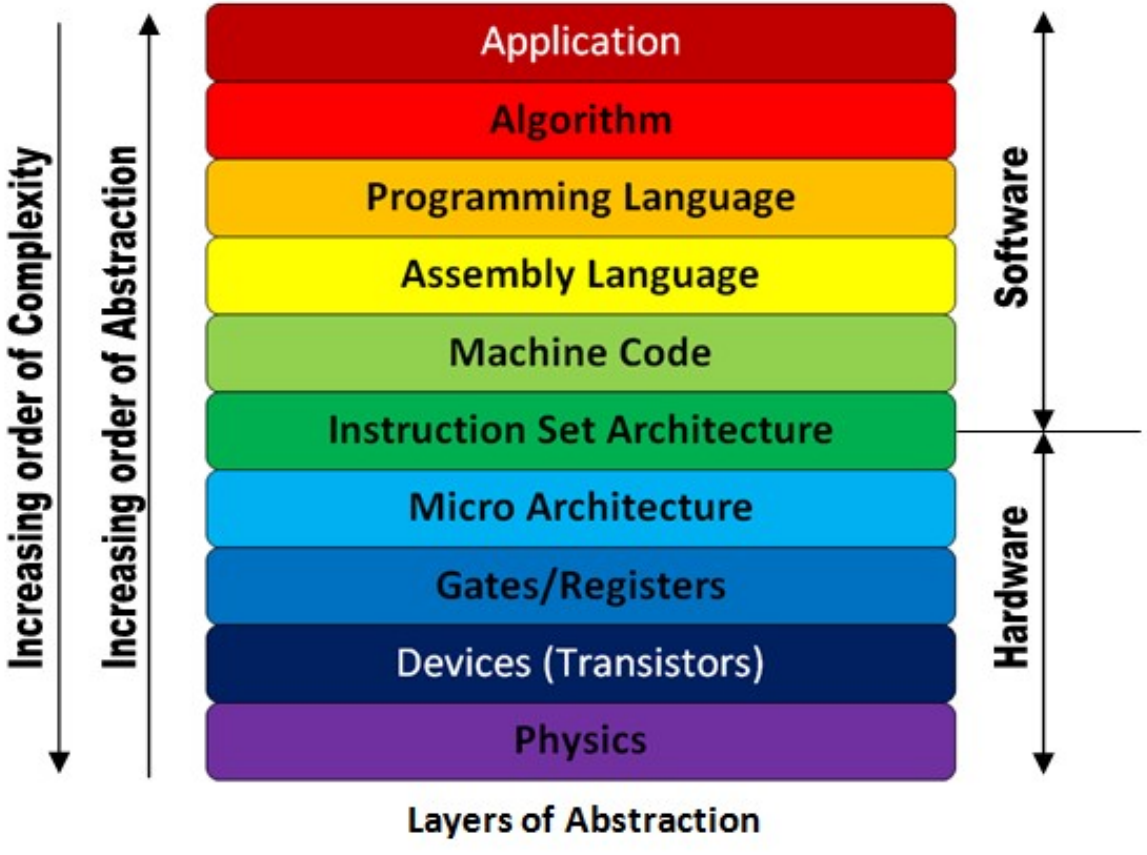
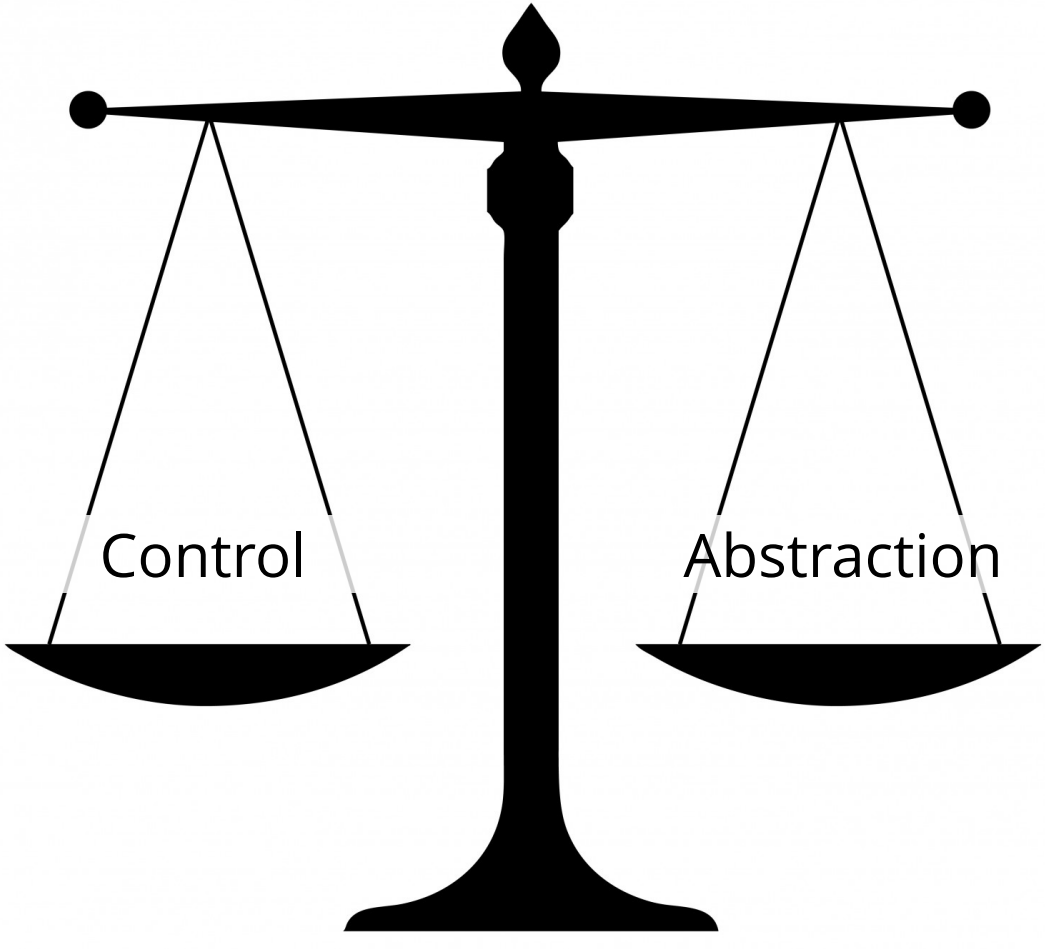


Image from: http://theembeddedguy.com/2016/05/15/layers-of-abstraction/


Control
Abstraction

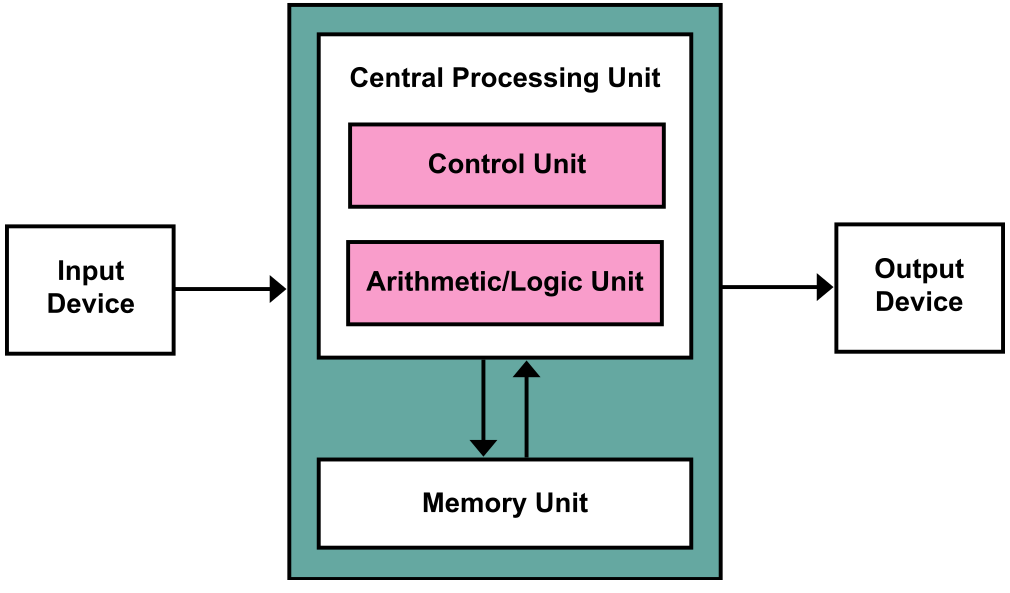
```
int temp;
for(int i2=0; i2<=length; i2++)
{
  for(int j=0; j<length; j++)
  {
    if(array[j]>array[j+1])
    {
      temp=array[j];
      array[j]=array[j+1];
      array[j+1]=temp;
    }
  }
}
```

Bubble sort in C++

Static loop bounds

Map array to HW

Loop dependencies

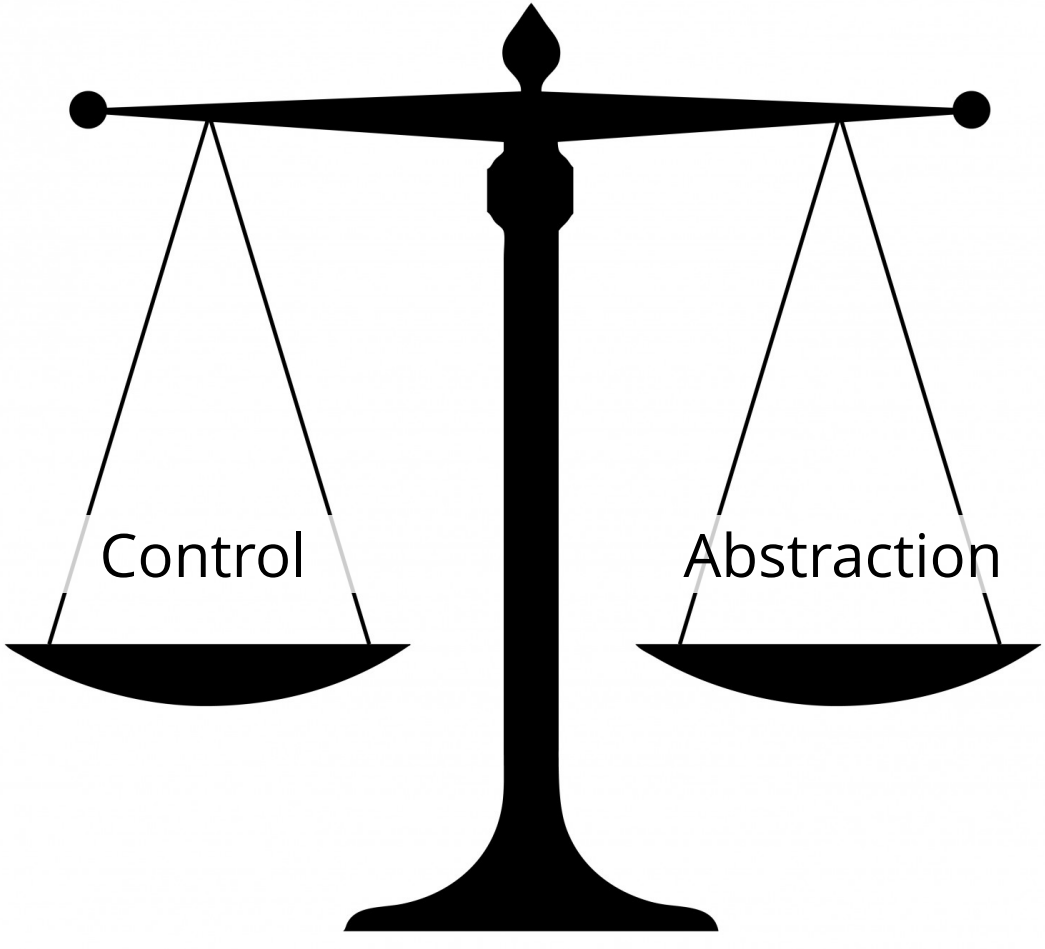


From https://commons.wikimedia.org/wiki/File%3AVon_Neumann_Architecture.svg

If I had asked people what they wanted, they would have said faster horses.

Henry Ford - maybe


APPROVED


Control
Abstraction

SME

Synchronous Message Exchange

Sequential

```
public class SimpleMockMemory : SimpleProcess
{
    [InputBus, OutputBus]
    IMemoryInterface Interface;

    ulong[] m_data = new ulong[1024];

    protected override void OnTick()
    {
        if (Interface.ReadEnabled)
            Interface.ReadValue = m_data[Interface.ReadAddr];

        if (Interface.WriteEnabled)
            m_data[Interface.WriteAddr] = Interface.WriteValue;
    }
}
```

- Processes
- Busses

Concurrency

```
public interface IMemoryInterface : IBus
{
    [InitialValue(false)]
    bool WriteEnabled { get; set; }
    [InitialValue(false)]
    bool ReadEnabled { get; set; }

    uint ReadAddr { get; set; }
    uint WriteAddr { get; set; }

    ulong WriteValue { get; set; }
    ulong ReadValue { get; set; }
}
```

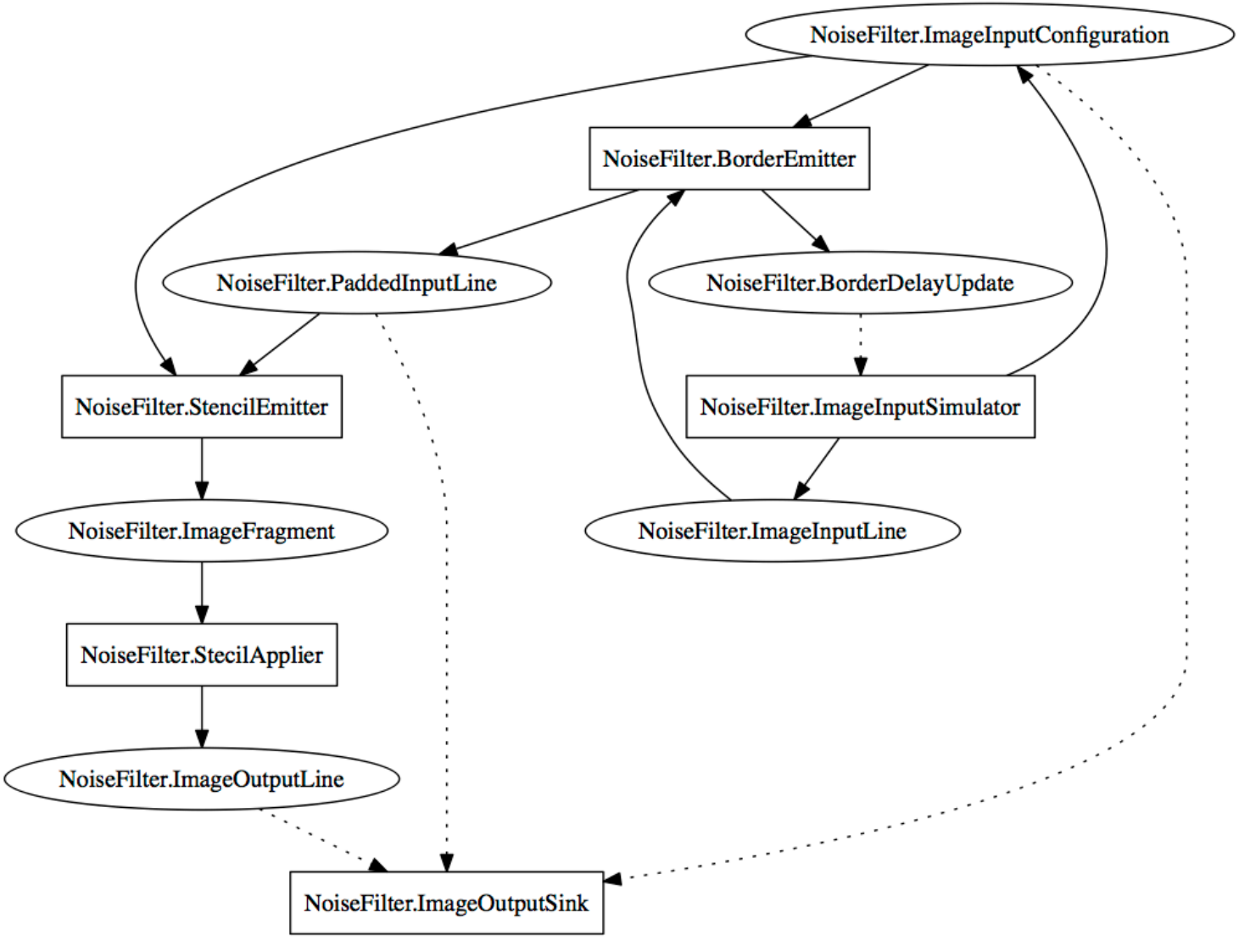
```
public interface IMemoryInterface : IBus
{
    [InitialValue(false)]
    bool WriteEnabled { get; set; }
    [InitialValue(false)]
    bool ReadEnabled { get; set; }

    uint ReadAddr { get; set; }
    uint WriteAddr { get; set; }

    ulong WriteValue { get; set; }
    ulong ReadValue { get; set; }
}
```

```
public class TickCounterMemory : SimpleProcess
{
    [InputBus]
    IInputBus Input;
    [OutputBus]
    IOutputBus Output;

    protected override void OnTick()
    {
        var before = Output.Ticks;
        if (Input.Reset)
        {
            Output.Ticks = 0;
            Output.LastTicks = Output.Ticks;
        }
        else
        {
            Output.Ticks++;
        }

        // before is always the same as after,
        // because the output value is not propagated
        // immediately, but waits for a tick
        var after = Output.Ticks;
    }
}
```

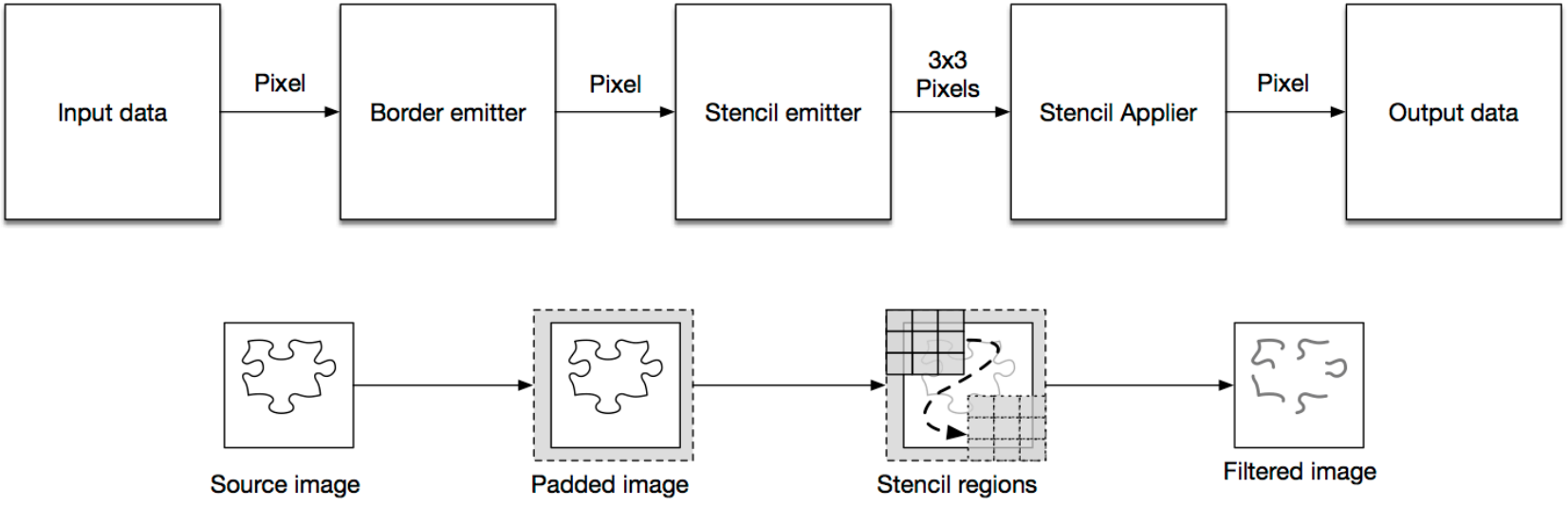
```
public class TickCounterMemory : SimpleProcess
{
    [InputBus]
    IInputBus Input;
    [OutputBus]
    IOutputBus Output;

    protected override void OnTick()
    {
        var before = Output.Ticks;
        if (Input.Reset)
        {
            Output.Ticks = 0;
            Output.LastTicks = Output.Ticks;
        }
        else
        {
            Output.Ticks++;
        }

        // before is always the same as after,
        // because the output value is not propagated
        // immediately, but waits for a tick
        var after = Output.Ticks;
    }
}
```

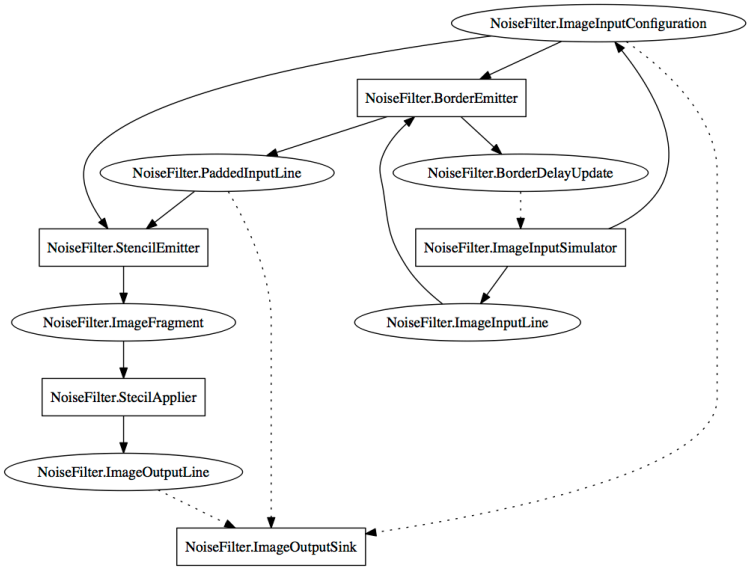
Like CSP in that the collection of busses is communicated as a single channel action

Like a KPN because there is no blocking


NoiseFilter.ImageInputConfiguration
NoiseFilter.BorderEmitter
NoiseFilter.PaddedInputLine
NoiseFilter.BorderDelayUpdate
NoiseFilter.StencilEmitter
NoiseFilter.ImageInputSimulator
NoiseFilter.ImageFragment
NoiseFilter.ImageInputLine
NoiseFilter.StecilApplier
NoiseFilter.ImageOutputLine
NoiseFilter.ImageOutputSink

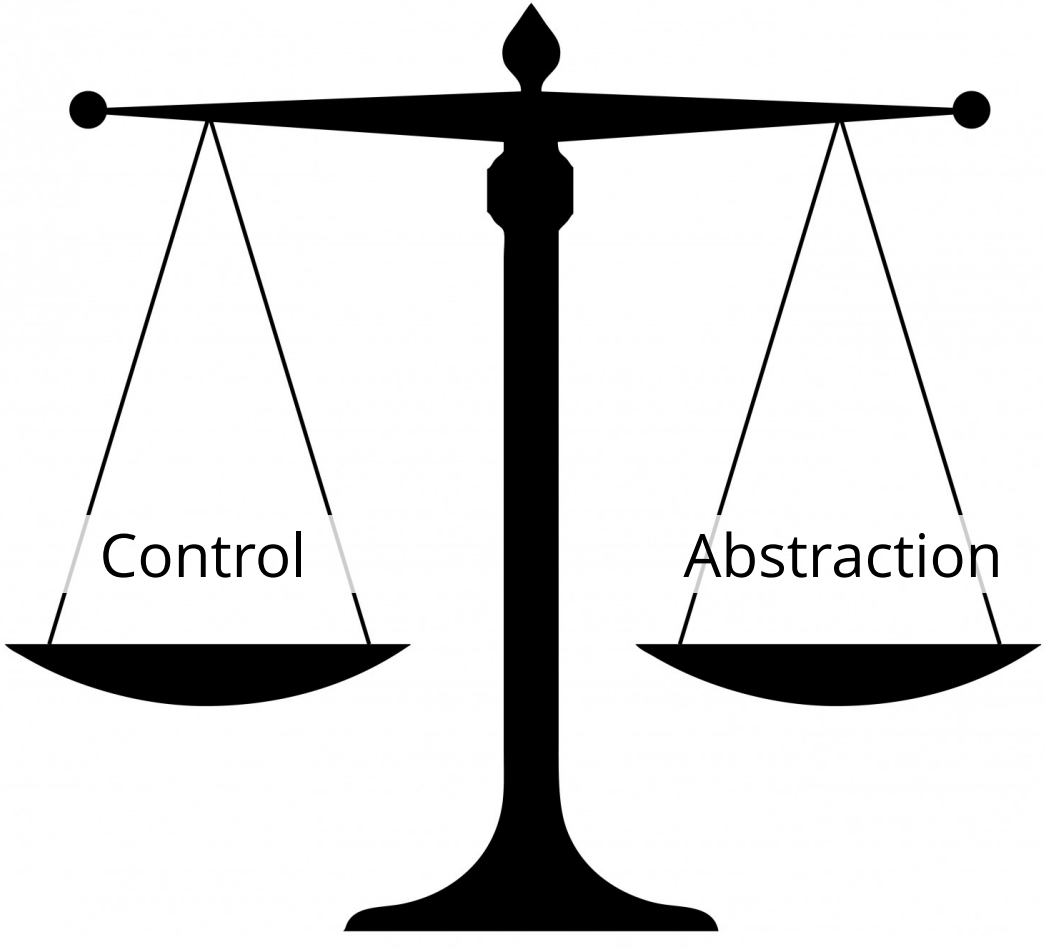
Stencil network example

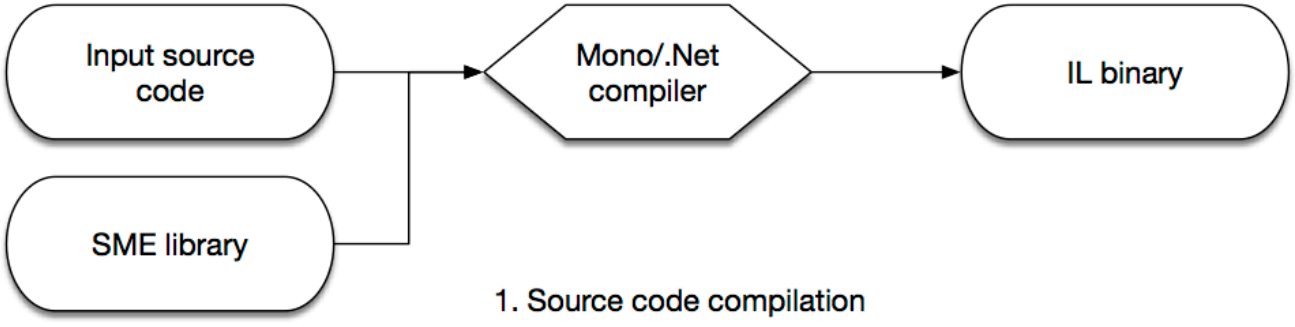




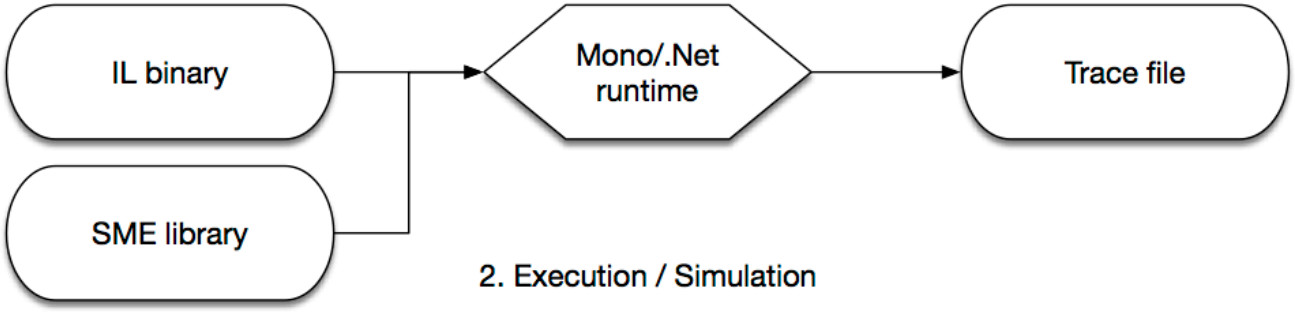

```
[InputBus] ImageFragment Input;
[OutputBus] ImageOutputLine Output;
static readonly byte[] FILTER = new byte[] {
    1,1,1, 1,1,1, 1,1,1,
    1,1,1, 1,1,1, 1,1,1,
    1,1,1, 1,1,1, 1,1,1
};
```

```
protected override void OnTick()
{
    Output.IsValid = false;
    for (var i = 0; i < COLOR_WIDTH; i++)
        Output.Color[i] = 0;
    for (var i = 0; i < m_buffer.Length; i++)
        m_buffer[i] = 0;
    if (Input.IsValid)
    {
        for (var i = 0; i < Input.Data.Length; i += COLOR_WIDTH)
            for (var j = 0; j < m_buffer.Length; j++)
                m_buffer[j] += FILTER[i + j] * Input.Data[i + j];
        for (var i = 0; i < m_buffer.Length; i++)
            Output.Color[i] = (byte)(m_buffer[i] / FILTER_SUMS[i]);
        Internal.Index++;
        Output.IsValid = true;
    }
}
```

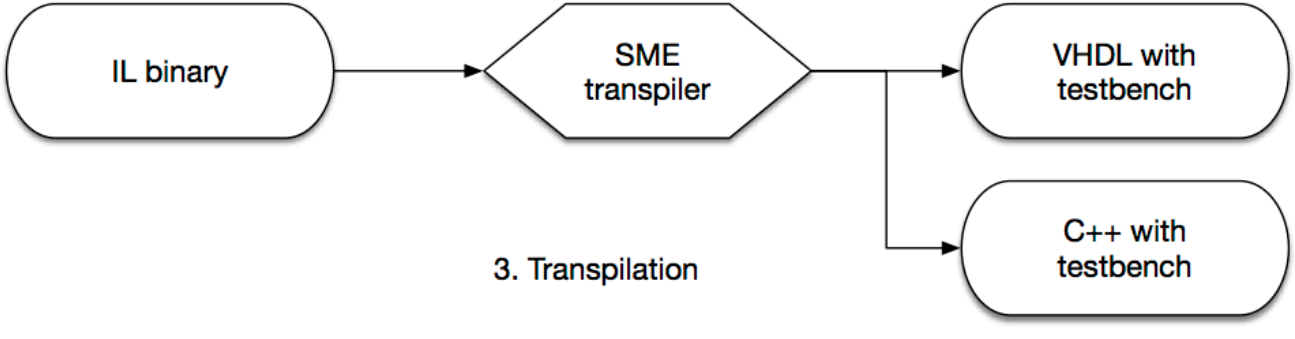

Control
Abstraction



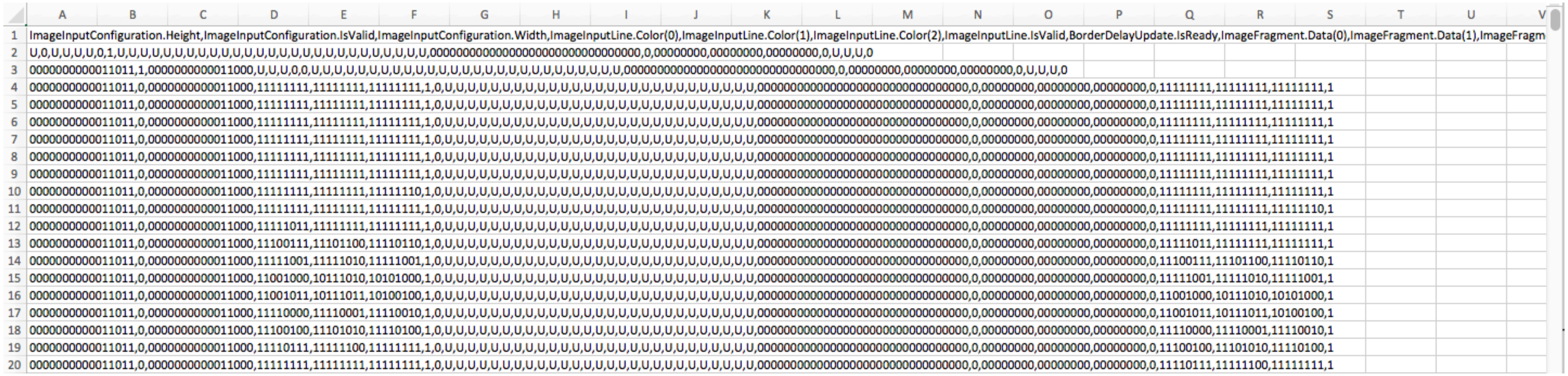
1. Source code compilation



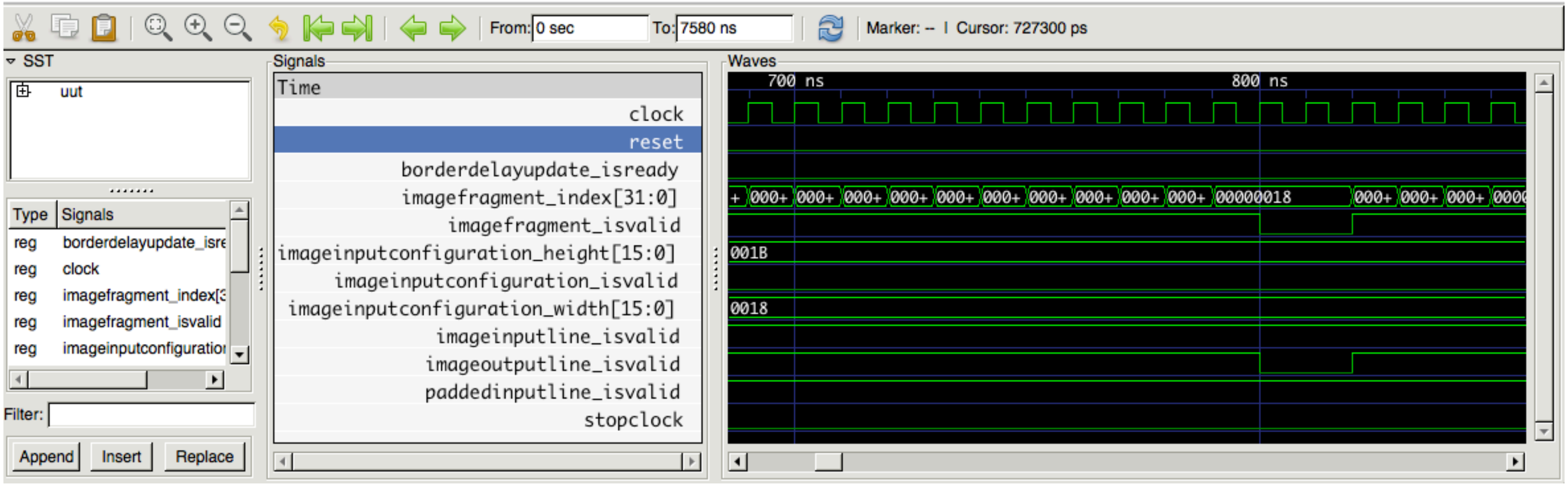
2. Execution / Simulation



3. Transpilation



	A
1	ImageInputConfiguration.Height,ImageInputConfiguration.IsValid,ImageInputConfiguration.Width,ImageInputLine.Color(0),ImageInputLine.Color(1),ImageInputLine.Color(2),ImageInputLine.IsValid,BorderDelayUpdate.IsReady,ImageFragment.Data(0),ImageFragment.Data(1),ImageFragm
2	U,0,U,U,U,U,0,1,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,U,U,U,0
3	0000000000011011,1,0000000000011000,U,U,U,0,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,U,U,U,0
4	0000000000011011,0,0000000000011000,11111111,11111111,11111111,1,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,11111111,11111111,11111111,1
5	0000000000011011,0,0000000000011000,11111111,11111111,11111111,1,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,11111111,11111111,11111111,1
6	0000000000011011,0,0000000000011000,11111111,11111111,11111111,1,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,11111111,11111111,11111111,1
7	0000000000011011,0,0000000000011000,11111111,11111111,11111111,1,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,11111111,11111111,11111111,1
8	0000000000011011,0,0000000000011000,11111111,11111111,11111111,1,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,11111111,11111111,11111111,1
9	0000000000011011,0,0000000000011000,11111111,11111111,11111111,1,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,11111111,11111111,11111111,1
10	0000000000011011,0,0000000000011000,11111111,11111111,11111110,1,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,11111111,11111111,11111111,1
11	0000000000011011,0,0000000000011000,11111111,11111111,11111111,1,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,11111111,11111111,11111110,1
12	0000000000011011,0,0000000000011000,11111011,11111111,11111111,1,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,11111111,11111111,11111111,1
13	0000000000011011,0,0000000000011000,11100111,11101100,11110110,1,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,11111011,11111111,11111111,1
14	0000000000011011,0,0000000000011000,11111001,11111010,11111001,1,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,11100111,11101100,11110110,1
15	0000000000011011,0,0000000000011000,11001000,10111010,10101000,1,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,11111001,11111010,11111001,1
16	0000000000011011,0,0000000000011000,11001011,10111011,10100100,1,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,11001000,10111010,10101000,1
17	0000000000011011,0,0000000000011000,11110000,11110001,11110010,1,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,11001011,10111011,10100100,1
18	0000000000011011,0,0000000000011000,11100100,11101010,11110100,1,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,11110000,11110001,11110010,1
19	0000000000011011,0,0000000000011000,11110111,11111100,11111111,1,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,11100100,11101010,11110100,1
20	0000000000011011,0,0000000000011000,11111111,11111111,11111111,1,0,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,U,00000000000000000000000000000000,0,00000000,00000000,00000000,0,11110111,11111100,11111111,1



Supported

- Control - if, switch, fixed iteration loops
- Structure - functions
- Data - anything static, 2-bit ... n-bit
- Boolean logic - and, or, xor, etc
- Bitwise - shifts, and, or, xor
- Integer arithmetics - add, sub, mul, div
- Arrays - fixed length

Not supported

(supported in modelling, but not in transpiler)

- Anything dynamic - strings, lists, objects
- Floating point - single, double, decimal

- IP needs simulation implementation

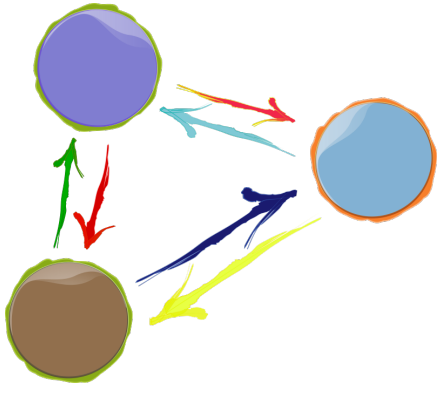
```
private static uint SubByte (uint a) {
  uint value = 0xff & a;
  uint result = SBox[value];
  value = 0xff & (a >> 8);
  result |= (uint)SBox[value] << 8;
  value = 0xff & (a >> 16);
  result |= (uint)SBox[value] << 16;
  value = 0xff & (a >> 24);
  return result | (uint)(SBox[value] << 24);
}
```

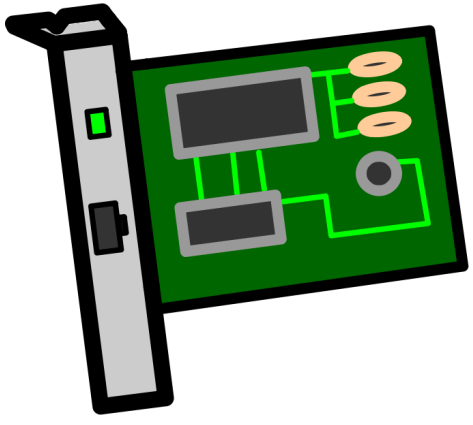
C# version

```
system_uint32 AESCore::SubByte(system_uint32 a) {
    system_uint32 num = 0;
    system_uint32 num2 = 0;

    num = 255 & a;
    num2 = (system_uint32)AES256CBC_AESCore_SBox[(system_int32)num];
    num = 255 & (a >> 8);
    num2 |= (system_uint32)
((system_uint32)AES256CBC_AESCore_SBox[(system_int32)num] << 8);
    num = 255 & (a >> 16);
    num2 |= (system_uint32)
((system_uint32)AES256CBC_AESCore_SBox[(system_int32)num] << 16);
    num = 255 & (a >> 24);
    return num2 | (system_uint32)
((system_uint32)AES256CBC_AESCore_SBox[(system_int32)num] << 24);
}
```

C++ version

```
pure function SubByte(constant a: in T_SYSTEM_UINT32) return T_SYSTEM_UINT32 is
    variable tmpvar_1: T_SYSTEM_UINT32;
    variable num: T_SYSTEM_UINT32;
    variable num2: T_SYSTEM_UINT32;
begin
    num := STD_LOGIC_VECTOR(TO_UNSIGNED(255, T_SYSTEM_UINT32'length)) and a;
    num2 := STD_LOGIC_VECTOR(resize(UNSIGNED(AES256CBC_AESCore_SBox(TO_INTEGER(SIGNED(num)))), T_SYSTEM_UINT32'length));
    num := STD_LOGIC_VECTOR(TO_UNSIGNED(255, T_SYSTEM_UINT32'length)) and STD_LOGIC_VECTOR((shift_right(UNSIGNED(a), 8)));
    num2 := num2 or STD_LOGIC_VECTOR((shift_left(UNSIGNED(STD_LOGIC_VECTOR(resize(UNSIGNED(AES256CBC_AESCore_SBox(TO_INTEGER(SIGNED(num)))),
                T_SYSTEM_UINT32'length))), 8)));
    num := STD_LOGIC_VECTOR(TO_UNSIGNED(255, T_SYSTEM_UINT32'length)) and STD_LOGIC_VECTOR((shift_right(UNSIGNED(a), 16)));
    num2 := num2 or STD_LOGIC_VECTOR((shift_left(UNSIGNED(STD_LOGIC_VECTOR(resize(UNSIGNED(AES256CBC_AESCore_SBox(TO_INTEGER(SIGNED(num)))),
                T_SYSTEM_UINT32'length))), 16)));
    num := STD_LOGIC_VECTOR(TO_UNSIGNED(255, T_SYSTEM_UINT32'length)) and STD_LOGIC_VECTOR((shift_right(UNSIGNED(a), 24)));
    tmpvar_1 := num2 or STD_LOGIC_VECTOR((shift_left(UNSIGNED(STD_LOGIC_VECTOR(resize(UNSIGNED(AES256CBC_AESCore_SBox(TO_INTEGER(SIGNED(num)))),
                T_SYSTEM_UINT32'length))), 24)));
    return tmpvar_1;
end SubByte;
```





C#

C++

FPGA

Shared SME source code

	Wall-clock time	Time pr. cycle
C++	4s	0.011ms
C#	30s	0.085ms
GHDL	1m 14s	0.210ms

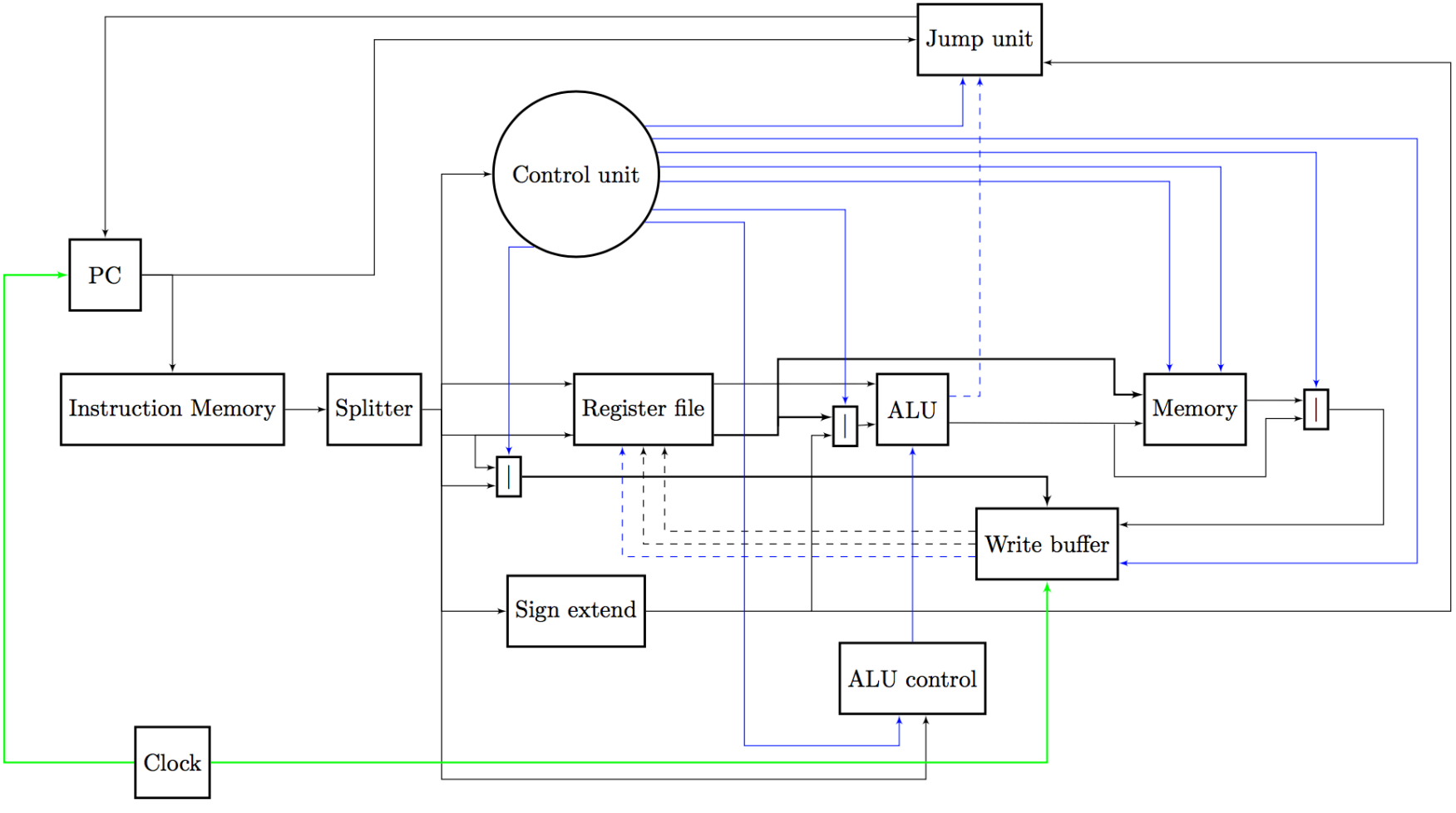
ColorBin execution times

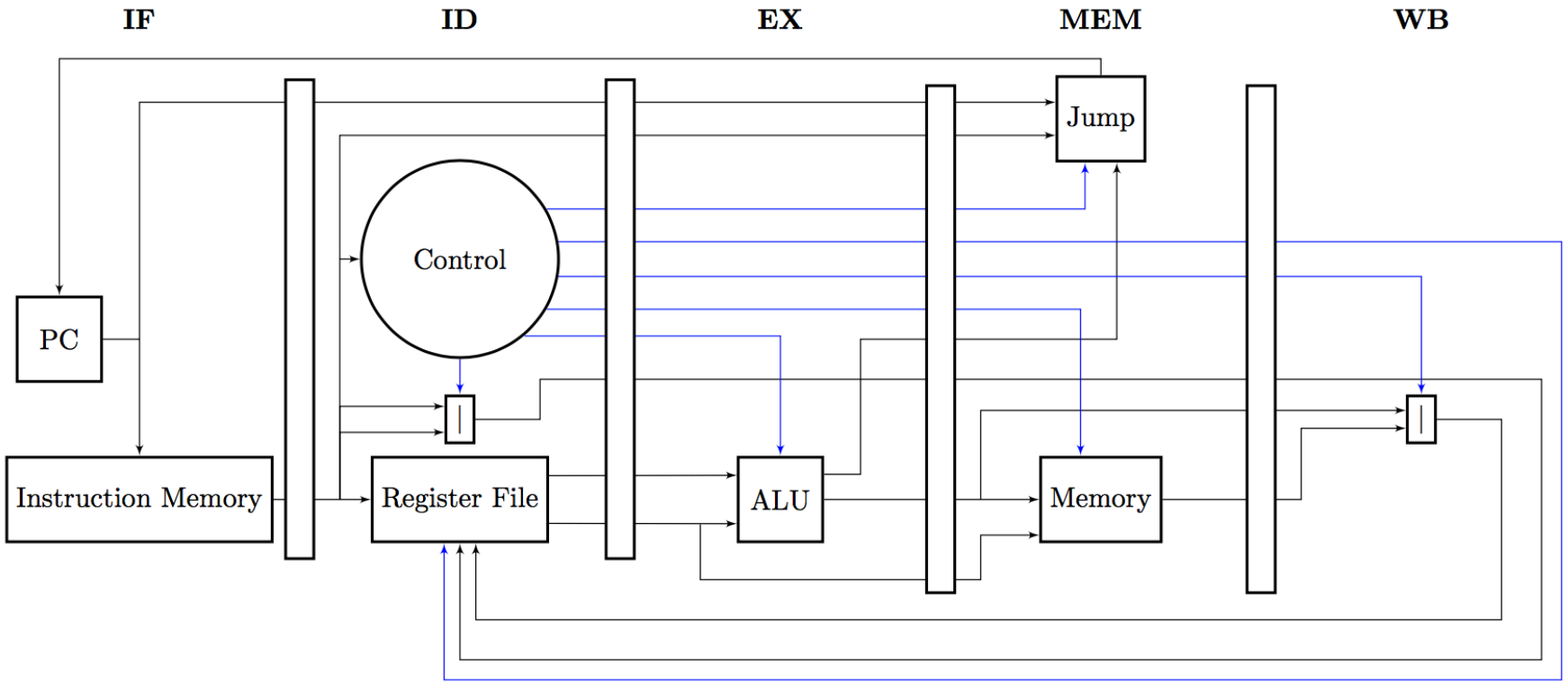
	Wall-clock time	Time pr. cycle
C++	11s	0.031ms
C#	1m 56s	0.325ms
GHDL	35m 19s	5.949ms

Stencil execution times

	Wall-clock time	Time pr. cycle
C++	450ms	0.449ms
C#	1s 251ms	1.249ms
GHDL	4s 946ms	4.936ms

10,000 AES CBC rounds


Jump unit
Control unit
PC
Instruction Memory
Splitter
Register file
ALU
Memory
Write buffer
Sign extend
ALU control
Clock


IF
ID
EX
MEM
WB
Jump
Control
PC
Instruction Memory
Register File
ALU
Memory

Planned work

- Communication links
  - C# <-> C++ via memory or pipes
  - C# <-> FPGA via AXI, DRAM or ACP
- Components
  - VGA driver
  - Block RAM
  - DSP
- PySME equivalence
  - Shared transpiler