

# An RTL Designers Guide to the HLS Universe

Stefen Boyd -TLM Systems

Sergio Ramirez - Cadence Design Systems

# Agenda

- What is High Level Synthesis?
- History
- Barriers to Adoption
- Coding Guidelines for a High Throughput Design
- Summary

# Agenda

- What is High Level Synthesis?
- History
- Barriers
- Guidelines
- Summary

# What is High Level Synthesis?

- Focus on C++ / SystemC to RTL
- Higher abstraction than RTL
  - However it has a lower abstraction than most software
- Historical strengths
  - Datapath
  - Digital Signal Processing (Wireless / Video)
- Recent developments
  - Sequential logic
  - Transaction Level Modeling

# Agenda

- What is High Level Synthesis?
- **History**
- Barriers
- Guidelines
- Summary

# History

- Rosenthal in Tübingen and others laid the R&D foundation for HLS tool development
- First generation tools:
  - Mentor Graphic's Monet
  - Synopsys Behavioral Compiler
- Second generation:
  - Forte Synthesizer (~2002)
    - System C
  - Calypto Catapult Synthesis (2004)
    - Originally C++ then System C
  - Cadence C-to-Silicon Compiler (2008)

# Agenda

- What is High Level Synthesis?
- History
- **Barriers to Adoption**
- Guidelines
- Summary

# Psychological Barriers to Adoption

- Wrong Expectations
  - Engineering managers wrongly believe that a piece of code from internet or old software model will result in efficient Hardware ('a monkey could do it')

## Fear of displacement:

- Hardware designers feel unnecessary
  - Wrong expectation
- Resistance to change:
  - Different design methodology and thought processes



# Technical Barriers to Adoption

- Immaturity of synthesis technology
- Missing standard coding style for High Level Synthesis C++/SystemC
  - The code for the three available HLS tools is somewhat different
- Niche technology:
  - Primary success in Wireless / Video
- Skill set
- Verbosity

# Skill Set Barrier

- Broader skill set required than RTL designer
  - System Skills (DSP, numerical methods, algorithms)
  - Software Engineering Skills (C++, Object Oriented Programming)
  - Hardware Design Skills (Latency, throughput, memory bandwidth)
  - RTL coding
- Usually spread across many engineers and departments

# Language Verbosity Barrier

## SystemC as RTL not compelling

Verilog

```
module dff (input clk, rstn,  
           input [7:0] data_in,  
           output [7:0] data_out);  
  always @(posedge clk)  
    if (!rstn)  
      data_out <= 8'b0;  
    else  
      data_out <= data_in;  
  
endmodule // dff
```

SystemC

```
#include "systemc.h"  
#include "dff.h"  
  
void dff::dff_func() {  
  if (!rstn.read()) {  
    data_out.write(0);  
  } else {  
    data_out.write(data_in.read());  
  }  
}
```

```
SC_MODULE(dff) {  
  
  sc_in<bool>      clk;  
  sc_in<bool>      rstn;  
  sc_in<sc_int<8>> data_in;  
  sc_out<sc_int<8>> data_out;  
  
  SC_CTOR(dff)  
  {  
    SC_METHOD(dff_func);  
    dont_initialize();  
    sensitive << clk.pos();  
  }  
  void dff_func();  
};  
#endif
```

# Convincing Skeptics

- Different Hardware can be obtained from the same System C
  - 2 instances of IP are needed
    - One fast/big, One slow/small
  - Hand RTL fast, but big
  - Smaller version needed
- Scenario 1
  - Design two versions of RTL – despite schedule hit
- Scenario 2
  - HLS generates two versions of RTL - Increased productivity!!!

# Agenda

- What is High Level Synthesis?
- History
- Barriers
- **Guidelines**
- Summary

# IDCT Example

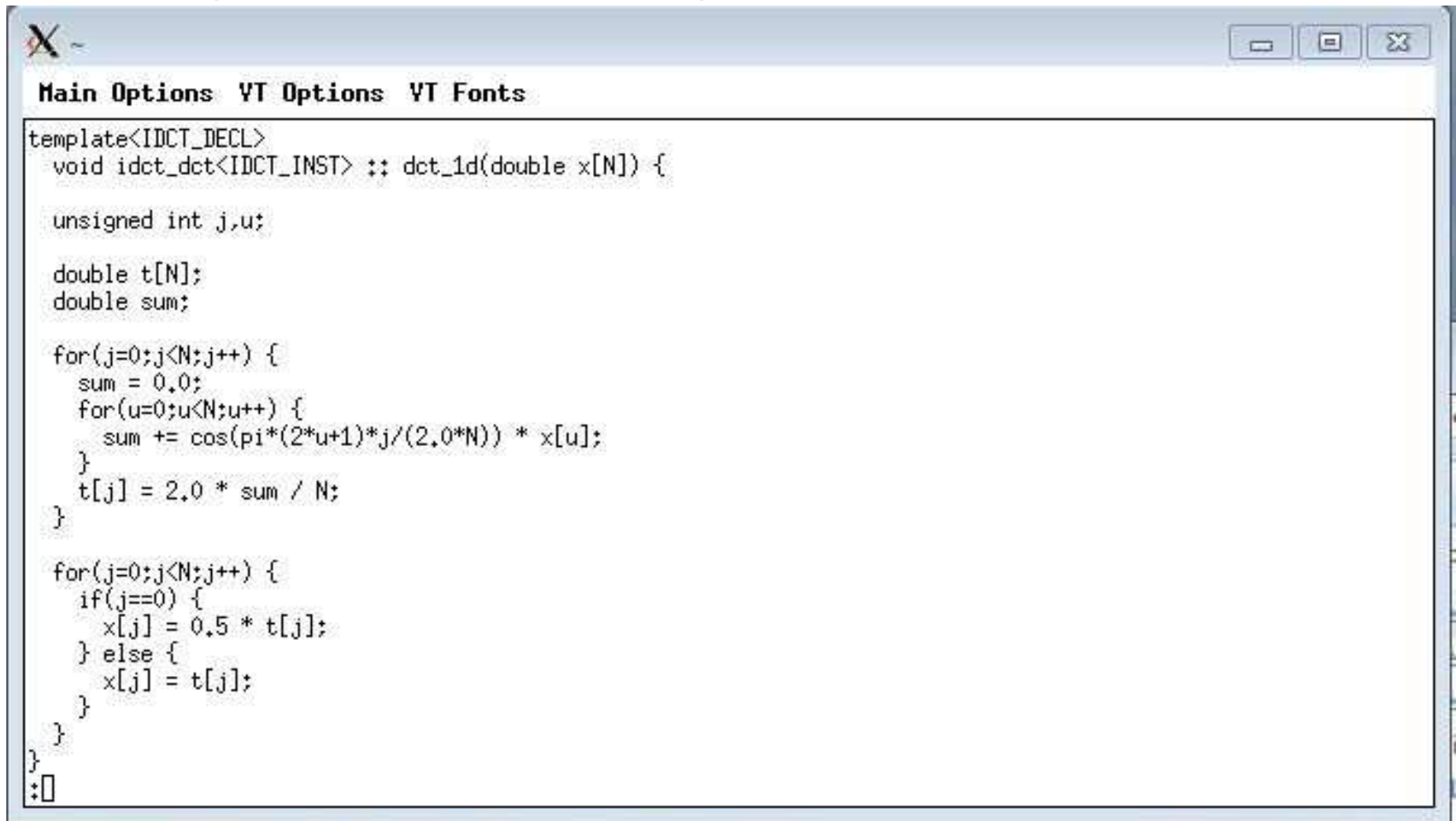
- Complex Design Used in Video Application
- Data path intensive
  - The design performs the following transformation

$$B(k_1, k_2) = \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} 4 \cdot A(i, j) \cdot \cos\left[\frac{\pi \cdot k_1}{2 \cdot N_1} \cdot (2 \cdot i + 1)\right] \cdot \cos\left[\frac{\pi \cdot k_2}{2 \cdot N_2} \cdot (2 \cdot j + 1)\right]$$

- Performance Requirements
  - Latency: 80 clock cycles before the first pixel out.
  - Throughput: One IDCT every 64 clock cycles

# Software Model

- Defines computation but not hardware friendly
  - Design requires very high memory bandwidth



```
template<IDCT_DECL>
void idct_dct<IDCT_INST> :: dct_1d(double x[N]) {

    unsigned int j,u;

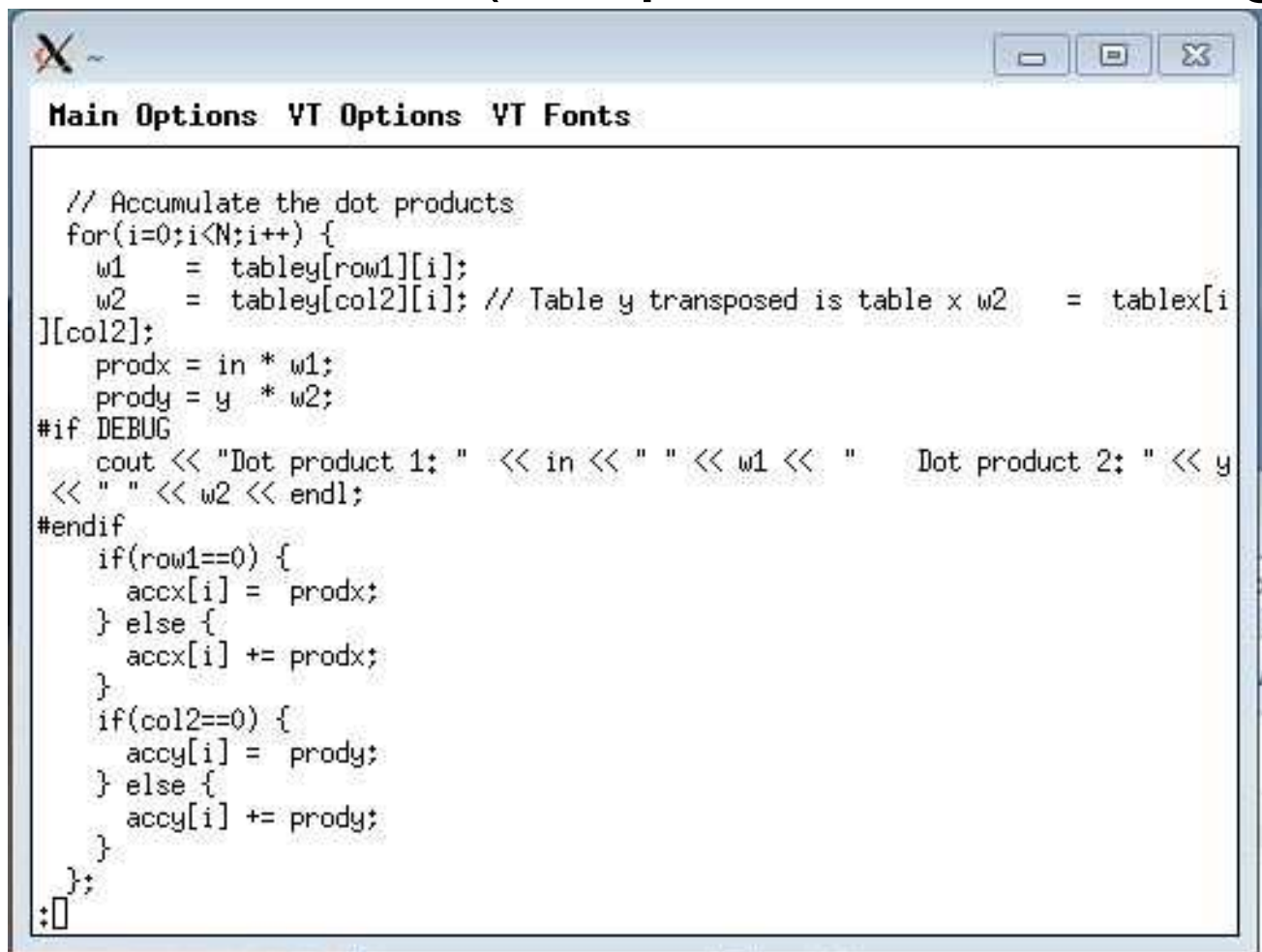
    double t[N];
    double sum;

    for(j=0;j<N;j++) {
        sum = 0.0;
        for(u=0;u<N;u++) {
            sum += cos(pi*(2*u+1)*j/(2.0*N)) * x[u];
        }
        t[j] = 2.0 * sum / N;
    }

    for(j=0;j<N;j++) {
        if(j==0) {
            x[j] = 0.5 * t[j];
        } else {
            x[j] = t[j];
        }
    }
}
};
```

# SystemC Model Guidelines

- Describe computation
- And describes underlying pipelined architecture.
- Parameterizable (template based design)



```
// Accumulate the dot products
for(i=0;i<N;i++) {
    w1  = tabley[row1][i];
    w2  = tabley[col2][i]; // Table y transposed is table x w2    = tablex[i
][col2];
    prodx = in * w1;
    prody = y * w2;
#if DEBUG
    cout << "Dot product 1: " << in << " " << w1 << "    Dot product 2: " << y
<< " " << w2 << endl;
#endif
    if(row1==0) {
        accx[i] = prodx;
    } else {
        accx[i] += prodx;
    }
    if(col2==0) {
        accy[i] = prody;
    } else {
        accy[i] += prody;
    }
};
};
```



# SystemC Model Guidelines (cont.)

- Transcendental functions described as tables
  - Represents ROM or constants in hardware
- Accumulation hardware of the pipeline described in the SystemC code
- SystemC code written with “Pipelining” in mind
- The data path explicitly described
  - Note: accumulation loop

# Memory Access

- Designed to minimize memory bandwidth
  - One read port
  - One write port
- Single read / write port possible
  - Throughput of one pixel every two clock cycles
  - Alternative implementation from same SystemC

# Agenda

- What is High Level Synthesis?
- History
- Barriers
- Guidelines
- **Summary**

# Summary

- Coding style provided runs in top three HLS tools with slight modification
- A family of IDCTs can be generated from the provided code (not a single RTL snapshot)
- Result will overcome barriers
  - Time to Silicon
  - Area and Performance
- Right level of abstraction