Sphere Detector for 802.16e Broadband Wireless Systems Implementation on FPGAs using High-Level Synthesis Tools

Juanjo Noguera, Stephen Neuendorffer, Kees Vissers, Chris Dick, Xilinx, Inc

Sven Van Haastregt, Leiden University, The Netherlands

Jesus Barba, University of Castilla-La Mancha, Spain

Agenda

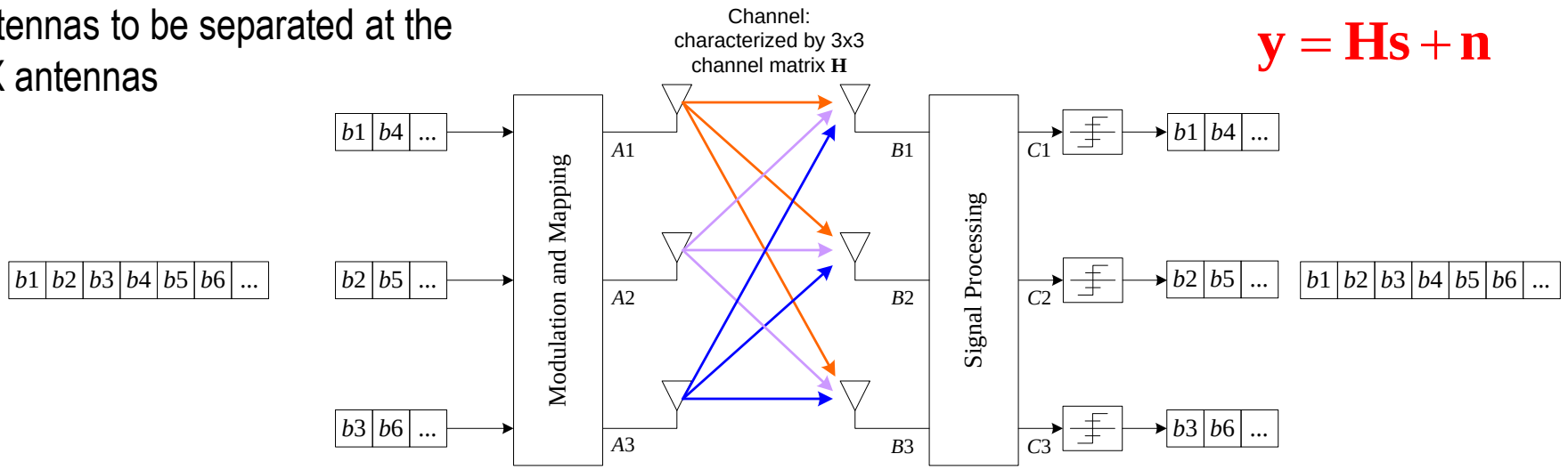
- Introduction to Spatial Multiplexing in MIMO systems
- MIMO Sphere Decoder functionality and requirements
- Overview of High-Level Synthesis for FPGAs
- MIMO Sphere Decoder implementation using AutoESL
- Summary

Spatial Multiplexing MIMO Link

Exploit multipath to permit different antennas to be separated at the RX antennas

Received vector:

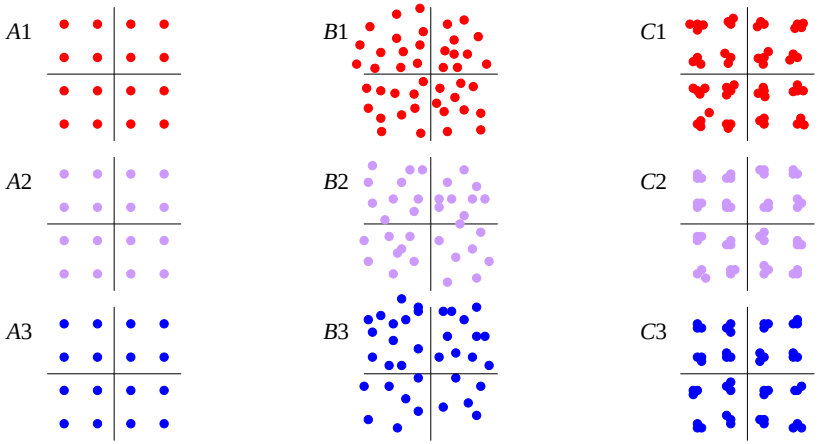
$$\mathbf{y} = \mathbf{Hs} + \mathbf{n}$$



TX ant num

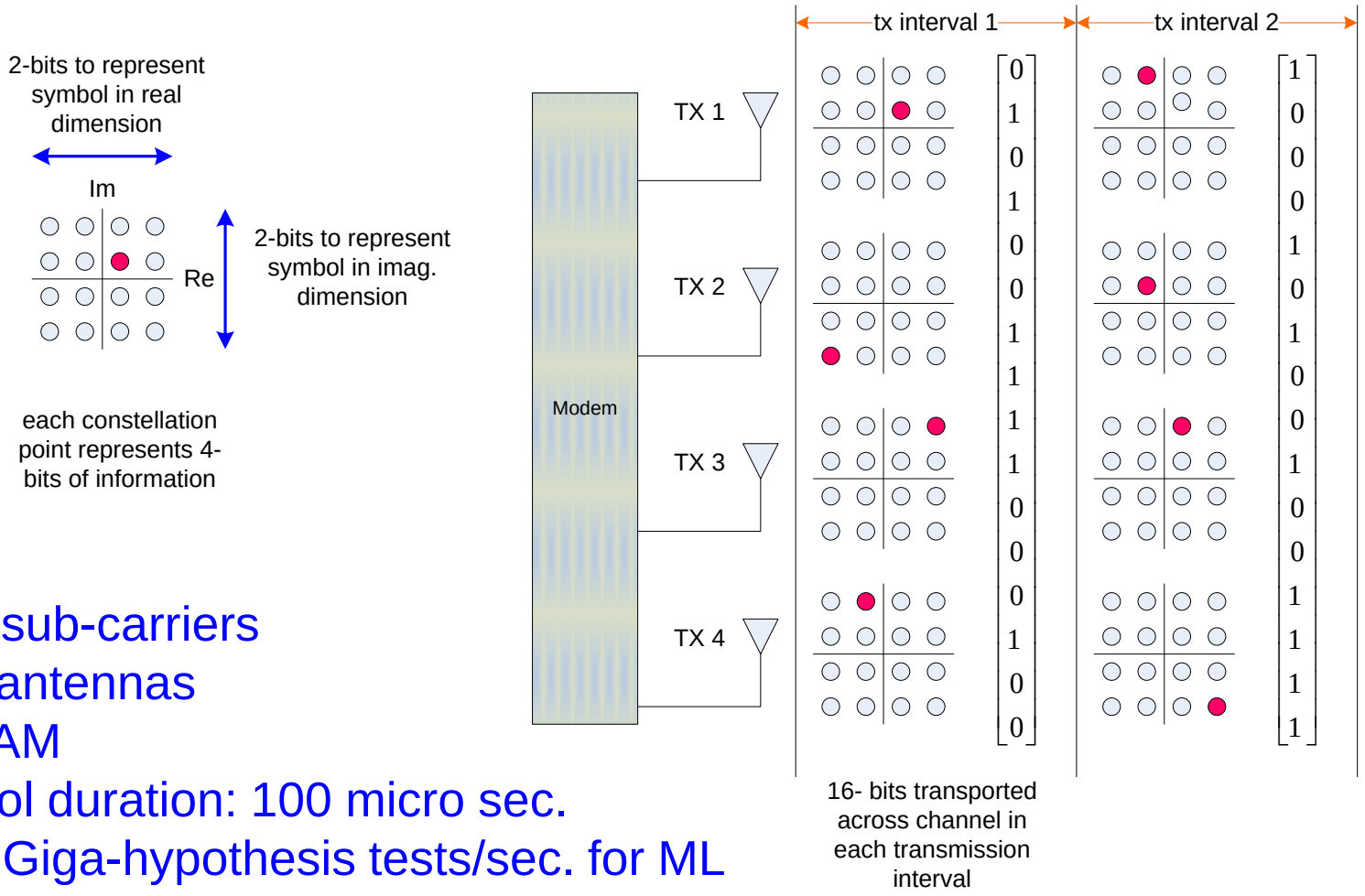
$$\mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

RX ant num



XILINX

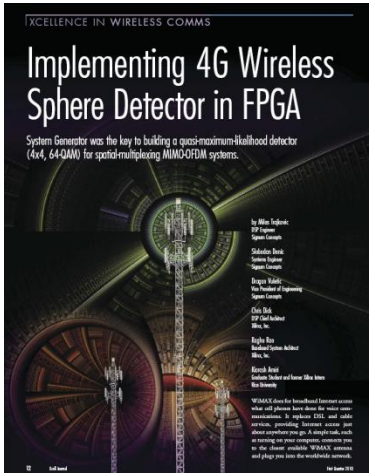
SM MLD MIMO Decoder Compute Requirements



- 1000 sub-carriers
- 4 TX antennas
- 16-QAM
- symbol duration: 100 micro sec.
- ~655 Giga-hypothesis tests/sec. for ML

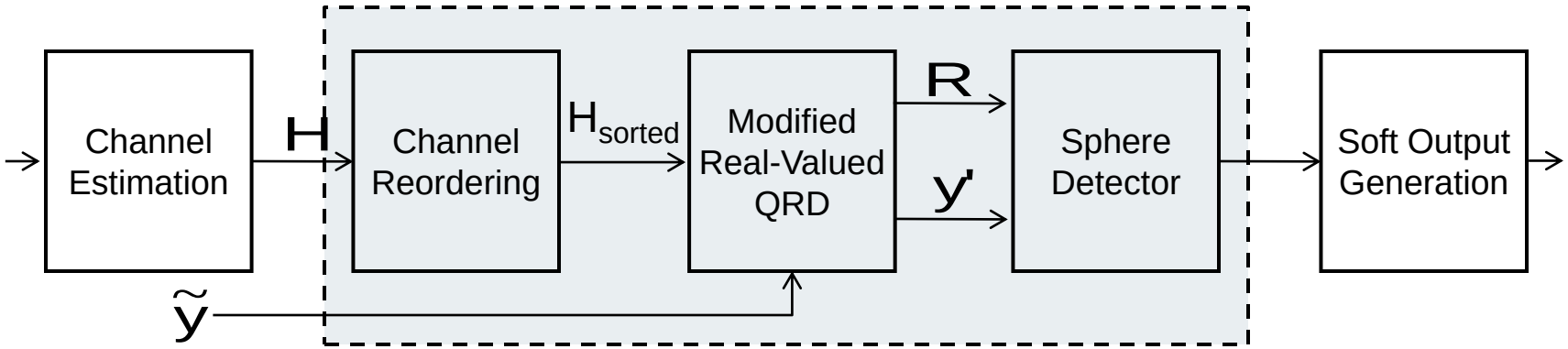
802.16e Sphere Detector Implementation

- **Specifications**
  - 5MHz WiMAX
  - 360 data sub-carriers every 102 usec
  - Challenge: minimize communication latency in addition to area
- **RTL Reference Implementation**
  - Target clock frequency was 225MHz in Virtex5-2
  - Implemented in System Generator
  - Reference Matlab
- **HLS implementation**
  - Same BER performance
  - Target same design point as reference implementation
  - ISE 12.2, AutoPilot 2010.07



http://www.xilinx.com/publications/archives/xcell/Xcell70.pdf

802.16e Sphere Detector Top-Level Block Diagram



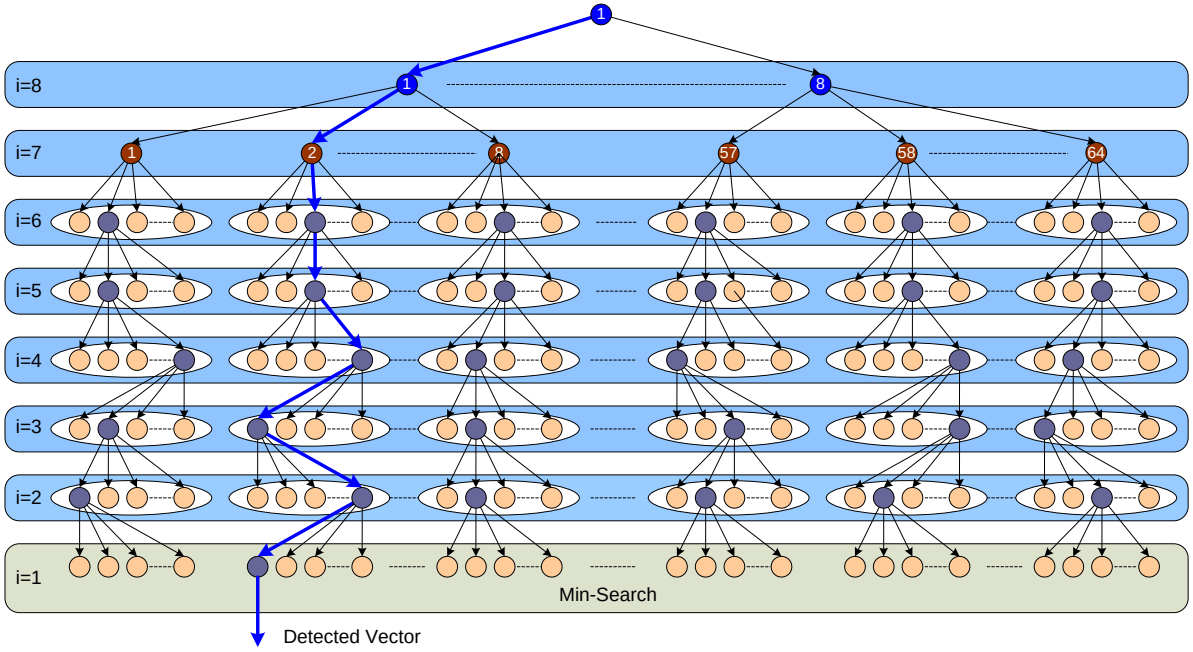
- **Key to achieving good BER performance is around how the antenna ordering is performed**
  - Maximum Likelihood detection $\hat{s} = \arg\min_{s \in \Lambda^{M_T}} \|\mathbf{y} - \mathbf{Hs}\|^2$ $\mathbf{H} = \mathbf{QR}$
- **System model $\mathbf{y} = \mathbf{Hs} + \mathbf{n}$ can be written as**

$$\begin{pmatrix} \Re(\mathbf{y}) \\ \Im(\mathbf{y}) \end{pmatrix} = \begin{pmatrix} \Re(\mathbf{H}) & -\Im(\mathbf{H}) \\ \Im(\mathbf{H}) & \Re(\mathbf{H}) \end{pmatrix} \cdot \begin{pmatrix} \Re(\mathbf{s}) \\ \Im(\mathbf{s}) \end{pmatrix} + \begin{pmatrix} \Re(\mathbf{n}) \\ \Im(\mathbf{n}) \end{pmatrix}$$

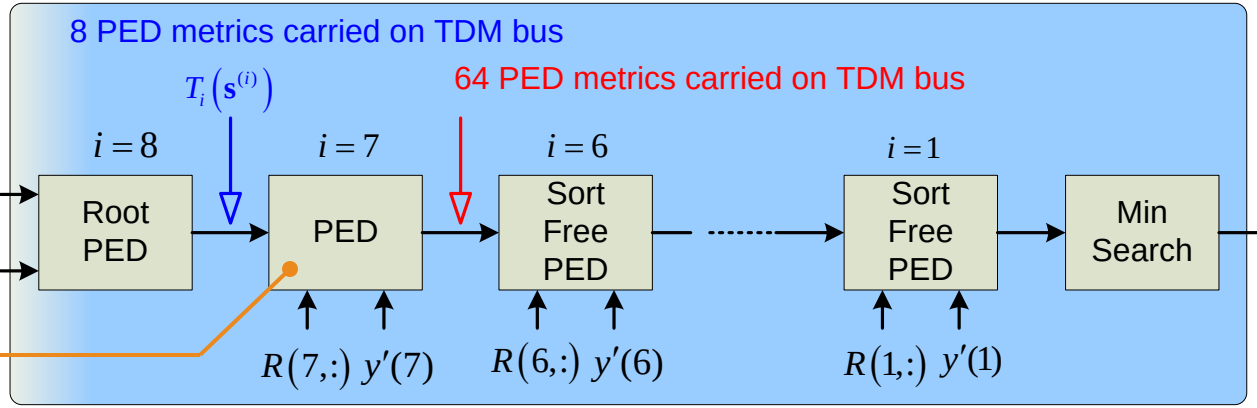
where each $s_i \in \mathbf{s}$ can take a value from the set $\Omega = \{\pm 7, \pm 5, \pm 3, \pm 1\}$ for 64 QAM

802.16e Sphere Detector Tree Search

- 64 cycles to process each matrix @ 225MHz
- Reference design:
  - Fully pipelined
  - Streaming
  - No bubbles



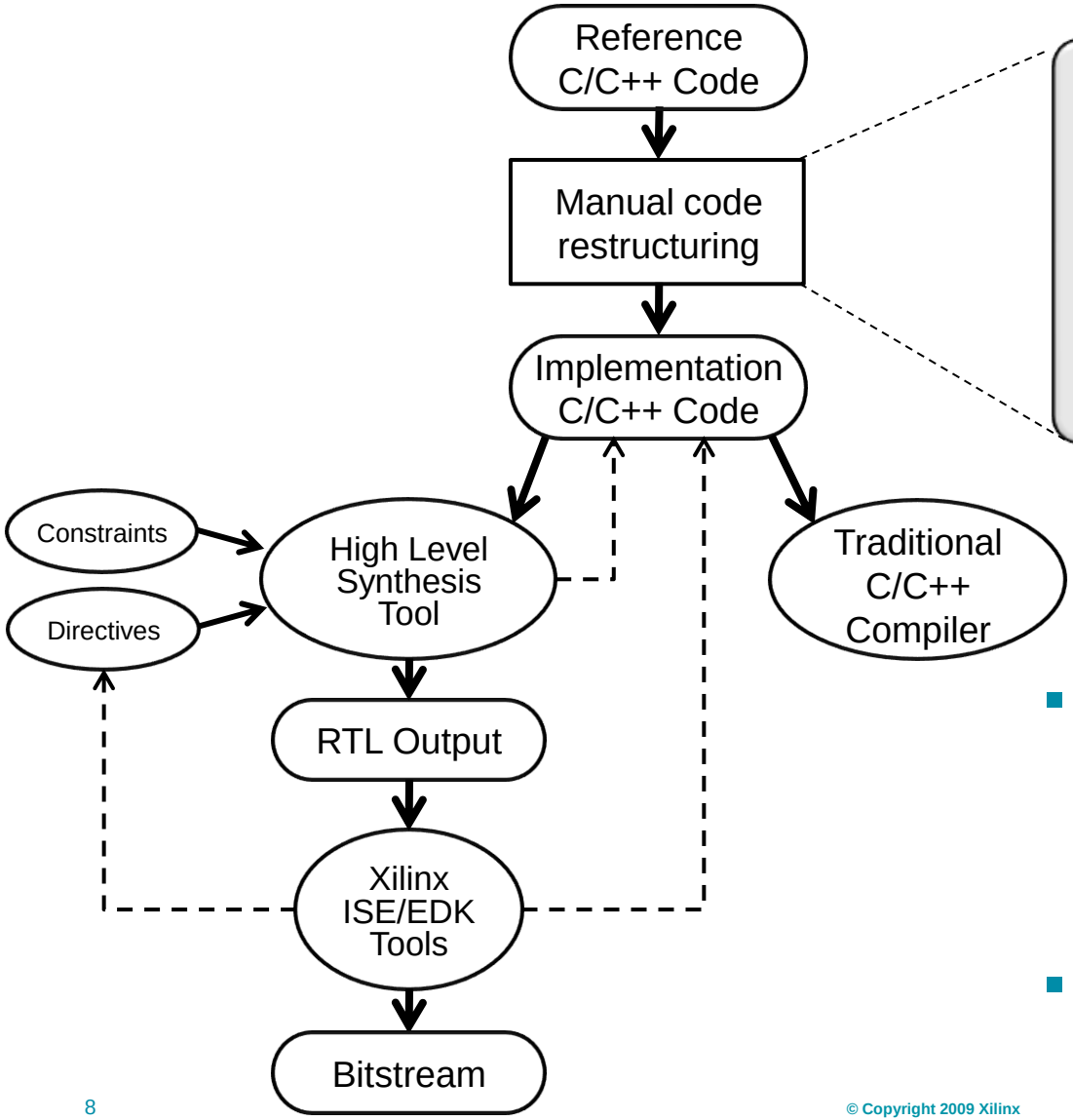
Sphere Detector Processing Pipeline

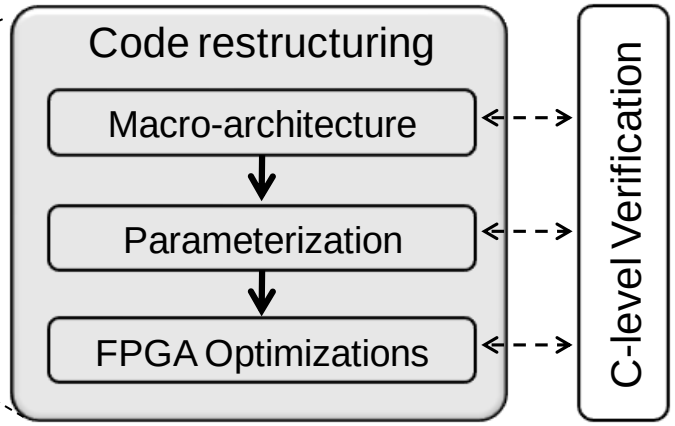


$$T_i\left(\mathbf{s}^{(i)}\right) = T_{i+1}\left(\mathbf{s}^{(i+1)}\right) + \left|e_i\left(\mathbf{s}^{(i)}\right)\right|^2$$

$$b_{i+1}\left(\mathbf{s}^{(i+1)}\right) = \hat{y}_i - \sum_{j=i+1}^{M_T} R_{ij} s_j$$

High-Level Synthesis for FPGAs





- **Enable DSP programmers to easily target FPGAs**
  - Support C-based algorithm development
- **Deliver Productivity Increase for RTL Designers**

Code Restructuring: Marcro-architecture

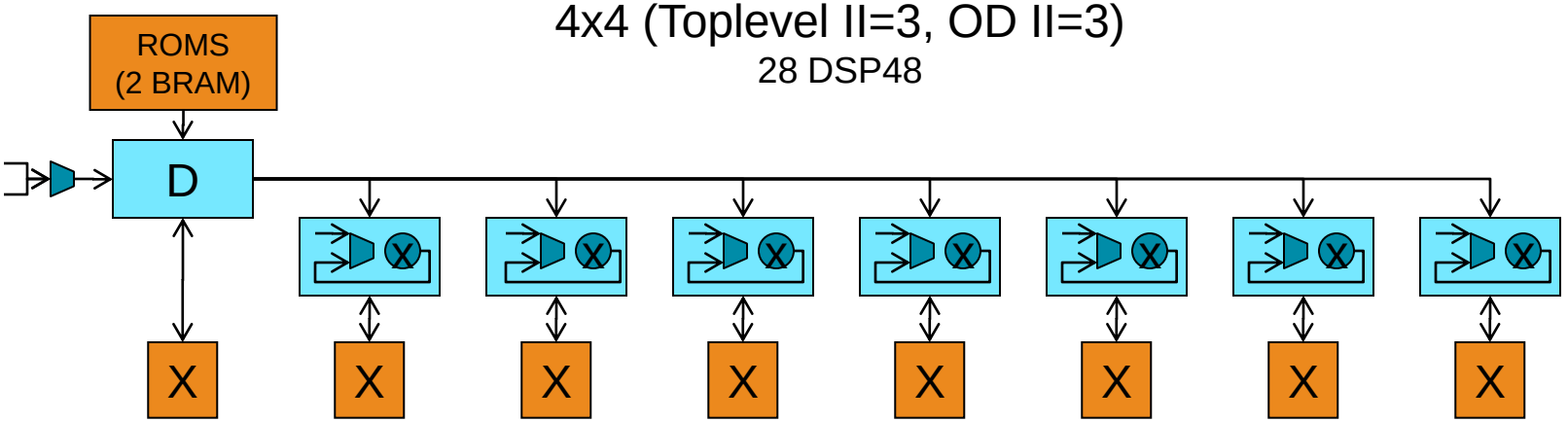
```
1: void sphere_detector_top (…) {
2:   #pragma AP DATAFLOW
3:   // PED streams between pipeline blocks
4:   ap_int<18>    PED_7[RVD_MODULATION];
5:   #pragma AP ARRAY_STREAM variable=PED_7 depth=1 stream
6:   ap_int<4>     symb_7[RVD_MODULATION];
7:   #pragma AP ARRAY_STREAM variable=symb_7 depth=1 stream
8:   main_label:{
9:     RootPED(r_7, y_7, ..., symb_7);
10:    PED(r_6, y_6, ..., symb_7, symb_6);
11:    SortFreePED<5,2>(r_5, y_5, ..., symb_6, symb_5);
12:    SortFreePED<4,3>(r_4, y_4, ..., symb_5, symb_4);
13:    SortFreePED<3,4>(r_3, y_3, ..., symb_4, symb_3);
14:    SortFreePED<2,5>(r_2, y_2, ..., symb_3, symb_2);
15:    SortFreePED<1,6>(r_1, y_1, ..., symb_2, symb_1);
16:    SortFreePED<0,7>(r_0, y_0, ..., symb_1, symb_0);
17:    // find minimum PED
18:    min_finder(PED_0, symb_0, min_PED, min_symb_list);
19:  }
20:}
```

Code Restructuring: Parameterization

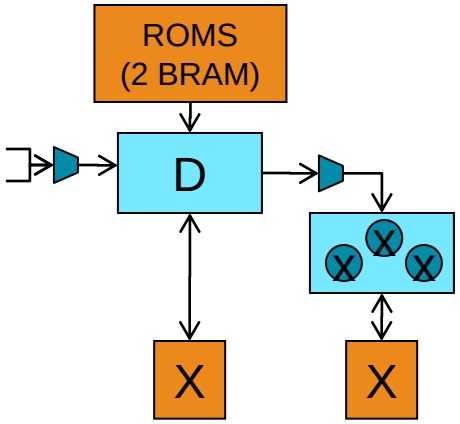
```
1:  template<int X_DIMENSION, int Y_DIMENSION, int MM_II>
2:  void matrix_multiply_(...) {
3:  #pragma AP ARRAY_PARTITION variable=chunk_in_re dim=1
4:  #pragma AP ARRAY_PARTITION variable=chunk_in_im dim=1
5:   // matrix multiplication of a A'*A matrix
6:   for (index_a = 0; index_a < TDM_CHUNKS; index_a++) {
7:     for (index_b = 0; index_b < X_DIMENSION; index_b++) {
8:       for (index_c = 0; index_c < Y_DIMENSION; index_c++) {
9:  #pragma AP PIPELINE II = MM_II
10:           <loop body>
11:   } } }
12: }
```

Code Restructuring: Parameterization

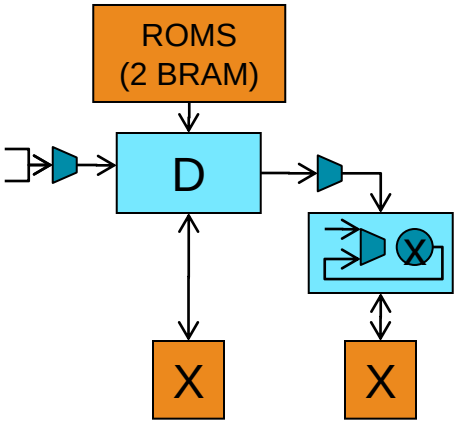
4x4 (Toplevel II=3, OD II=3)
28 DSP48



3x3 (Toplevel II=5, OD II=1)
12 DSP48

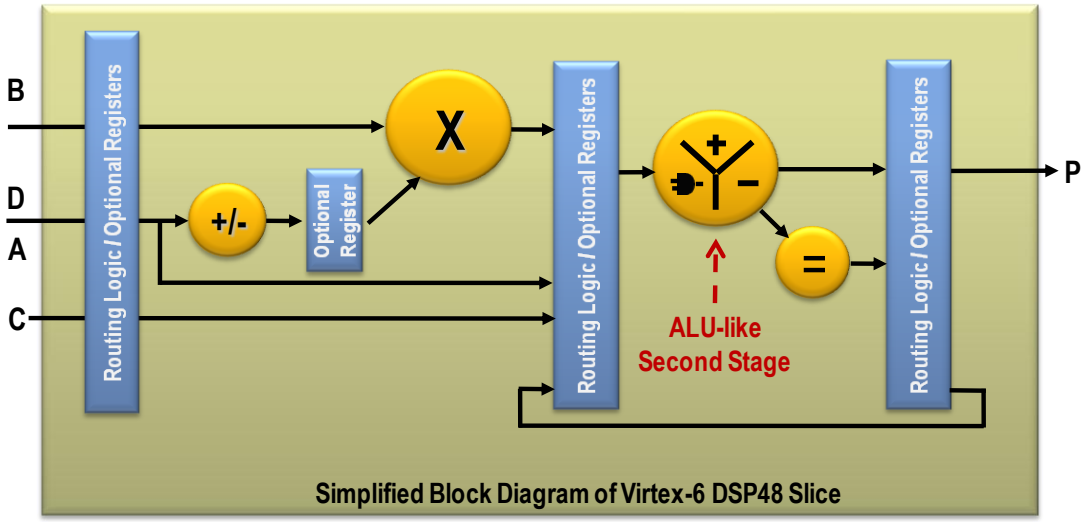


2x2 (Toplevel II=9, OD II=3)
4 DSP48



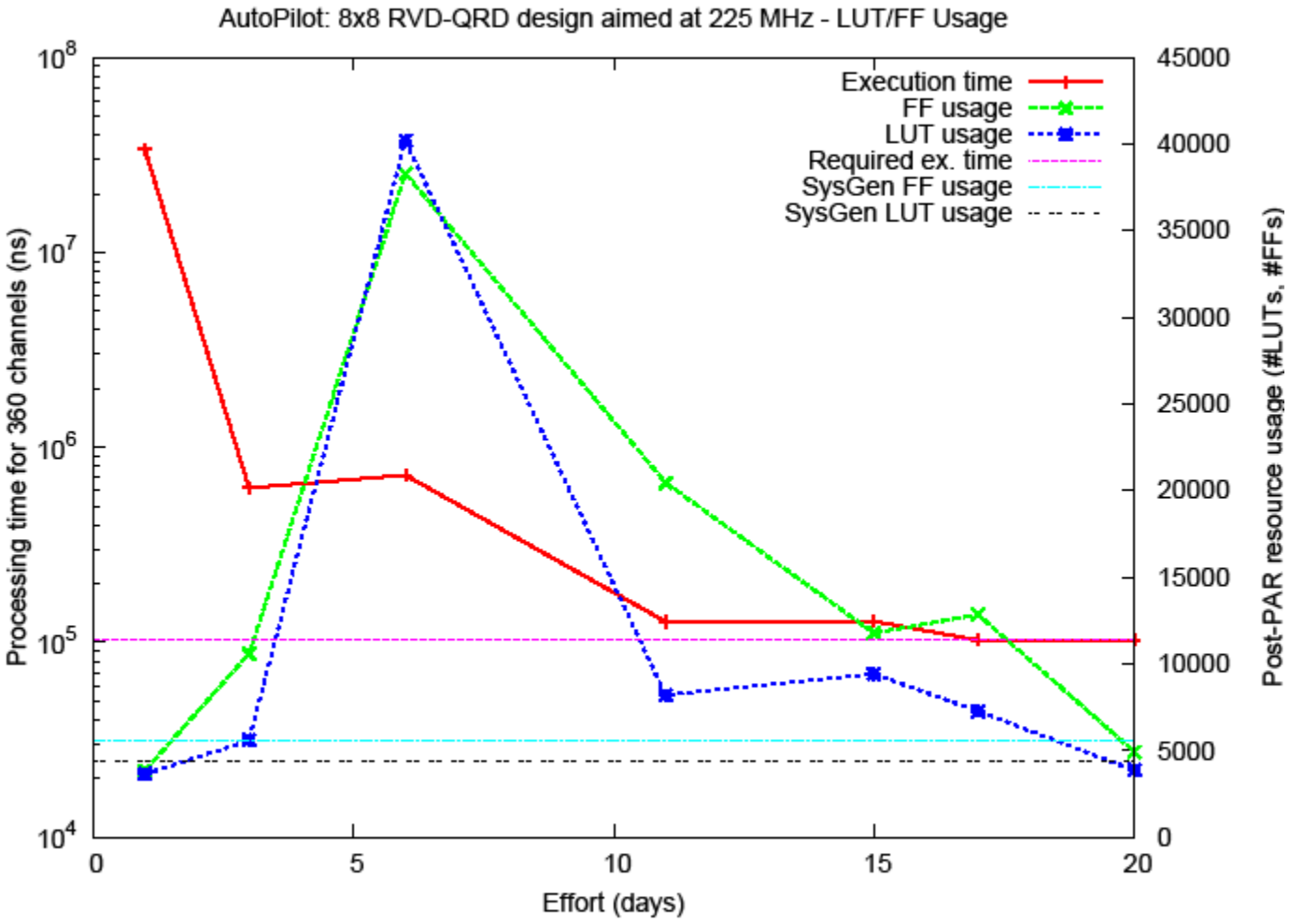
Code Restructuring: FPGA Optimizations

```
1:  template <int Wa, int Wb, int Wc>
2:  ap_int<36> SUBMUL(ap_int<Wa> a, ap_int<Wb> b, ap_int<Wc> c) {
3:    #pragma AP LATENCY max=2
4:    #pragma AP INTERFACE ap_none port=return register
5:
6:    ap_int<36> c_36 = c; // sign extension
7:    return c_36-a*b;
8:  }
```



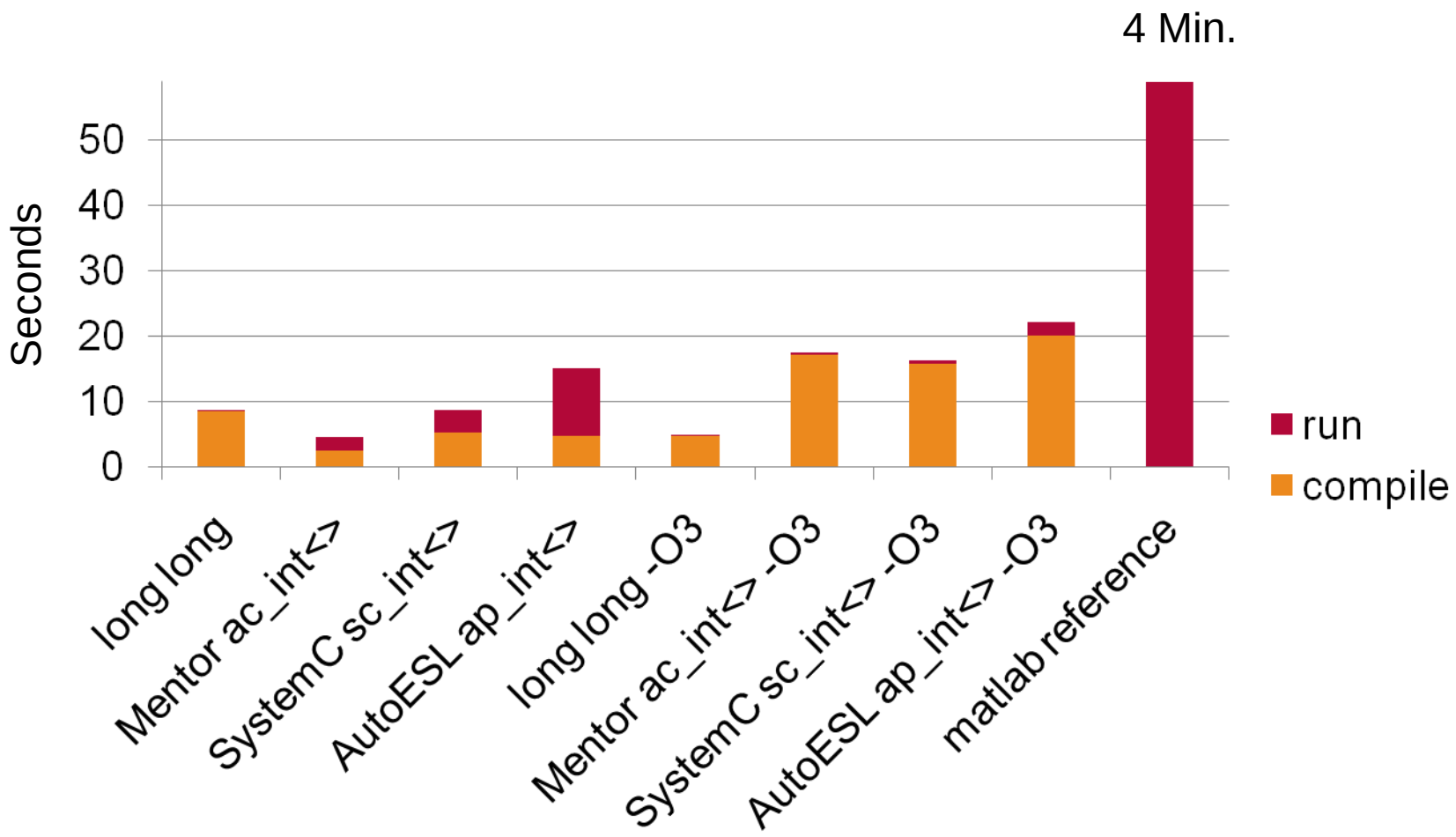
Simplified Block Diagram of Virtex-6 DSP48 Slice

Time to result



XILINX

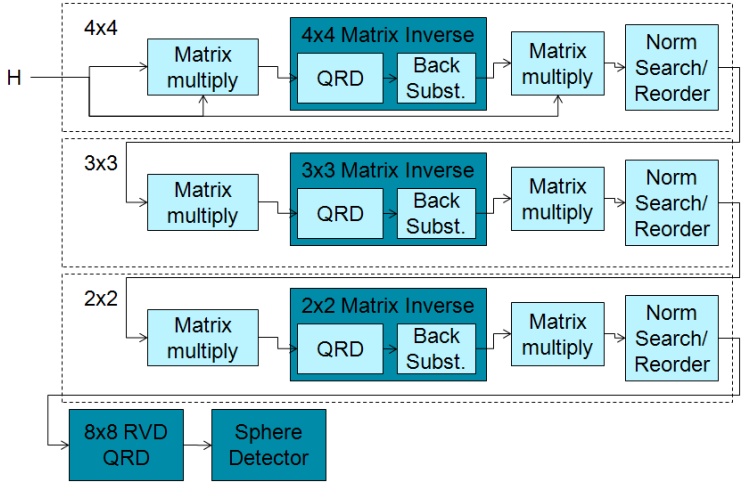
Simulation times
(4x4 Matrix inversion, 3600 subcarriers)



XILINX

FPGA Implementation Results

- **Compared Development Time and Results for Xilinx Expert user using SysGen and AutoESL**
  - Xilinx wireless apps team using System Generator
- **Development time for AutoESL includes learning the tool, producing results, design space exploration and detailed verification**



Metric	SysGen	AutoESL – Expert Result	% Diff
Development Time	16.5	15*	-9%
LUTs	27,870	29,060	+4%
Registers	42,035	31, 000	-26%
DSP48 slices	237	201	-15%
18K Brams	138	99	-28%
Design Iteration Time	3 hours, 45 min	22 min	

Summary

- **Implementation of a complex wireless MIMO receiver using a high-level synthesis tool targeting a Xilinx FPGA**
- **High-level Synthesis Tool for FPGAs achieve:**
  - Significant abstractions from low-level FPGA implementation details
    - e.g., timing and pipeline design
  - Competitive Quality of Results (QoR) to RTL design
  - Fast and continuous C/C++ level verification
    - Contributes to the reduction in the overall development time
- **Obtaining excellent results for complex and challenging designs requires good macro-architecture definition and FPGA tools knowledge**

Thanks!

Juanjo Noguera

Xilinx Research Labs

juanjo.noguera@xilinx.com

+353 1 461 5556