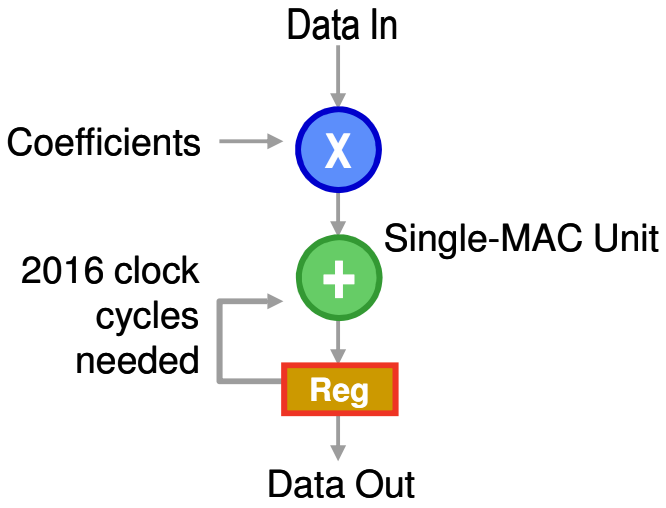
Zynq for Video Applications

Mike Mitchell
DSP Specialist,
Austin, TX

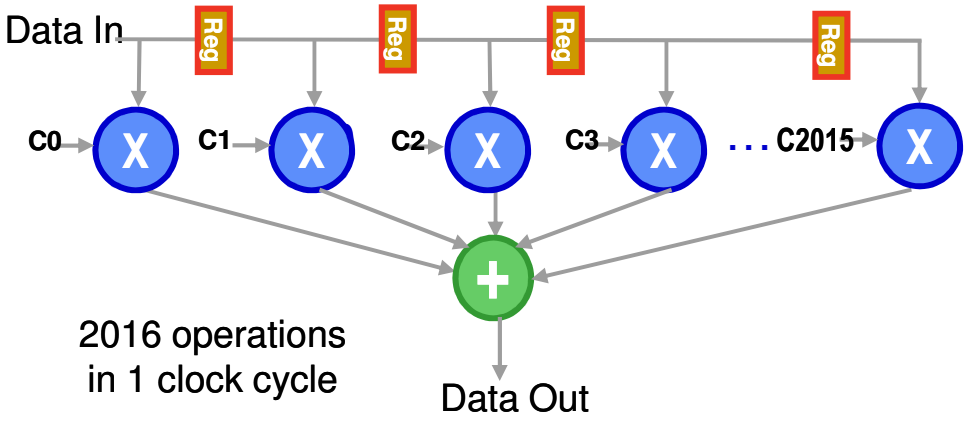
Serial vs. Parallel DSP Processing

Standard DSP Processor ALU – Sequential (Generic DSP)



$$\frac{1.2\ \text{GHz}}{2016\ \text{clock cycles}} = 595\ \text{KSPS}$$

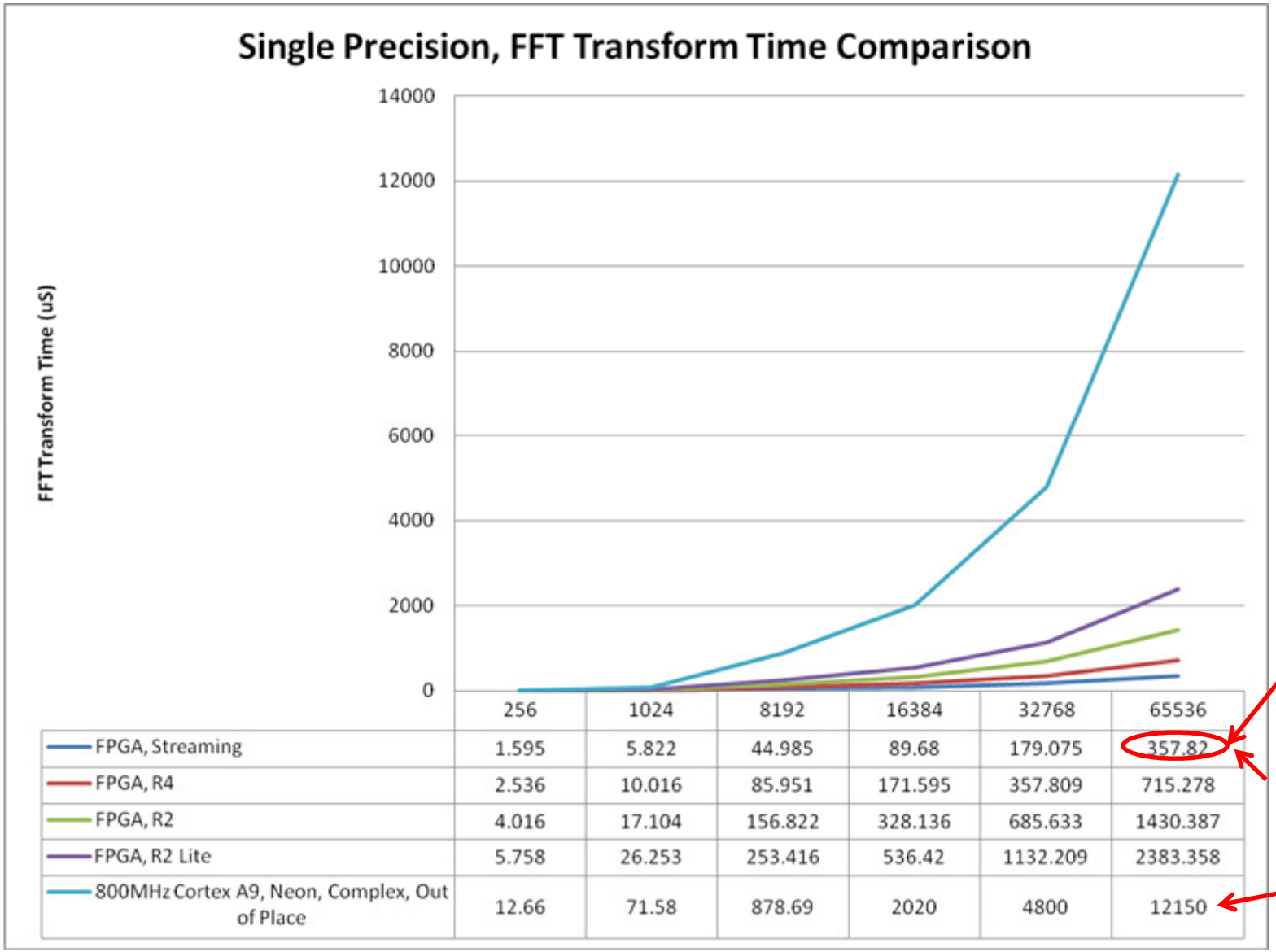
FPGA - Fully Parallel Implementation (Virtex-6 FPGA)



$$\frac{600\ \text{MHz}}{1\ \text{clock cycle}} = 600\ \text{MSPS}$$

demuxed filter architectures can enable even higher data rates

Software vs Hardware Acceleration



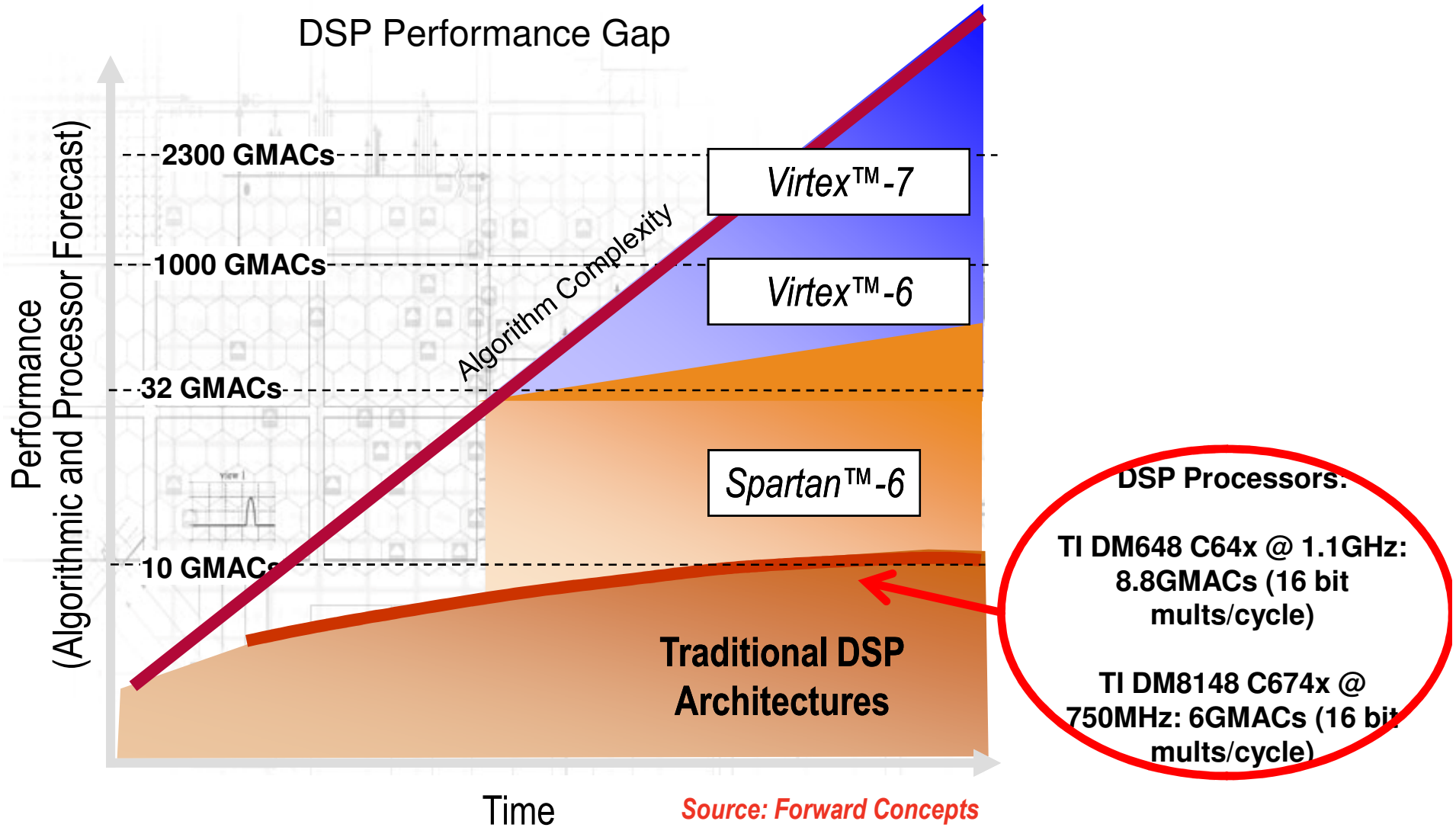
	256	1024	8192	16384	32768	65536
FPGA, Streaming	1.595	5.822	44.985	89.68	179.075	357.82
FPGA, R4	2.536	10.016	85.951	171.595	357.809	715.278
FPGA, R2	4.016	17.104	156.822	328.136	685.633	1430.387
FPGA, R2 Lite	5.758	26.253	253.416	536.42	1132.209	2383.358
800MHz Cortex A9, Neon, Complex, Out of Place	12.66	71.58	878.69	2020	4800	12150

Different FFT architectures can push performance even higher, consider sub 5us 2D FFT (GSPS)

550MSPS rate
After pipeline is filled

5.4MSPS rate

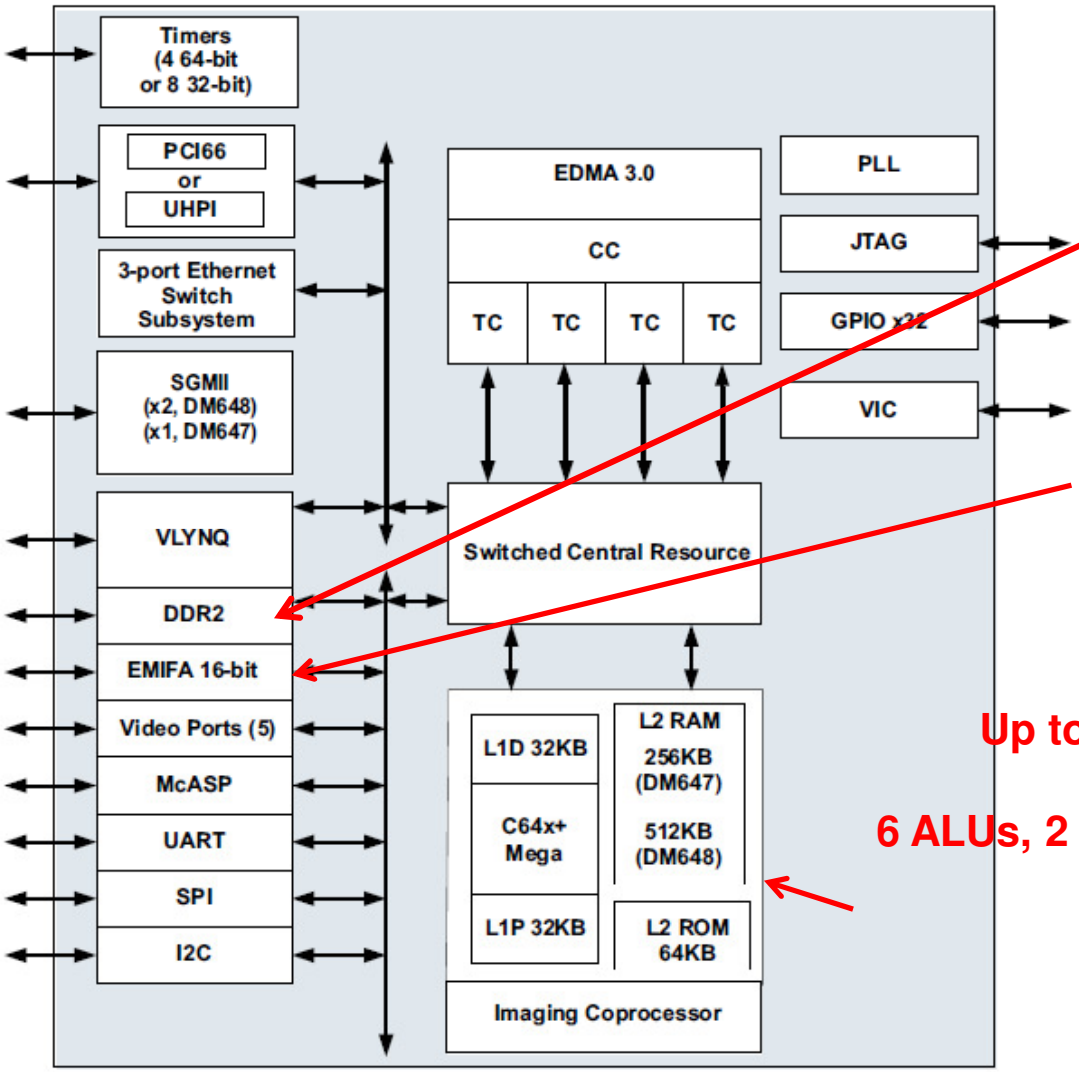
Xilinx DSP Performance Leadership



Source: Forward Concepts

Comparing TI DM648 Functionality and Zynq

TI TMS320DM648



512MB address space
DDR2 (32 bit)

EMIFA 16 bit interface
46.875MHz -167.67MHz
Table 6.42 pg 112
tms320dm648.pdf

Up to 1,1GHz C64x clock rates
8 MIPs/MHz
6 ALUs, 2 Multipliers (32, 16 or 8 bit ops)
8.8GMACs

32KB I/D L1 Cache
512KB L2
64KB ROM

90nm process technology

TI DM648 vs Zynq

- ***TI TMS320DM648***
- **90nm process**
- **Up to 1.1GHz C64x**
  - 8.0 DMIPS/MHz max (based on clock frequency)
  - 32KB I/D cache
  - 512KB L2 cache
  - 64KB ROM
  - 8.8GMACs max (based on 1.1GHz clock frequency)

- ***Zynq***
- **28nm process**
- **Dual, 667-800MHz Cortex A9 + NEON coprocessor**
  - 2.5 DMIPs/MHz & 2.88 CoreMark/MHz
  - 32KB I/D cache
  - 512KB L2 cache
  - 256KB on chip memory
  - NEON has FP (2MFLOPS/MHz) + fixed point SIMD capability per core
  - 7020 device has 220 DSP48s (628MHz -3 speed grade) for 138.1GMACs of performance
  - Speed grades -1 (464MHz), -2 (550), -3 (628)

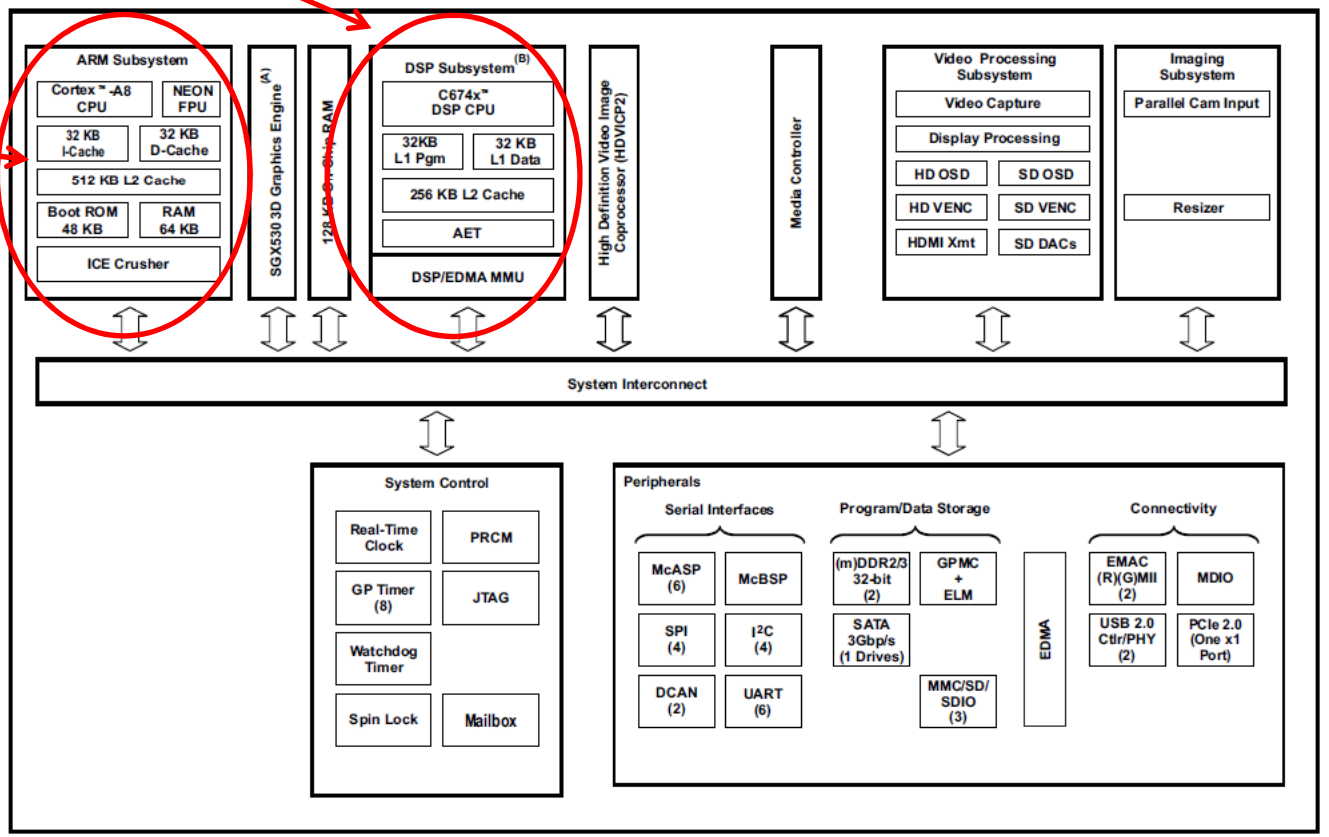
Comparing TI DM8148 Functionality and Zynq

TI TMS320DM8148

Up to 750MHz C674x
32KB L1, 256KB L2
6 ALUs, 2 Multipliers
(32, 16 or 8 bit ops)
8.8GMACs,
6 MFLOPs/MHz

Cortex A8
32KB I/D L1 Cache
512KB L2 Cache
64KB RAM

45nm process technology

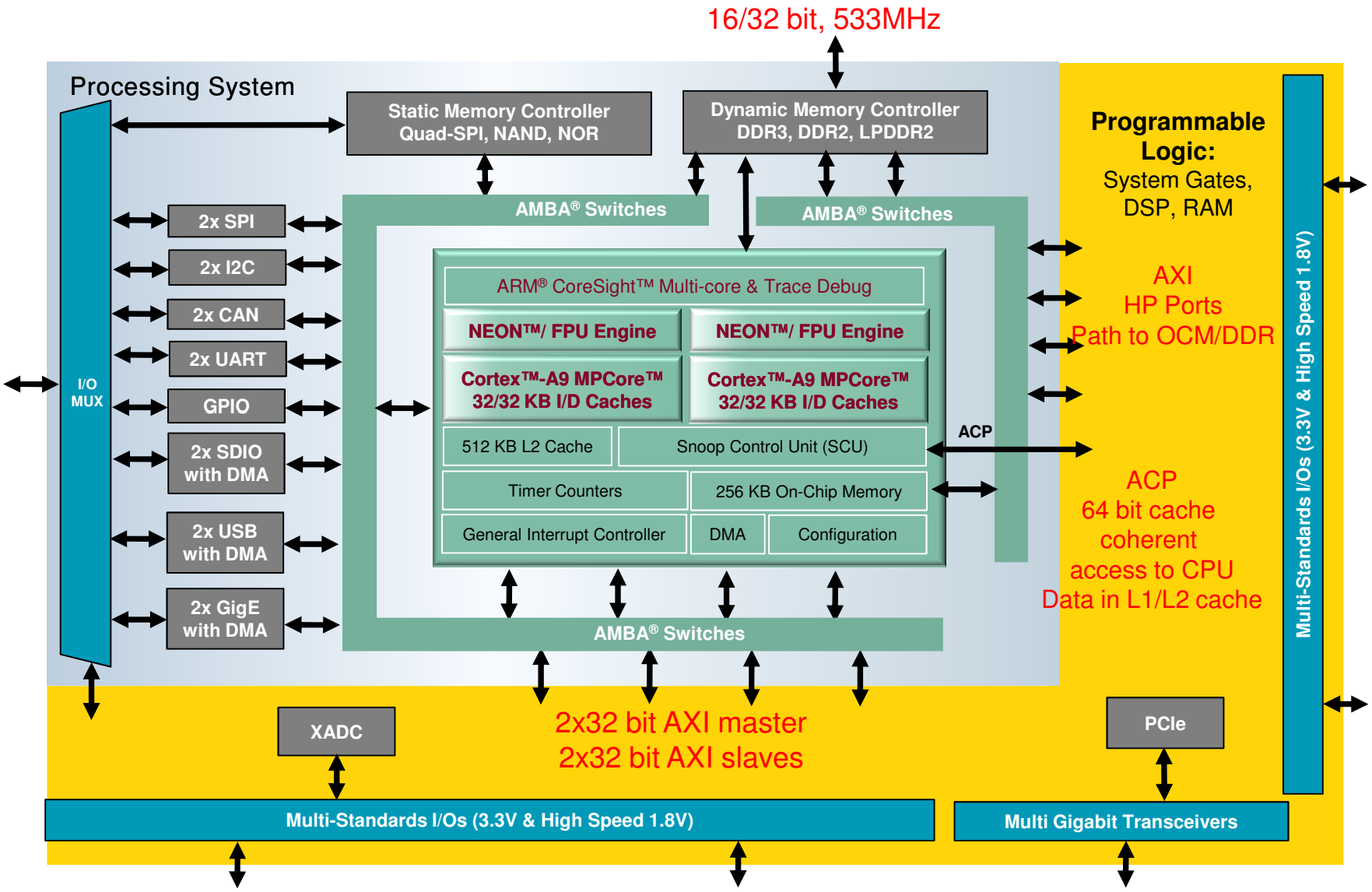


TI DM8148 vs Zynq

- ***TI TMS320DM648***
- **45nm process**
- **Up to 1GHz Cortex A8 + NEON coprocessor**
  - 2.0 DMIPs/MHz
  - 32KB I/D cache
  - 512KB L2 cache
  - 64KB on chip memory
- **Up to 750MHz C674x**
  - 8.0 MMACs/MHz max (based on clock frequency)
  - 6MFLOPs/MHz (based on clock frequency)

- ***Zynq***
- **28nm process**
- **Dual, 667-800MHz Cortex A9 + NEON coprocessor**
  - 2.5 DMIPs/MHz
  - 32KB I/D cache
  - 512KB L2 cache
  - 256KB on chip memory
  - NEON has FP (2MFLOPS/MHz) + fixed point SIMD capability per core
  - 7020 device has 220 DSP48s (628MHz -3 speed grade) for 138.1GMACs of performance
  - Speed grades -1 (464MHz), -2 (550), -3 (628)

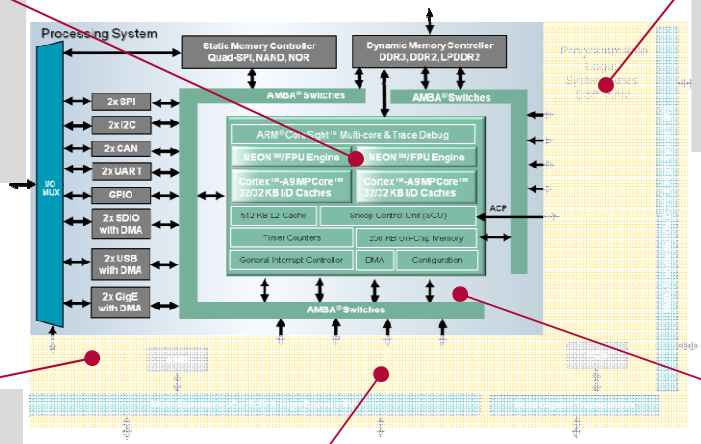
Zynq – A Full Custom, Single Chip ASSP



Zynq Targeted to Signal Processing Applications

Processor Core Complex

- Dual ARM® Cortex™-A9 MPCore™ with NEON™ extensions
- Single / Double Precision Floating Point support – 2 MFLOPS/MHz
- Up to 800 MHz operation



Over 3000 Internal Interconnects

- Up to 100Gb of BW
- Memory-mapped interfaces

Enables Massive Parallel Processing

- Up to 900 DSP blocks delivering over 1080 GMACs (symmetric FIR implementation)

Application specific custom hardware accelerator

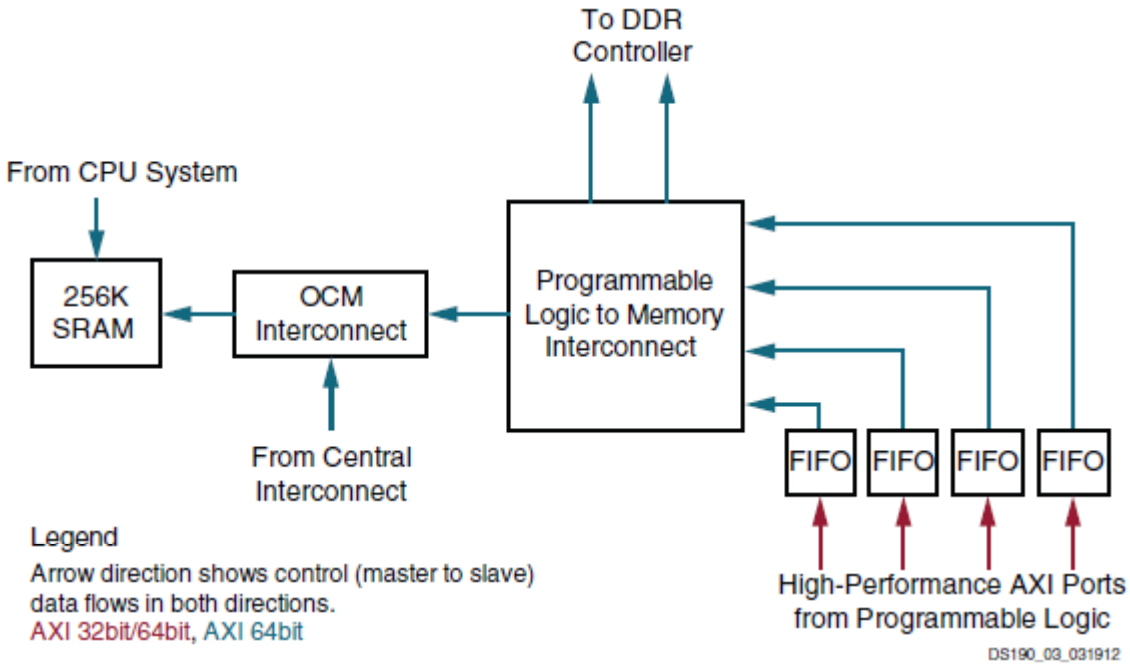
AMBA Open Standard Interconnnect

- High bandwidth interconnect between processing system and programmable logic
- ACP port for enhanced hardware acceleration and cache coherency for additional soft processors

Processing System Ready to Program

XILINX ALL PROGRAMMABLE™

Zynq High Performance (HP) Ports



PL Interface to PS Memory Subsystem

Each high-performance AXI port has these characteristics:

- Reduced latency between PL and processing system memory
- 1 KB deep FIFO
- Configurable either as 32- or 64-bit AXI interfaces
- Supports up to a 32 word buffer for read acceptance
- Supports data release control for write accesses to use AXI interconnect bandwidth more efficiently
- Supports multiple AXI commands issuing to DDR and OCM

SW vs HW Task Implementation Decisions

- **Keep it in Software**
  - Not in critical path
  - Enough “free” cycles
  - Easier to code in software than hardware
    - Use math library functions
  - NEON coprocessor
    - Supports integer vector operations
    - Single floating point operations
- **Move to Hardware**
  - Higher performance
    - Architecture determines area / throughput (ie: reference FFT page 3 & size vs latency discussion)
  - Customize to meet your needs & reduce processor bandwidth requirements
  - Excellent for iterative , pipelined & parallel processing
- **Move to another processor**
  - Both Microblaze & Picoblaze can exist in the fabric

TI OMAP35xx vs Zynq for H.264 CODEC

- http://www.iqmagazineonline.com/archive27/pdf/Pg32-37.pdf - h.264
- **Base CODEC is C code (ie: not optimized for NEON coprocessor or C64x VLIW).**
  - C code was also optimized for C64x VLIW core & CA8 NEON coprocessor.
  - Use of Cortex A8 + NEON shows better performance than the C64x VLIW core in an OMAP3530.
- **Demonstrates required processor clock rate compared to achieve the same VGA 30fps 1Mbps video streams for a Cortex A8 vs. OMAP3430**
  - lower clock rate MHz means better performance
- **Demonstrates NEON coprocessor is better than OMAP3530 C64x VLIW core.**

MPEG-4 SW Decoder

Version	OMAP 3430 (MHz)	Profiler (MHz)
Base	225,3	244,0
Optimized	137,6	110,3

Table 3: Measured performance for VGA 30fps 1Mbps MPEG-4 video stream decoding

MPEG-4 SW Encoder

Version	OMAP 3430 (MHz)	Profiler (MHz)
Base	1090,7	901,8
Optimized	488,3	369,0

Table 4: Measured performance for VGA 30fps 1Mbps MPEG-4 video stream encoding

Zynq NEON 3rd party ecosystem – Image, Audio, Video

ingenient technologies	H.264, VC1, MPEG-4
on2 technologies	VP6/7, MPEG-4, VC-1, H.264 (enc+dec), video stabilization
Ittiam	MPEG-4, MPEG-2, H.263, H.264, WMV9, VC1, DD+
ARICENT	MPEG-4, H.263, H.264, WMV9, audio
TATA ELXSI Engineering Creativity	H.264, VC1
SPIRIT DSP Embedded Voice Experience	TEAMSpirit voice & video
VisualOn	H.264, MPEG-4, H.263, WMV
ACTIMAGINE	MobiClip
Fraunhofer Institut Integrierte Schaltungen	Codecs
DOLBY	Multichannel audio processing
Adobe	Flash products

Algorithm Specialist TMC	MPEG-4
YAPPA	GUI visual efffects
drawElements Finland	2D GUI library
Espico Ltd	Audio: low-bitrate & digital theater, consulting
CORECODEC	CoreAVC ultra fast codec

ARM NEON widely supported by software partners

Full list on www.arm.com NEON ecosystem page

Xilinx IP Library for Camera Systems

Same IP supports video processing applications and displays

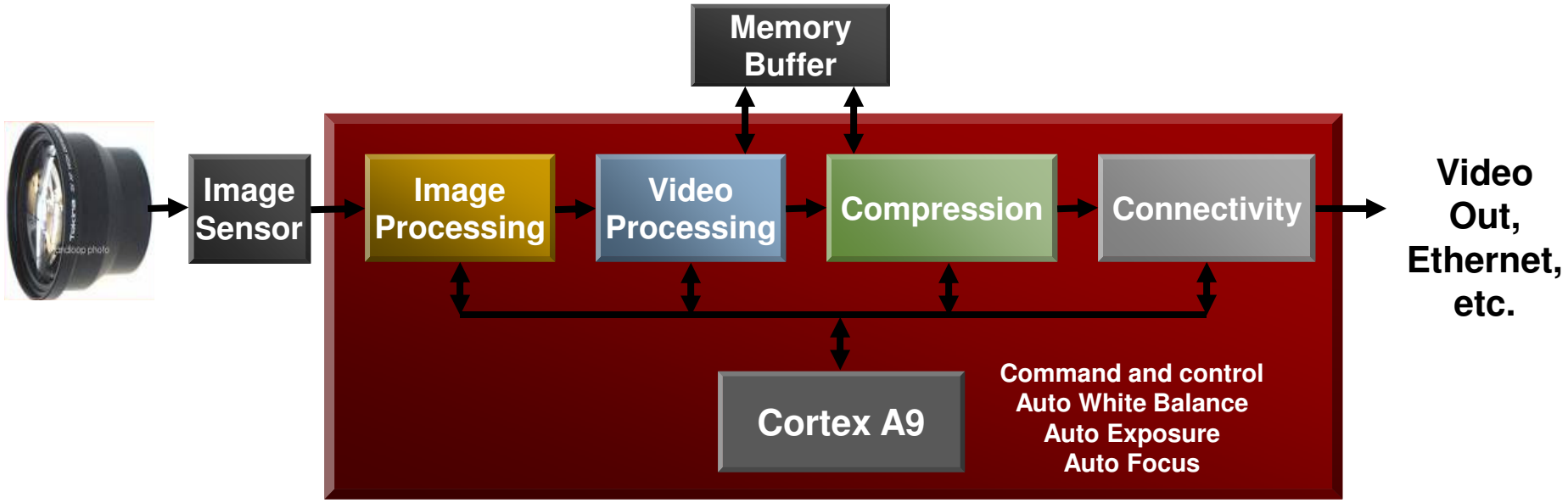


Image Processing
- Defective Pixel Correction
- Color Filter Array Interp
- Color Correction Matrix
- Gamma Correction
- Color Space conversion
- Statistics Module
- Noise Reduction
- Edge Enhancement

Video Processing
- Video Scaler
- On-Screen-Display
- Motion Adaptive Noise Reduction
- Image Characterization
- Object Segmentation

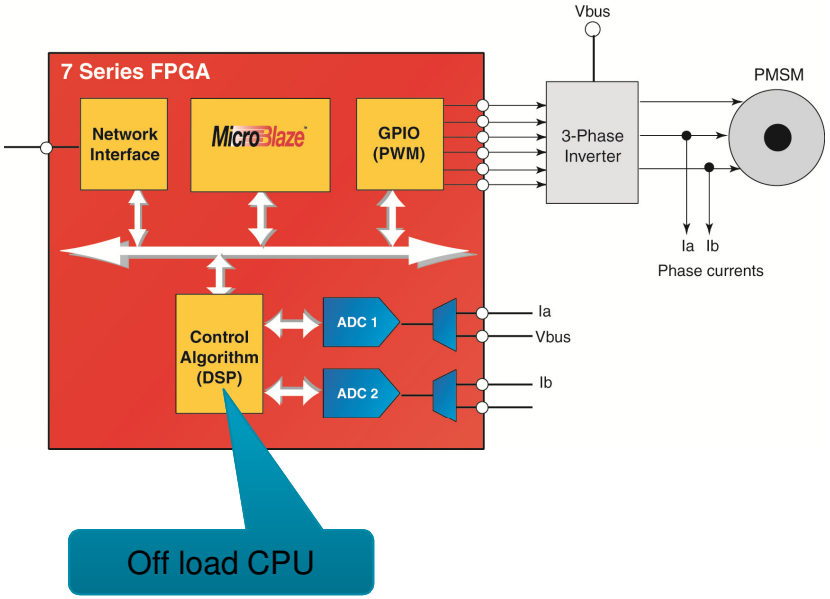
Memory and timing
- Timing Controller
- MPMC + VFBC
- Video DMA

Compression
- H.264/MPEG-4
- MPEG-2
- MPEG-4
- JPEG
- JPEG-2000

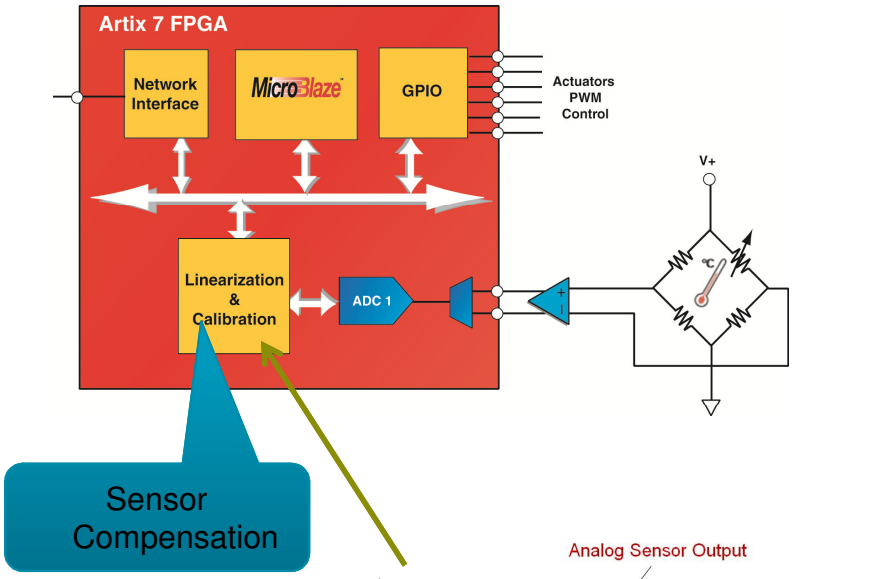
Connectivity
- 10/100/1G Ethernet
- SDI
- Ethernet AVB
- Display port
- Vx1, Vx4
- HDMI/DVI
- Firewire
- GigE Vision
- Camera Link

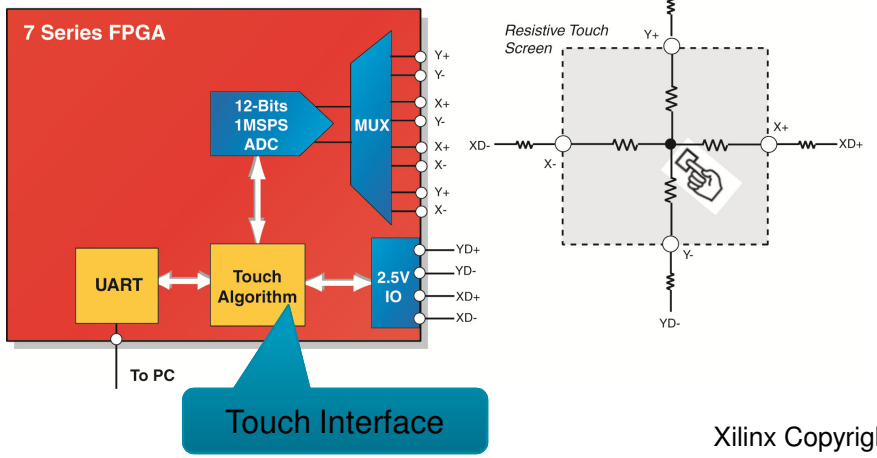
Some XADC Examples

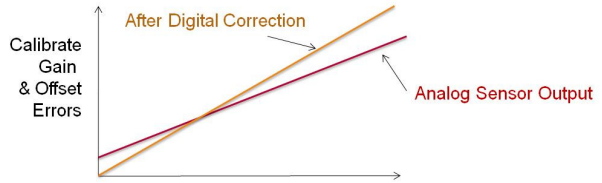
Motor Control / Energy Conversion



Sensor Interfacing

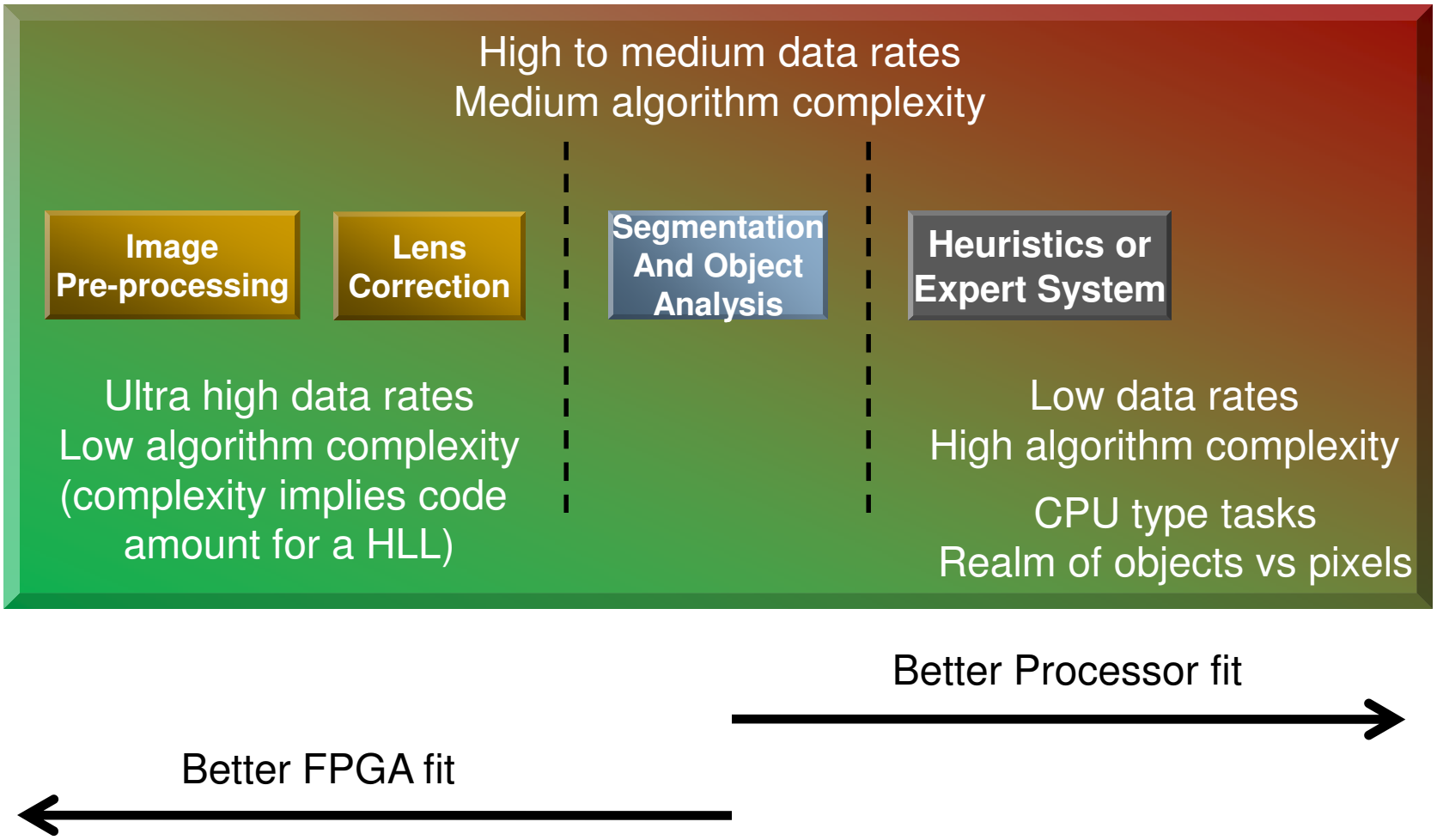






Video Applications

Computer Vision Processing Complexity



Computer Vision

- **Implementing Computer Vision Applications is challenging**
  - Limited engineering experience in building practical systems
  - Most solutions are ad hoc and highly application specific
  - Embedded systems are often highly constrained in cost, size, and power consumption
- **Computer Vision Is Computationally Intensive**
  - A 720p optical flow algorithm optimized for a C64x VLIW DSP architecture, consumed about 200MHz/frame/second for a throughput of 5 frames per second
- **Embedded Computer Vision is a Good Fit for FPGAs**
  - Highly custom designs based on application
  - High computational requirements, algorithms are diverse and dynamic, no off the shelf ASSPs to address

Computer Vision Hardware Design

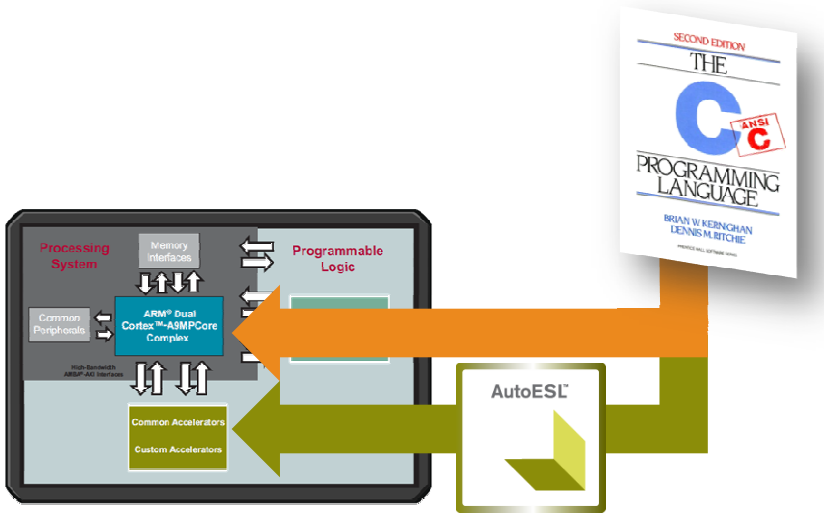
- Simulation modeling may be developed & tested using Matlab or C
- Converting the algorithmic portions (segmentation, object analysis, lens correction, image warping) into a hardware accelerator block may be challenging.
- Xilinx has options to simplify conversion of C or Matlab or Simulink designs into hardware accelerators.

Hardware/Software DSP Design Tradeoffs

- Hardware Accelerators can be exponentially faster than software and significantly reduce my code complexity and debug.
- Building Hardware Accelerators requires a hardware design methodology.
- I would like to write everything in software and then determine what needs to be accelerated which leads to…

Hardware/Software DSP Design Tradeoffs

- **The Dilemna:**
  - Which functionality should be in software vs hardware?

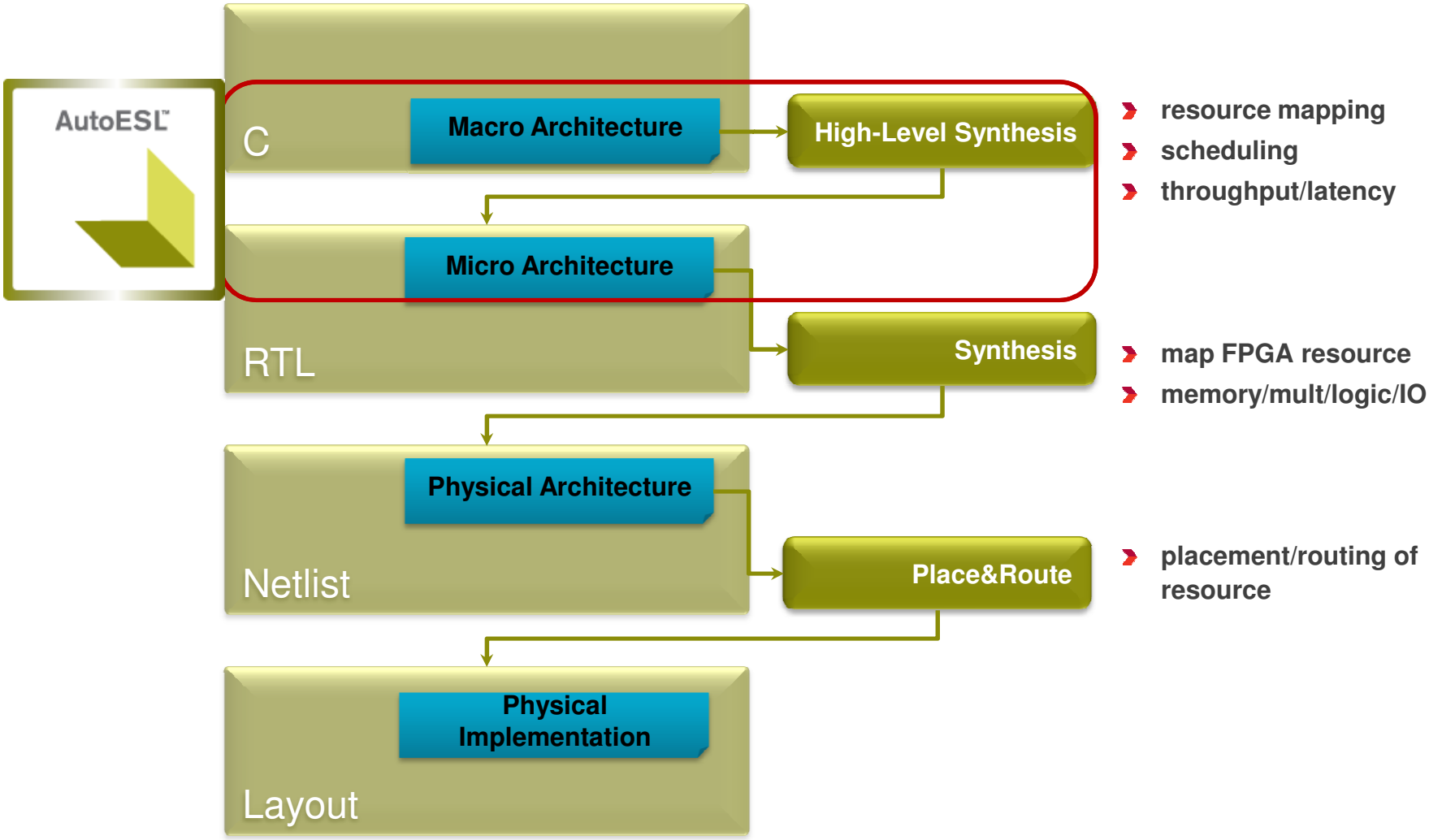


AutoESL Introduction

Agenda

- High-Level Synthesis
- AutoESL
- Design Examples
- Success Stories

High-Level Synthesis

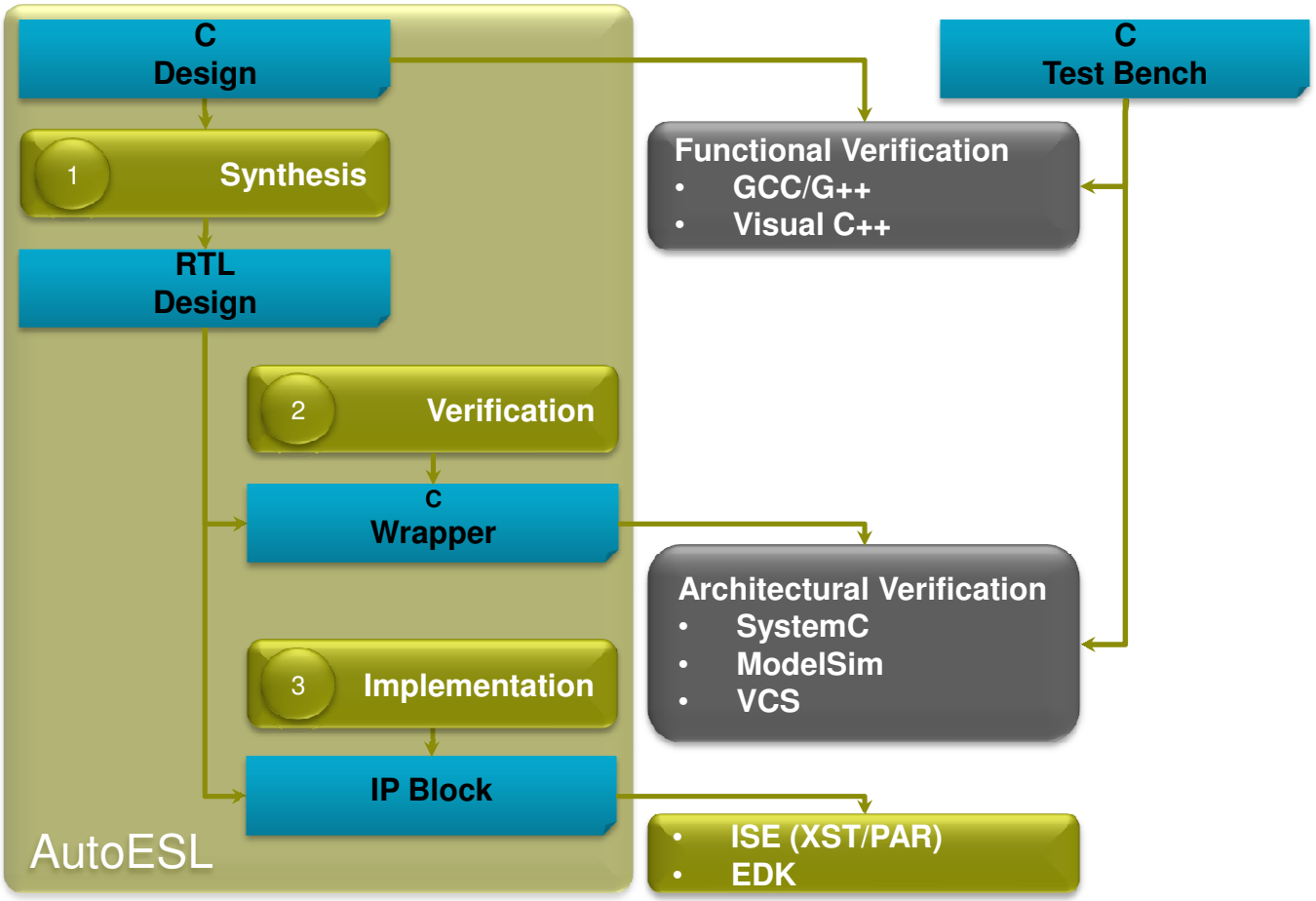


Agenda

- High-Level Synthesis
- AutoESL
- Design Examples
- Success Stories
- Roadmap

AutoESL: Block Creation Design Flow

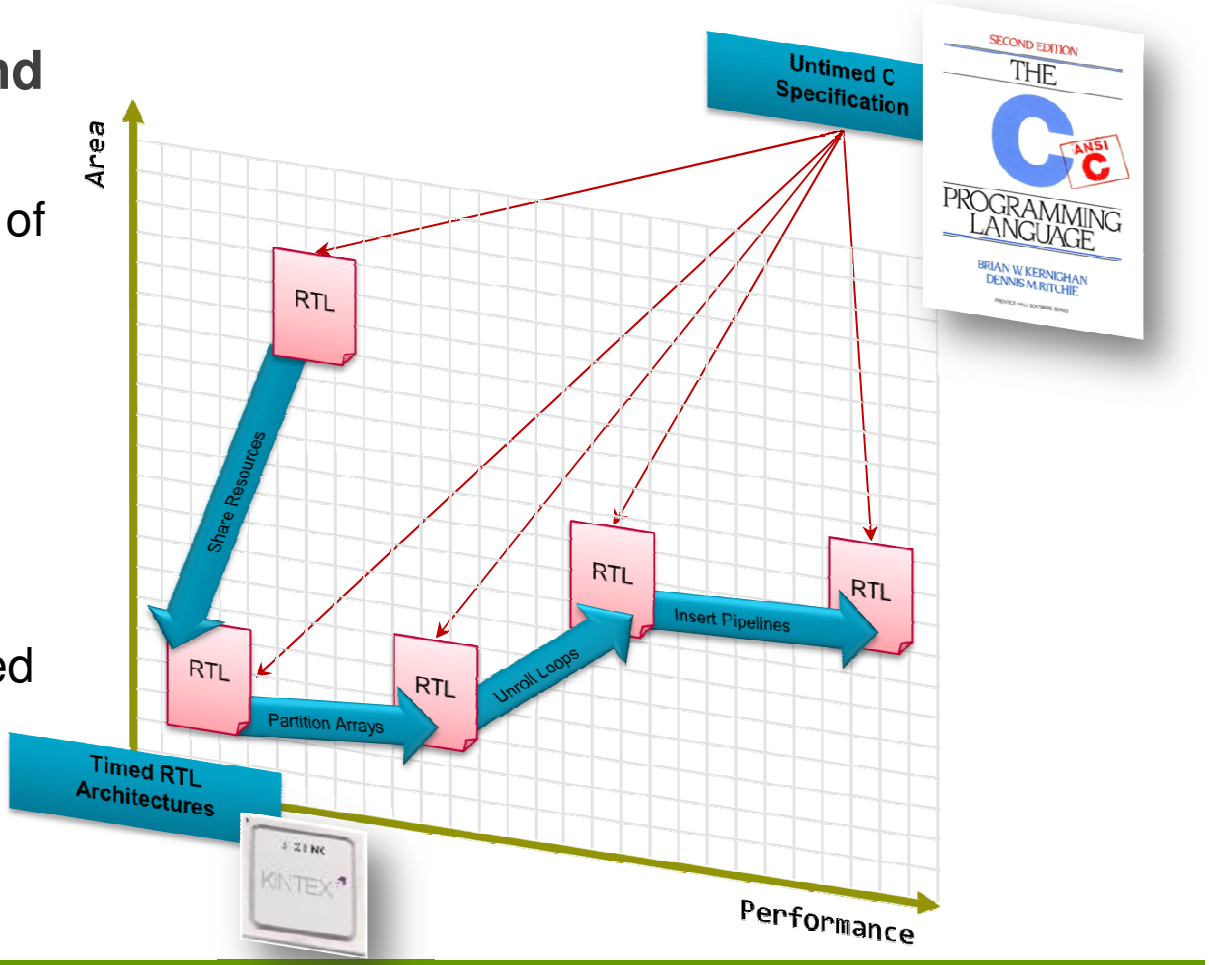
- Starts at C
  - C
  - C++
  - SystemC
- Produces RTL
  - Verilog
  - VHDL
  - SystemC
- Automates Flow
  - Verification
  - Implementation



Mapping C to FPGA

Benefits

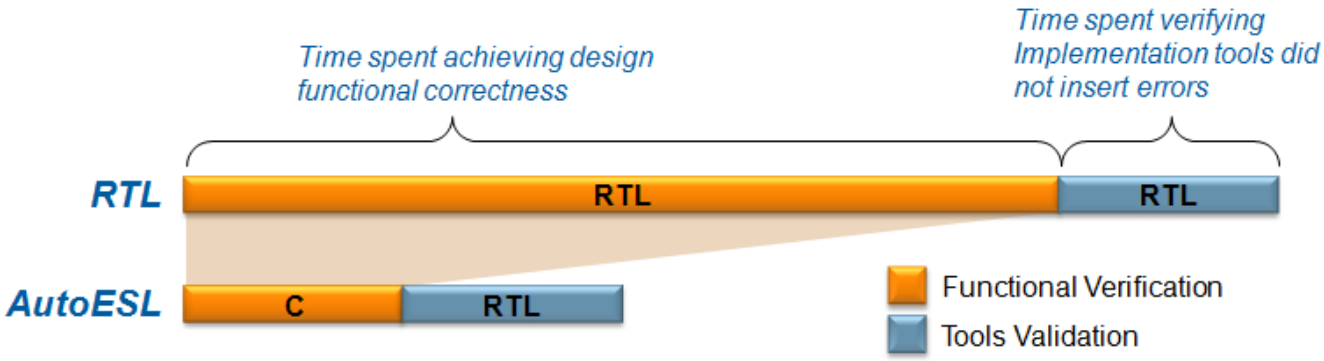
- **Automatic Scheduling and Resource Sharing**
  - Most time consuming part of FPGA design
  - Parallelization
  - Pipelining
- **QoR (Quality of Results)**
  - Results rivaling hand-coded RTL



Rapid design exploration

Benefits

- **More design iterations / day can be achieved by migrating functional verification to C/C++**
  - Orders of magnitude faster than RTL for large designs
  - RTL verification becomes final check



Optical flow Video Example

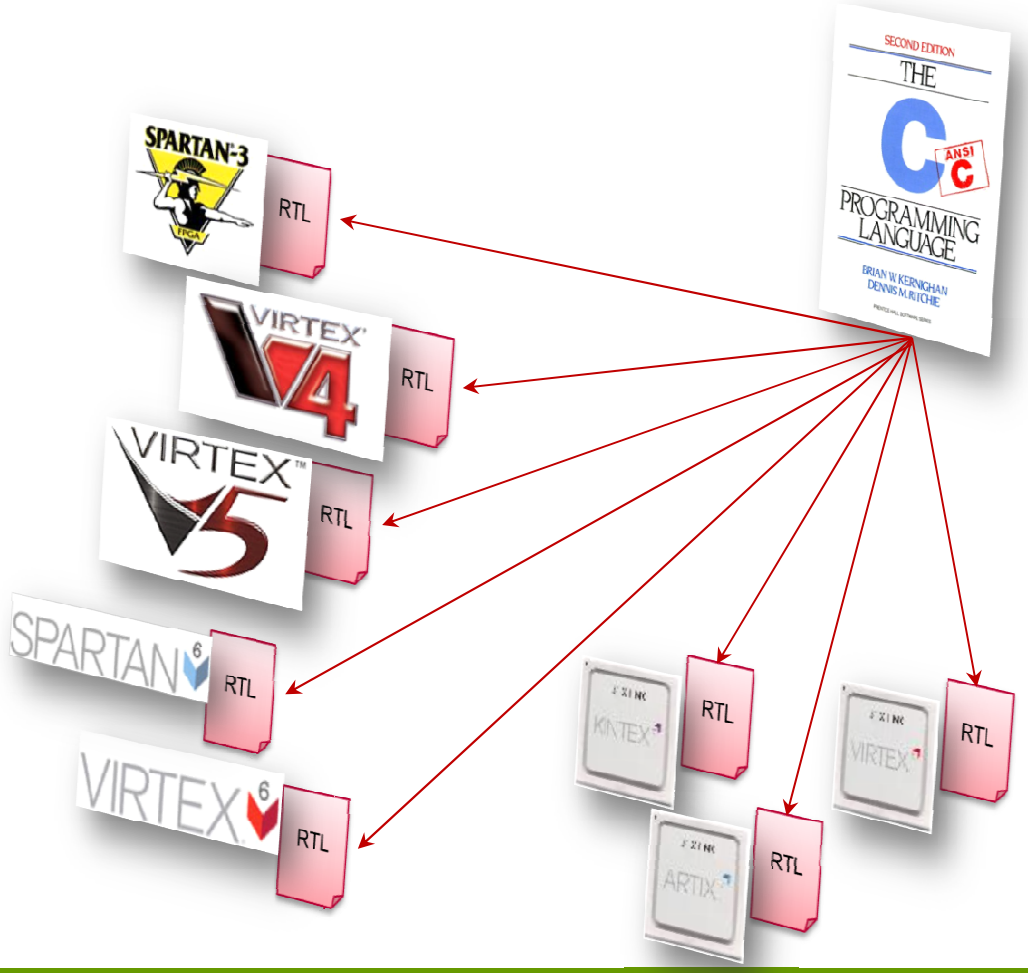
Input	C Simulation Time	RTL Simulation Time	Improvement
10 frames of video data	10 seconds	~2 days*	~12,000X

**RTL Simulations performed using ModelSim*

Block level verification *significantly* reduced

Benefits

- **Portability/Design Reuse**
  - Technology migration
  - Cost reduction
  - Full Resource/ Performance/Power analysis



Design and IP reuse

Benefits

- **Floating Point Support**
  - Use ‘float’ and ‘double’ C/C++ data types
  - Use standard math operators (+, -, *, /)
    - Additional float functions may require C/ C++ declarations
      - e.g. for single-precision square root

```
#include <math.h>
extern “C” float sqrtf(float);
```

  - Using Xilinx floating-point core library reference

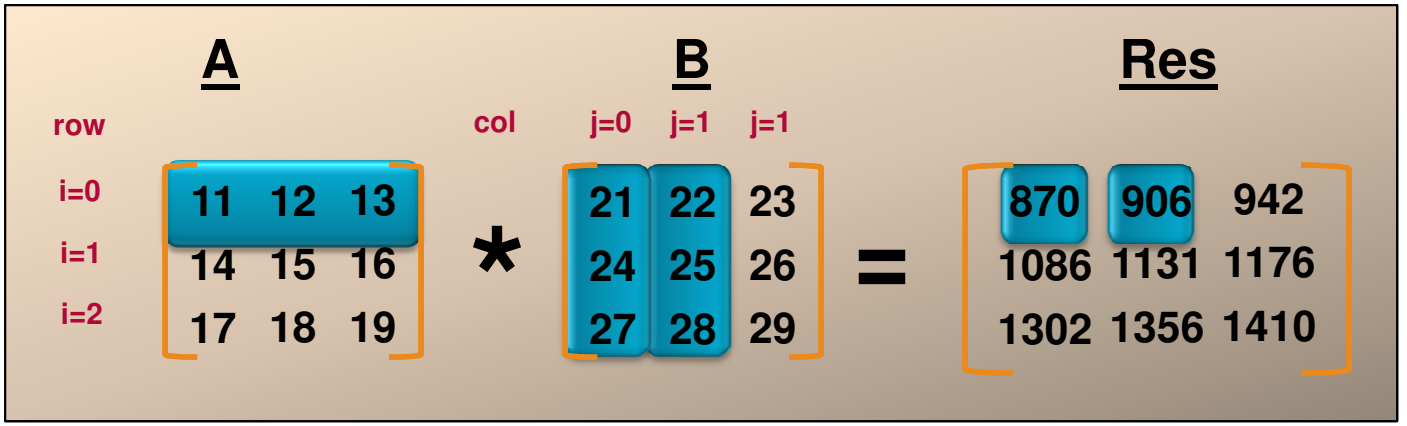
Floating Point Computation using AESL

Agenda

- High-Level Synthesis
- AutoESL
- Design Examples
- Success Stories

Example: Matrix Multiplication

- **Matrix Multiplication Example**
  - Multiply the rows of A with the columns of B and sum



  - Example in detail

$r_{00} = 11 * 21 + 12 * 24 + 13 * 27 = 870$

$r_{01} = 11 * 22 + 12 * 25 + 13 * 28 = 906$

Etc…

C Code

- **The C code for a matrix multiplication is fairly intuitive**
  - A series of nested loops

```
 // Iterate over the rows of the A matrix
Row: for(int i = 0; i < MAT_A_ROWS; i++) {
    // Iterate over the columns of the B matrix
    Col: for(int j = 0; j < MAT_B_COLS; j++) {
      // Do the inner product of a row of A and col of B
      res[i][j] = 0;
      Product: for(int k = 0; k < MAT_B_ROWS; k++) {
        res[i][j] += a[i][k] * b[k][j];
      }
    }
}
```

Algorithm easily parallelizable without changing code

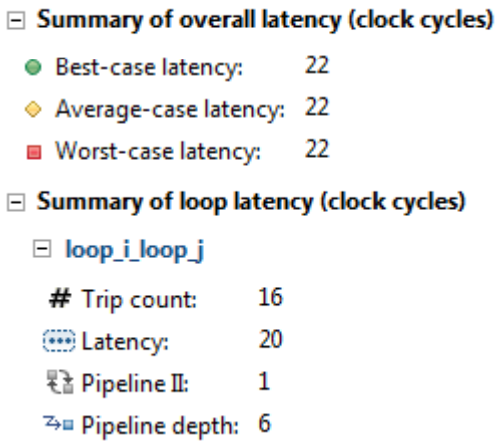
Matrix Multiply: Code

```
#include "mat_mult.h"

void mat_mult(complex_t c[M][M], complex_t a[M][M], complex_t b[M][M])
{
    int i,j,k;
    complex_t tmp, prod;


loop_i:
    for (i=0;i<M;i++) {

loop_j:
        for (j=0;j<M;j++) {
            tmp.r = 0;
            tmp.i = 0;

loop_k:
            for (k=0;k<M;k++) {


                //#pragma AP PIPELINE
                prod.r = a[i][k].r*b[k][j].r - a[i][k].i*b[k][j].i;
                prod.i = a[i][k].r*b[k][j].i + a[i][k].i*b[k][j].r;
                tmp.r += prod.r;
                tmp.i += prod.i;
            }

            c[i][j].r = tmp.r;
            c[i][j].i = tmp.i;

        }
    }
}
```

```
#ifndef mat_mult_H_
#define mat_mult_H_

// include required library headers
//#include <math.h>
//float sqrtf(float);

// Include AutoESL fixed point header files
#include "ap_int.h"
#include "ap_fixed.h"
//typedef ap_fixed<16,16, AP_RND_CONV, AP_SAT>  fx_data_t;
typedef ap_int<16> fx_data_t;

struct complex_t {
    fx_data_t r, i;
};

// Define the system required parameters
#define M   4   // dimension of square matrix

void mat_mult(complex_t c[M][M], complex_t a[M][M], complex_t b[M][M]);

#endif
```

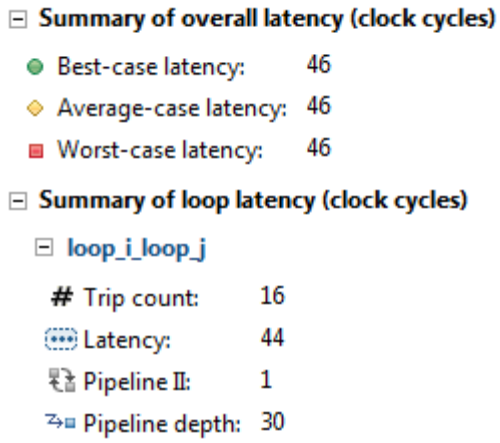
Matrix Multiply: Results (fixed pt)

- **Computes 4x4 matrix multiplication**
  - complex data sample of 16 bit each
  - one output element per clock cycle with 6 clk cycle latency
  - can be changed to produce all 16 elements in a single clk cycle at the cost of more IO and resources
  - 12M matrix mult operations per second
- **Target device: V6LX240 -2**
- **Timing : 5.2 ns (=192 MHz)**
- **Resource:**
  - 199 LUT, out of 150720
  - 391 FF, out of 301440
  - 18 DSP48, out of 768
  - 0 BRAMs, out of 416 of 36kbit
- **Can easily converted to floating point implementation**
  - by changing the variable types to float/double



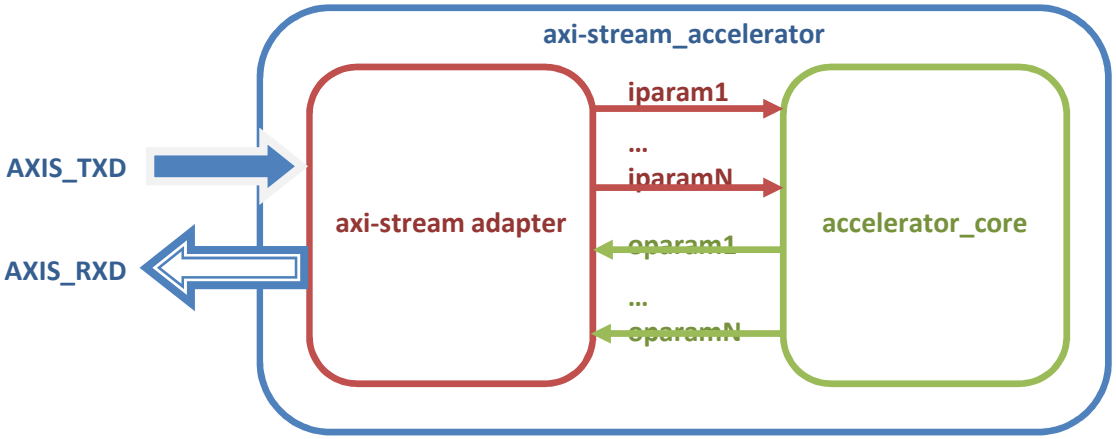
Matrix Multiply: Results (floating pt)

- **Computes 4x4 matrix multiplication**
  - complex data sample of single precision floating pt each
  - one output element per clock cycle with 30 clk cycle latency
  - 12M matrix mult operations per second
- **Target device: V6LX240 -2**
- **Timing : 5.3 ns (=189 MHz)**
- **Resource:**
  - 5612 LUT, out of 150720
  - 5486 FF, out of 301440
  - 76 DSP48, out of 768
  - 0 BRAMs, out of 416 of 36kbit



Floating Point Matrix Multiply Software vs. Hardware Acceleration

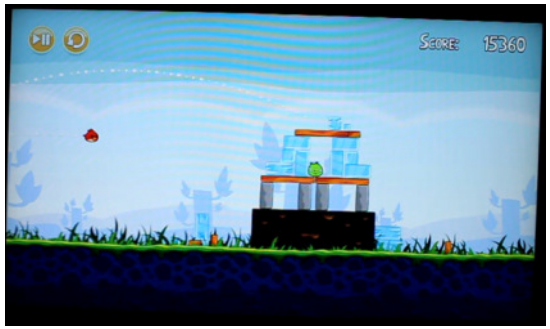
- **Multiplication of two 32x32 floating point Matrices**
  - 4.69x acceleration factor in a Zynq 7020
- **AXI based accelerator created using AutoESL, 166MHz Fabric clock**



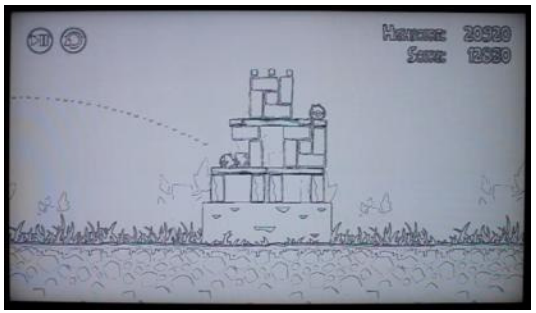
AutoESL hardware accelerator IP for Zynq

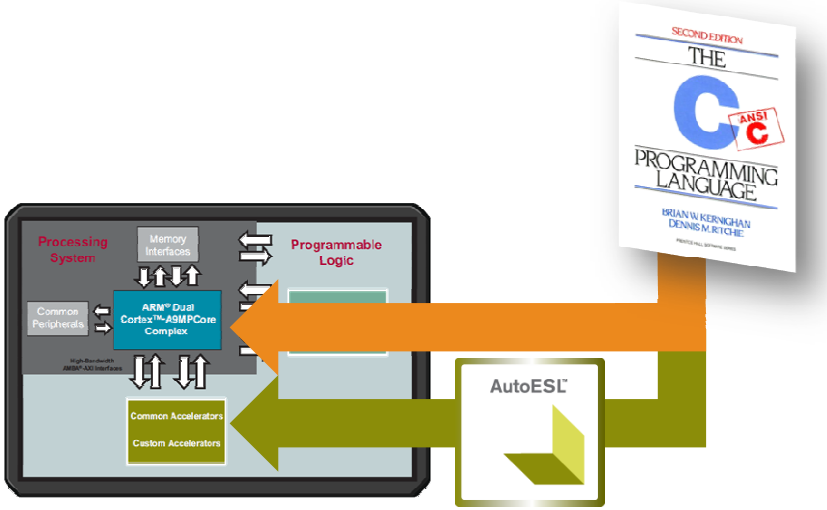
Zynq HD Video Processing Demo

1080p 60fps



Sobel Filter Output (Edge Detect)





Demonstrate Sobel filter Running in Software (CA9) & Hardware (via AutoESL)

Representative Processor Implementation

- Processor implementation uses arrays for input and output
- Full frame delay between input and output – all output pixels available to the next function in the pipeline at the same time
- 3x3 Sobel computation window is created by random access into input_image array

```
for(i = 0; i < height; i++){
    for(j=0; j < width; j++){
        x_dir = 0;
        y_dir = 0;
        if((i > 0) && (i < (height-1)) && (j > 0) && (j < (width-1))){
        for(rowOffset = -1; rowOffset <= 1; rowOffset++){
            for(colOffset = -1; colOffset <=1; colOffset++){
                x_dir = x_dir + input_image[i+rowOffset][j+colOffset] * Gx[1+rowOffset][1+colOffset];
                y_dir = y_dir + input_image[i+rowOffset][j+colOffset] * Gy[1+rowOffset][1+colOffset];
            }
        }
      edge_weight = ABS(x_dir) + ABS(y_dir);
      output_image[i][j] = edge_weight;
}
```

FPGA Optimized Implementation

```
for(row = 0; row < rows+1; row++){
  for(col = 0; col < cols+1; col++){
   if(col < cols){
                buff_A.shift_up(col);
                temp = buff_A.getval(0,col);
   }
   if(col < cols & row < rows){
        buff_A.insert_bottom(rgb2y(input_pixel[row][col]),col);
   }
    buff_C.shift_right();
  if(col < cols){
                buff_C.insert(buff_A.getval(2,col),0,2);
                buff_C.insert(temp,1,2);
                buff_C.insert(rgb2y(tempx),2,2);
   }
   if( row <= 1 || col <= 1 || row > (rows-1) || col > (cols-1)){
                edge.R = edge.G = edge.B = 0;
   }
   else{
      edge = sobel_operator(&buff_C);
   }
   if(row > 0 && col > 0){
                AXI_PIXEL output_pixel;
                output_pixel.data = (edge.B,edge.G);
                output_pixel.data = (output_pixel.data, edge.R);
                out_pix[row-1][col-1] = output_pixel;
   }
  }
 }
}
```

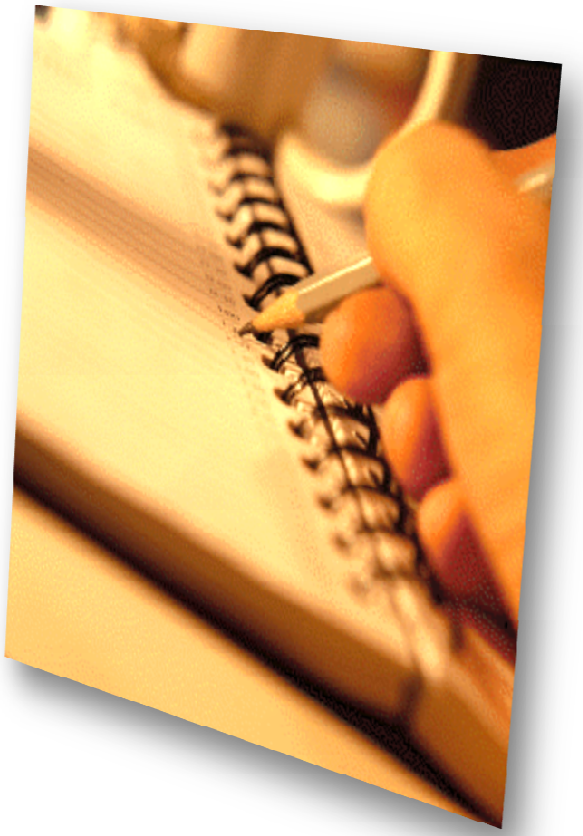
- **Changes with processor code center on data movement**
  - Creating line buffers “buff_A”
  - Creating 3x3 memory window for edge processing “buff_C”
  - Memory data movement operations on “buff_A” and “buff_C”
  - Iteration space extension to account for line buffers “rows+1 , cols+1”

Memory management code required for high performance implementation

Original processor code will synthesize in AutoESL, but will have poor performance.

Agenda

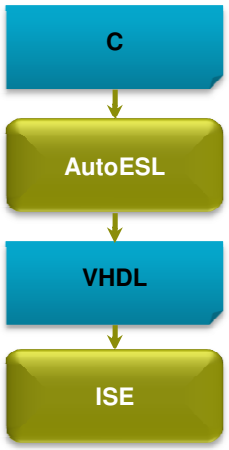
- High-Level Synthesis
- AutoESL
- Design Examples
- Success Stories



QR Decomposition

- **Application**
  - Radar
    - 1024x64 QRD
    - 120MHz
    - Floating Point
- **Key Benefit**
  - Reduced design time
  - Floating Point synthesis
  - Improved QoR

LOCKHEED MARTIN

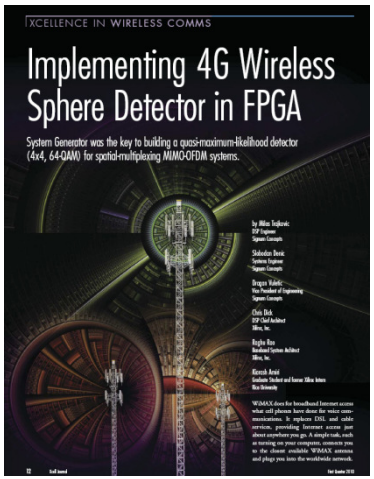


	Hand-coded VHDL	AutoESL C
Design Time (weeks)	12	1
Latency (ms)	37	21
Memory (RAMB18E1)	134 (16%)	10 (1%)
Memory (RAMB36E1)	273 (65%)	138 (33%)
Registers	29686 (9%)	14263 (4%)
LUTs	28152 (18%)	24257 (16%)

Quick path to FPGA

Sphere Decoder

- **Overview**
  - Key block in receiver structure in MIMO-OFDM systems
  - 5MHz MIMO-OFDM; 360 data sub-carriers every 102 usec
  - 4×4 antenna configuration; 64-QAM
  - ***Challenging computational requirements***
- **RTL Reference Implementation**
  - Target clock frequency was 225MHz in Virtex-5
  - Implemented in System Generator
  - Reference Matlab
- **HLS implementation**
  - Same BER performance
  - Target same design point as reference implementation

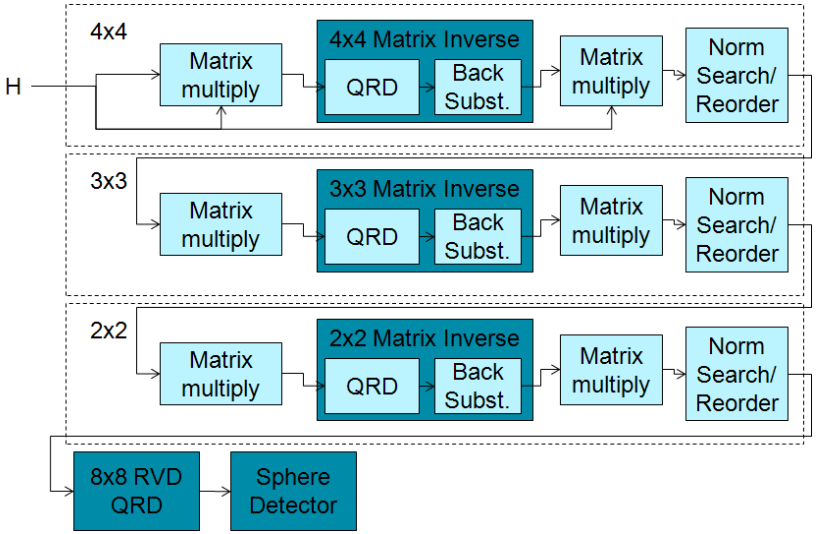


http://www.xilinx.com/publications/archives/xcell/Xcell70.pdf

http://www.xilinx.com/publications/archives/xcell/issue74/wireless-mimo-sphere-detector.pdf

Sphere Decoder: Results

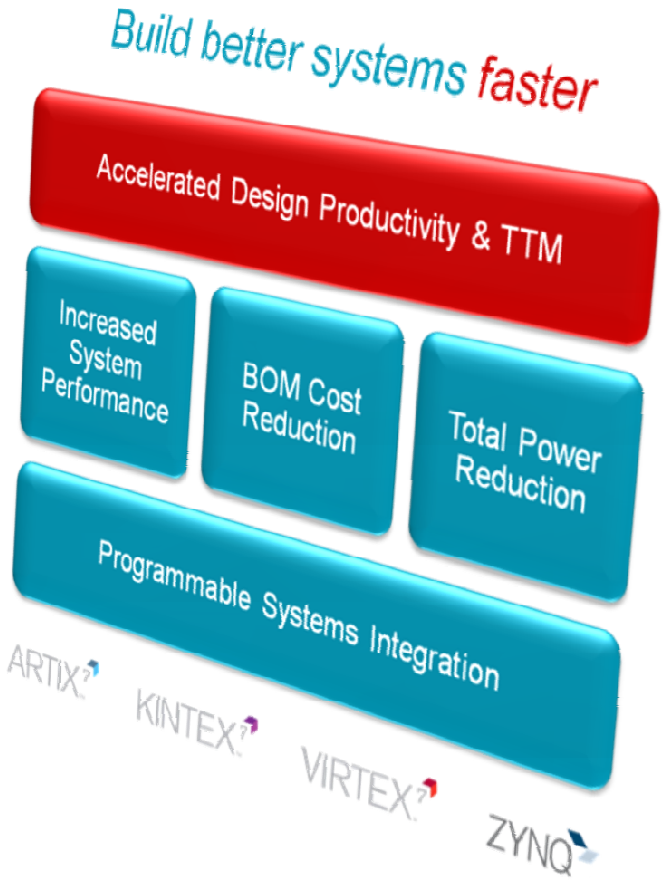
- **Compare Development Time and Quality of Results**
- * **Development time for AutoESL includes:**
  - Learning the tool
  - Producing results
  - Design space exploration
  - Detailed verification



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%

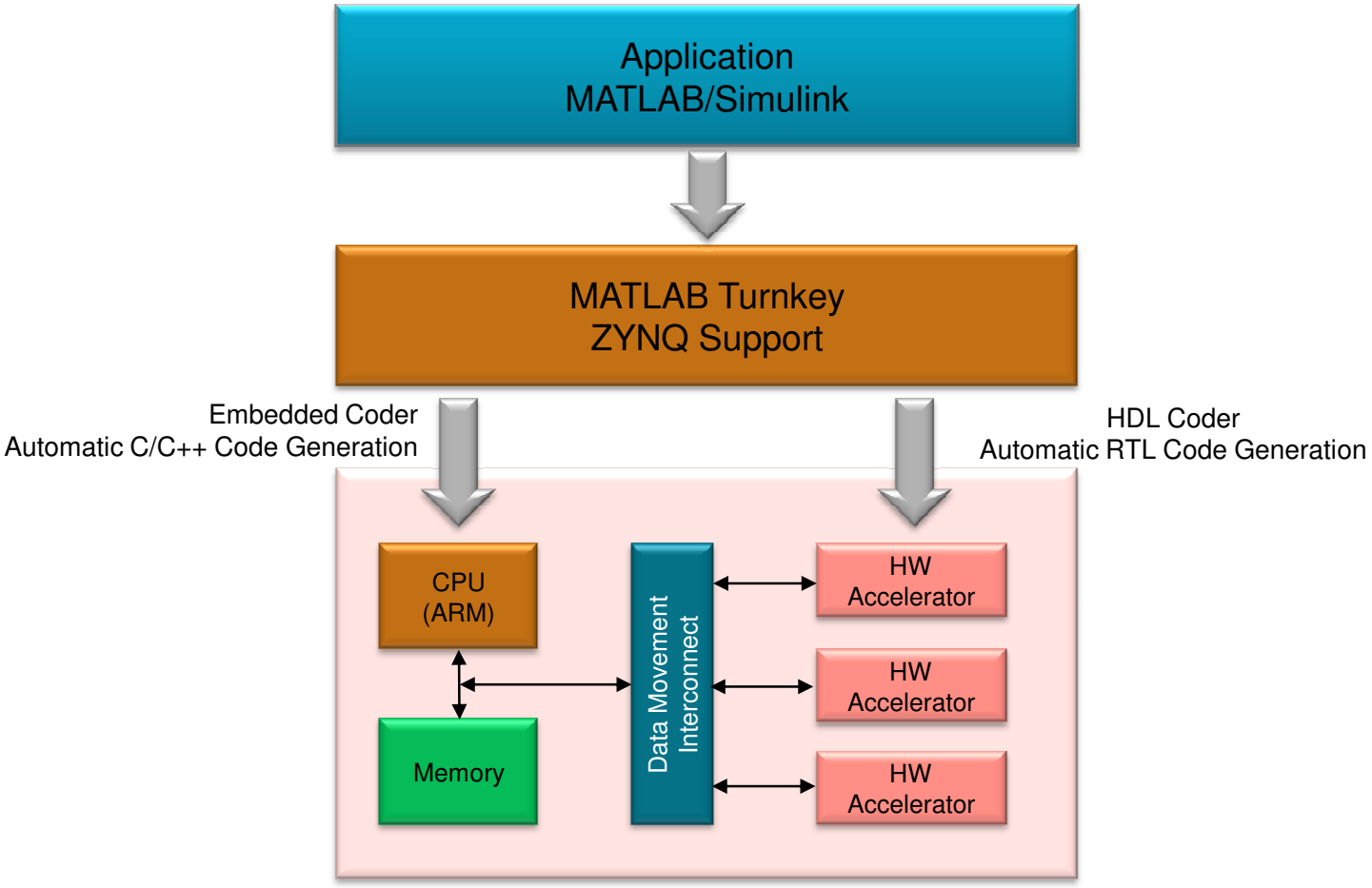
Summary

- Design Productivity
- Automatically schedules/resource sharing
- Portability/ Design Reuse
- Floating point support
- QOR
- Good for algorithmically complex design



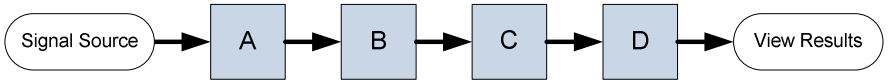
MathWorks Support for ZYNQ (Processing System + Programmable Logic)

Compiling Applications to CPU+FPGA

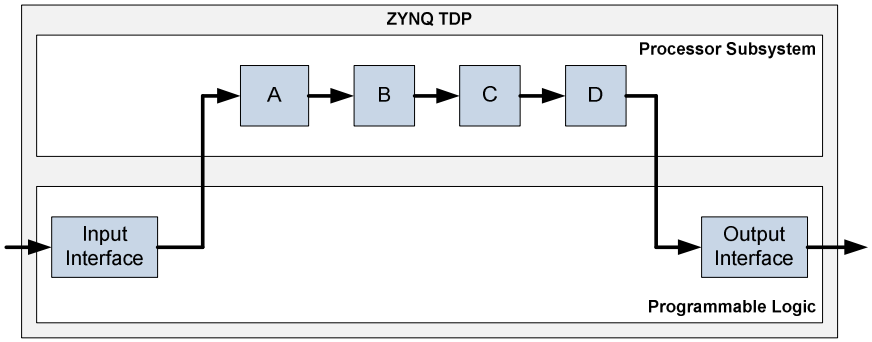


HW / SW Partitioning

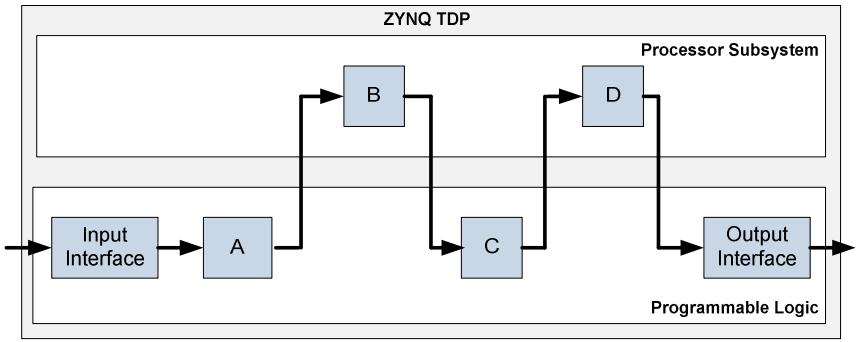
Simulink Algorithm



Execute on Processor

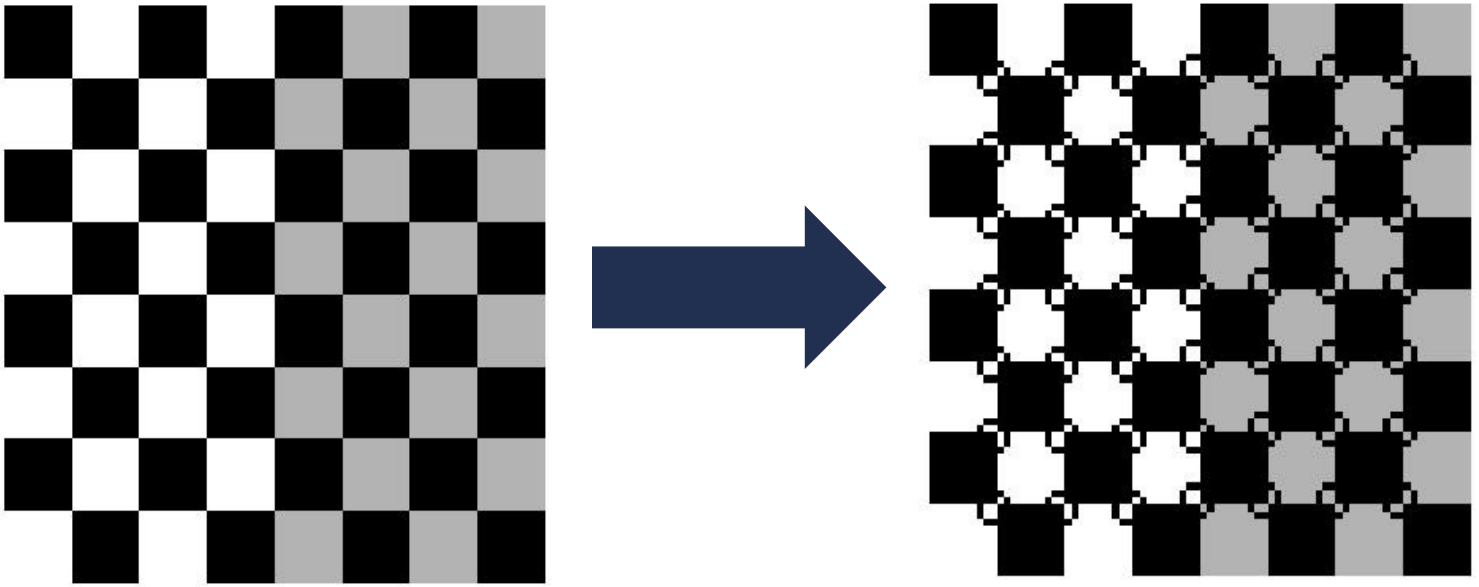


HW / SW Partition

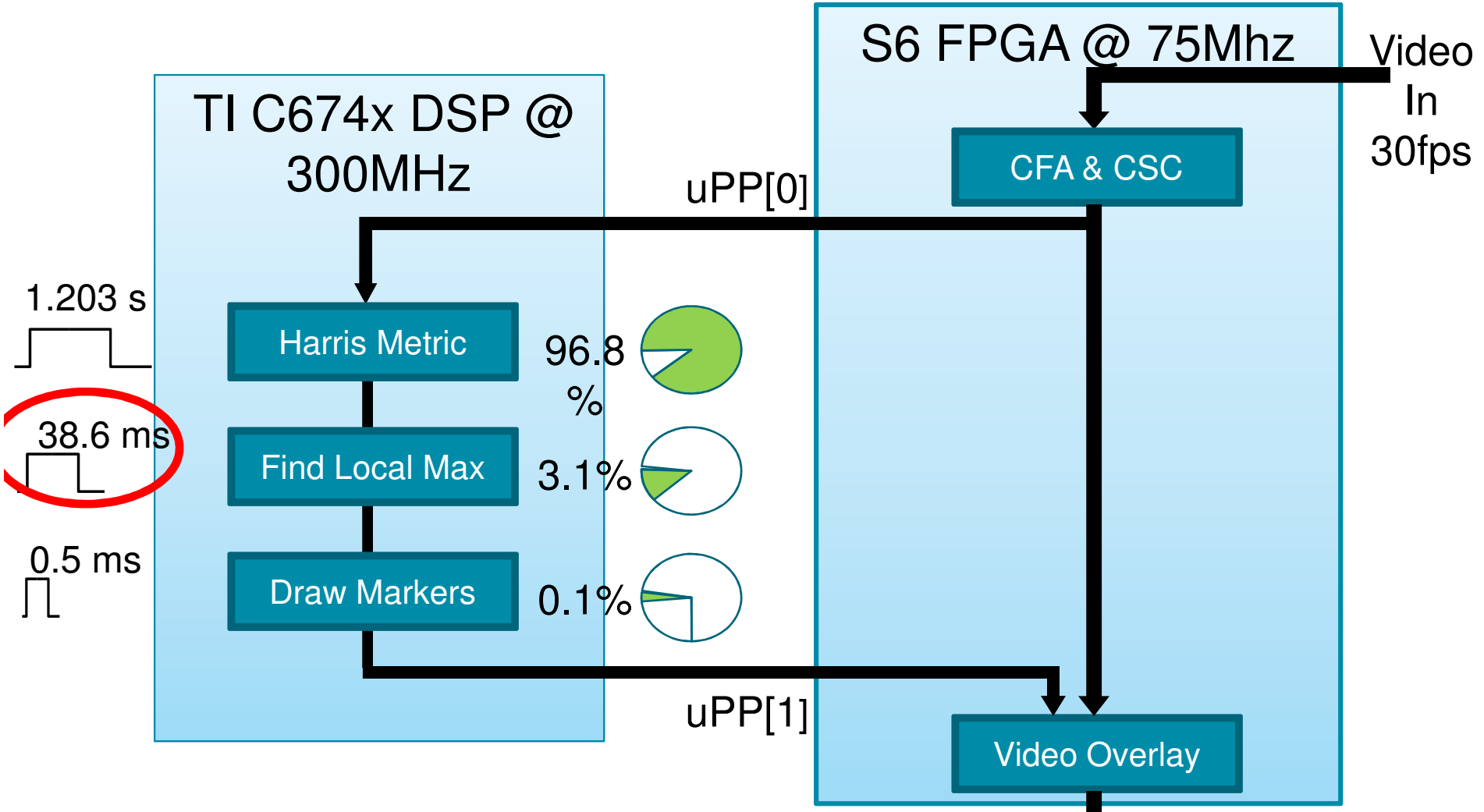


Harris-Stephens' Corner Detection Algorithm

- **Corner detection is used in many image processing & computer vision applications**
  - image mosaicking
  - object recognition

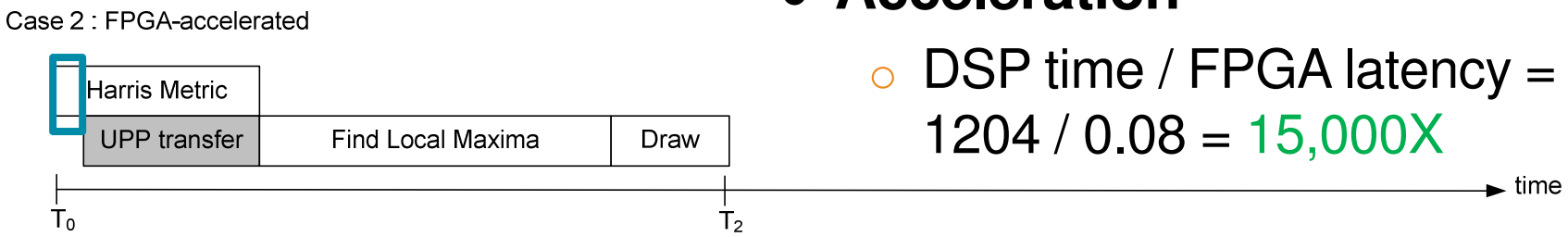


Measuring Block Execution Time – DSP only



- Toggle GPIO at code-block boundaries
  - Pulse-width = execution time

Calculating the Latency of the FPGA's Harris Metric



- **Acceleration**
  - DSP time / FPGA latency = 1204 / 0.08 = 15,000X

- **TI DSP processing is done sequentially (DSP takes 1.203 secs)**
  - **FPGA processing and UPP transfer occur simultaneously**
    - **Latency of FPGA Harris Metric**

– Delay Lines
  - Sobel Gradients (3x3) = 1 line
  - Gaussian Filters (5x5) = 2 lines
  - Total Cycles (including blanking) = 3 * 1056 = 3,168 cycles

– Additional Cycles (adders, multipliers, etc…) = 28 cycles

– Total latency = 3,196 * (1/40MHz) = 80 usec

Summary

- **Leverages power of MathWorks modeling environment for DSP and Video Processing**
  - MATLAB, Simulink, IP Toolboxes
- **Includes automatic code generation for HW / SW**
  - HDL Coder, MATLAB Coder, Simulink Coder, Embedded Coder
- **Developing New Technology to improve abstraction**
  - Turnkey Platform Support
  - System Performance Estimation
  - “One-Click” implementation
- **Path to Optimized Results**
  - Generated C-code optimized with NEON instructions
  - ARM Intrinsic IP
  - Xilinx DSP IP

Appendix

Artix-7 FPGA Product Table

Artix-7 FPGAs
Optimized for Lowest Cost and Power with Small Form-Factor Packaging for Highest Volume Applications
(1.0V, 0.9V)

	Part Number		XC7A100T	XC7A200T	XC7A350T
Logic Resources	Slices		15,850	33,650	56,250
	Logic Cells		101,440	215,360	360,000
	CLB Flip-Flops		126,800	269,200	450,000
Memory Resources	Maximum Distributed RAM (Kbits)		1,188	2,888	4,638
	Block RAM/FIFO w/ ECC (36Kbits each)		135	365	515
	Total Block RAM (Kbits)		4,860	13,140	18,540
Clock Resources	CMTs (1 MMCM + 1 PLL)		6	10	12
I/O Resources	Maximum Single-Ended I/O[4]		300	500	600
	Maximum Differential I/O Pairs[4]		144	240	288
Embedded Hard IP Resources	DSP48E1 Slices		240	740	1,040
	PCI Express®[1]		1	1	1
	Agile Mixed Signal (AMS) / XADC		1	1	1
	Configuration AES / HMAC Blocks		1	1	1
	GTP 5.4 / 6.6 Gb/s Transceivers		8	16	16
Speed Grades	Commercial		-1, -2	-1, -2	-1, -2
	Extended		-2L, -3	-2L, -3	-2L, -3
	Industrial		-1, -2	-1, -2	-1, -2
Configuration	Configuration Memory (Mbits)		29.3	62.4	96.1
	Package[3]	Dimensions (mm)	Available User I/O: 3.3V SelectIO™ Pins (GTP Transceivers)		
	CSG324	15 x 15	210 (0)		
	FTG256	17 x 17	170 (0)		
	SBG484	19 x 19		285 (4)	
Footprint Compatible	FGG484[2]	23 x 23	285 (4)		
	FBG484[2]	23 x 23		285 (4)	285 (4)
Footprint Compatible	FGG676[2]	27 x 27	300 (8)		
	FBG676[2]	27 x 27		400 (8)	400 (8)
	FFG1156[2]	35 x 35		500 (16)	600 (16)

XMP086 (v3.1.1)

CSG: 0.8mm Wire-bond chip-scale; **FTG**: 1.0mm Wire-bond fine-pitch; **SBG**: 0.8mm Lidless flip-chip; **FGG**: 1.0mm Wire-bond fine-pitch; **FBG** 1.0mm Lidless flip-chip; **FFG**: 1.0mm Flip-chip fine-pitch

Notes: 1. Supports PCI Express Base 2.1 specification at Gen1 and Gen2 data rates.
2. Leaded package options available.
3. Device migration is available within the Artix-7 family for like packages but is not supported between other 7 series families.

Product Tables

Zynq-7000 Device Table

HW Designer's View

BACK

Artix Fabric

Kintex Fabric

		Zynq™-7000 Extensible Processing Platform									
	Device Name	Z-7010		Z-7020		Z-7030			Z-7045		
	Part Number	XC7Z010		XC7Z020		XC7Z030			XC7Z045		
	Processing System (Dual ARM® Cortex™-A9 MPCore™ with NEON™ & Double Precision FPU Cache, Memory Controllers, DMA, Security and Peripherals)	Same Processing System for All Devices.									
Programmable Logic	Xilinx 7 Series Programmable Logic Equivalent	Artix™-7 FPGA		Artix™-7 FPGA		Kintex™-7 FPGA			Kintex™-7 FPGA		
	Programmable Logic Cells (Approximate ASIC Gates[2])	28K Logic Cells (~430K)		85K Logic Cells (~1.3M)		125K Logic Cells (~1.9M)			350K Logic Cells (~5.2M)		
	Logic Cells	28,160		85,120		125,760			349,760		
	Look-Up Tables LUTs	17,600		53,200		78,600			218,600		
	Flip Flops	35,200		106,400		157,200			437,200		
	Extensible Block RAM (# 36 Kb Blocks)	240 KB (60)		560 KB (140)		1,060 KB (265)			2,180 KB (545)		
	Programmable DSP Slices (18x25 MACCs)	80		220		400			900		
	Peak DSP Performance (Symmetric FIR)	58 GMACS		158 GMACS		480 GMACS			1080 GMACS		
	PCI Express® (Root Complex or Endpoint)	—		—		Gen2 x4			Gen2 x8		
	Agile Mixed Signal (AMS) / XADC	2x 12 bit, 1 MSPS ADCs with up to 17 Differential Inputs									
	Security[1]	AES and SHA 256b for secure configuration									
Packages	Package Type	CLG400	CLG484	CLG400	CLG484	FBG484	FBG676	FFG676	FBG676	FFG676	FFG900
	Size (mm)	17x17	19x19	17x17	19x19	23x23	27x27	27x27	27x27	27x27	31x31
	Pitch (mm)	0.8	0.8	0.8	0.8	1.0	1.0	1.0	1.0	1.0	1.0
	Processing System Total I/Os (includes Multiplexed I/Os)	130	130	130	130	130	130	130	130	130	130
	Multi-Standards and Multi-Voltage SelectIO™ Interfaces (1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V)	100	100	120	200	100	100	100	100	100	200
	Multi-Standards and Multi-Voltage High Performance SelectIO Interfaces (1.2V, 1.35V, 1.5V, 1.8V)	—	—	—	—	63	150	150	150	150	150
	Serial Transceivers	—	—	—	—	4	4	4	8	8	16
	Maximum Transceiver Speed (Speedgrade Dependant)	N/A	N/A	N/A	N/A	6.6 Gb/s	6.6 Gb/s	12.5 Gb/s	6.6 Gb/s	12.5 Gb/s	12.5 Gb/s

Notes: 1. Security is shared by the Processing System and the Programmable Logic.
2. Eqivalent ASIC gate count is dependent of the function implemented. The assumption is 1 Logic Cell = ~15 ASIC Gates.
3. Preliminary product information. Subject to change. Please contact your Xilinx representative for the latest information

Device Tables

XILINX ALL PROGRAMMABLE™

Zynq-7000 Device Table

SW Developer's View

BACK

Artix Fabric | Kintex Fabric

		Zynq™-7000 Extensible Processing Platform			
	Device Name	Z-7010	Z-7020	Z-7030	Z-7045
	Part Number	XC7Z010	XC7Z020	XC7Z030	XC7Z045
Processing System	Processor Core	Dual ARM® Cortex™-A9 MPCore™ with CoreSight™			
	Processor Extensions	NEON™ & Single / Double Precision Floating Point			
	Maximum Frequency	800 MHz			
	L1 Cache	32 KB Instruction, 32 KB Data per processor			
	L2 Cache	512 KB			
	On-Chip Memory	256 KB			
	External Memory Support	DDR3, DDR2, LPDDR2			
	External Static Memory Support	2x Quad-SPI, NAND, NOR			
	DMA Channels	8 (4 dedicated to Programmable Logic)			
	Peripherals	2x USB 2.0 (OTG) w/DMA, 2x Tri-mode Gigabit Ethernet w/DMA, 2x SD/SDIO w/DMA, 2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO			
	Security	AES and SHA 256b for secure boot			
	Peripherals and Static Memory Multiplexed I/O[1]	54			
	Processing System to Programmable Logic Interface Ports (Primary Interfaces & Interrupts Only)	2x AXI 32b Master, 2x AXI 32b Slave, 4x AXI 64b/32b Memory AXI 64b ACP 16 Interrupts			
Programmable Logic	Xilinx 7 Series Programmable Logic Equivalent	Artix™-7 FPGA	Artix™-7 FPGA	Kintex™-7 FPGA	Kintex™-7 FPGA
	Programmable Logic Cells (Approximate ASIC Gates[3])	28K Logic Cells (~430K)	85K Logic Cells (~1.3M)	125K Logic Cells (~1.9M)	350K Logic Cells (~5.2M)
	Extensible Block RAM (# 36 Kb Blocks)	240KB (60)	560KB (140)	1,060KB (265)	2,180 KB (545)
	Programmable DSP Slices (18x25 MACCs)	80	220	400	900
	Peak DSP Performance (Symmetric FIR)	58 GMACS	158 GMACS	480 GMACS	1080 GMACS
	PCI Express® (Root Complex or Endpoint)	—	—	Gen2 x4	Gen2 x8
	Agile Mixed Signal (AMS) / XADC	2x 12 bit, 1 MSPS ADCs with up to 17 Differential Inputs			
	Security	AES and SHA 256b for secure configuration			
	Multi-Standards I/O[2]	100	200	250	350
	Serial Transceivers[2]	—	—	4	16

Notes: 1. Static memory interface combined with the usage of many peripherals could require more than 54 I/Os. A designer can use the Programmable Logic I/Os.
2. Total Number of I/O and Transceivers depends on package used.
3. Eqivalent ASIC gate count is dependent of the function implemented. The assumption is 1 Logic Cell = ~15 ASIC Gates.
4. Preliminary product information. Subject to change. Please contact your Xilinx representative for the latest information

Device Tables

XILINX ALL PROGRAMMABLE™

Video and Image Processing IP Pack

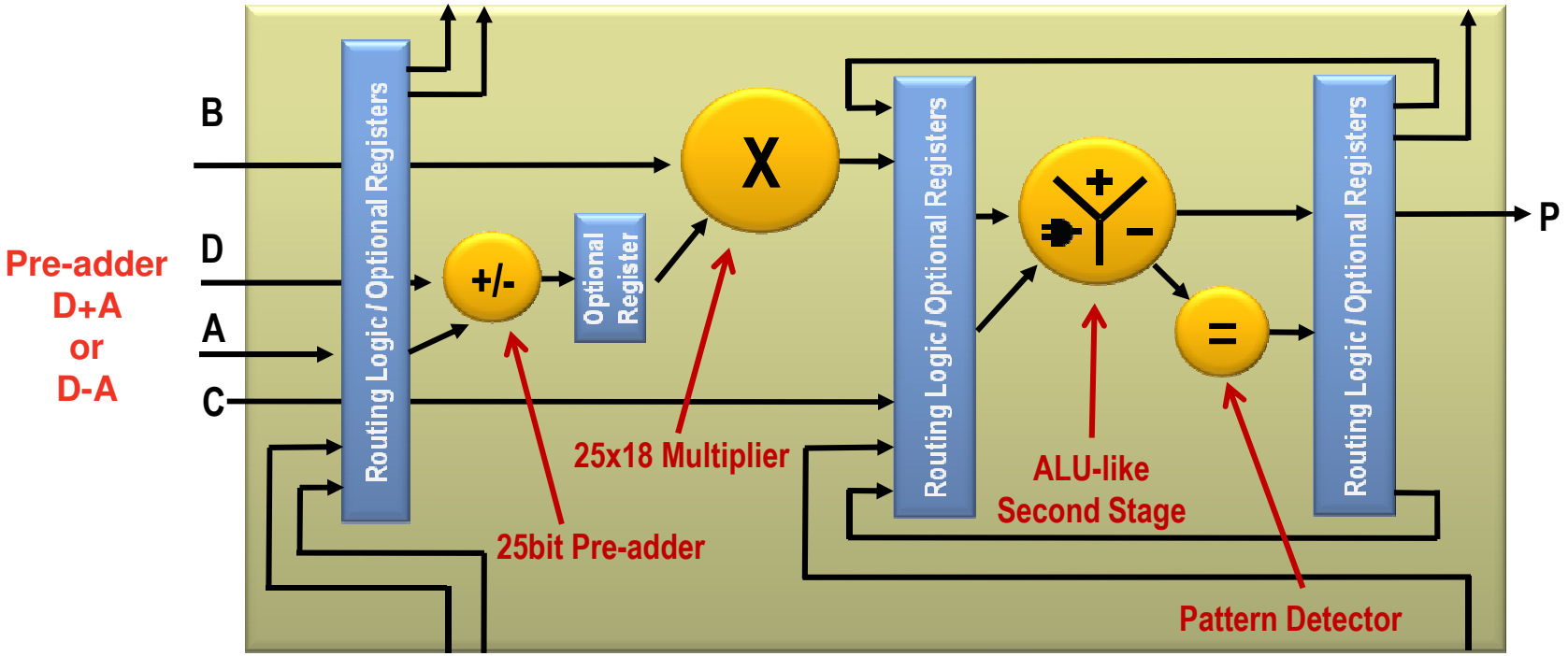
Bundled IP Pack Providing Savings and Convenience

- **Single IP core bundle providing low cost option for collection of IP**
- **Xilinx Deinterlacer is included**
- **Price $3k**

Video and Image Processing Pack

- Defective Pixel Correction
- Color Filter Array Interpolation
- Gamma Correction
- Color Correction Matrix
- Edge Enhancement
- Noise Reduction (2D)
- Image Statistics
- Chroma Resampler
- Video Scaler
- On-Screen Display
- Timing Controller
- Motion Adaptive Noise Reduction (3D)
- Image Characterization
- Object Segmentation
- Deinterlacer

Comprehensive DSP Design Platform - V6 DSP48E1



Power Consumption Benefits	Performance Benefits	Cost Benefits
▪ Lowest power operation of any FPGA solution ▪ 1.23mW/100Mz at 38% toggle rate	▪ 600MHz operations for any DSP operation ▪ ~1.2 TeraMAC/s in a single device	▪ Hardened pre-adder and adder cascade saves significant resources ▪ Logic functions can be mapped into DSP blocks

The XtremeDSP Slice

- **Xilinx DSP FPGAs include dedicated DSP processing blocks called DSP48s**
  - Spartan-6: DSP48A1 (18 bit pre-adder + 18x18 multiplier)
  - Virtex-6 DSP48E1 (25 bit pre-adder + 18x25 multiplier)
- **Blocks individually configurable to perform over 30 unique arithmetic functions**
  - Dynamic operational modes adapts DSP48 slice from clock to clock
- **V6 DSP48E1 has expanded capabilities compared to S6 DSP48A1**

V6 DSP48E1 or S6 DSP48A1 **=**

One Multiplier

+

300 Logic Cells

- 300 is typical
- actual range dependent upon application (range 50-1300 logic cells saved)

How Else Can the DSP48E1 Be Utilized?

- 48 bit compare, absolute value
- Counter (count limited and free running)
- 3 input 48 bit adder, subtract
- 25x18 bit multiply
- 25x25 bit pre-adder, 48 bit accumulator
- Cascade support that allows for larger mults or adds
- SIMD mode – 4x12 bit, 2x24, or 2x48 bit add or subtract using a single DSP48 slice (SAD operations!)
- Bus Multiplexer
- Shifters (left, right, barrel)
- Bit operations (and, or, not, nand, nor, xor, xnor, magnitude compare, equal to zero, greater than zero, less than zero)
- Complex multiply (4 clock cycles in one DSP48E1
- Overflow, underflow reporting, convergent rounding support, symmetric rounding support, random rounding, terminal count detection support and auto resetting, saturation
- Supports block floating point operations
- Divide (via successive mults or shift and subtract operations), square root operations (successive mults and subs).
- Filters!