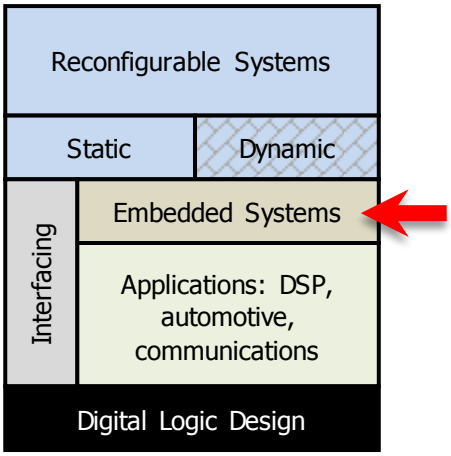
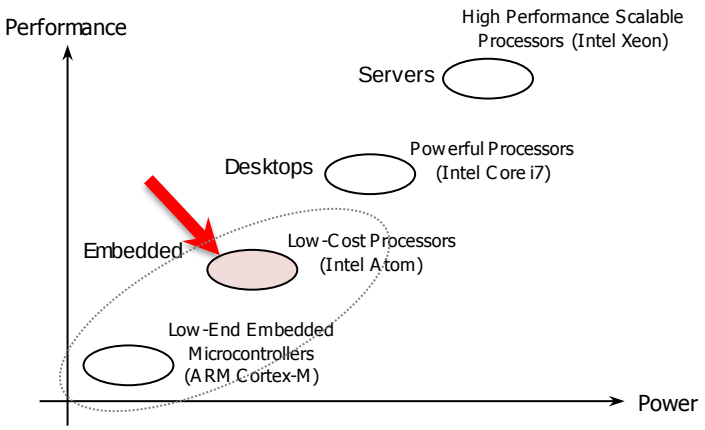
High-Performance Embedded Programming with the Intel® Atom™ Platform

INSTRUCTOR	Daniel Llamocca
CONTACT INFO	email: llamocca@oakland.edu
WEBPAGE	www.secs.oakland.edu/~llamocca/emb_intel.html
REFERENCES	▪ Lori M. Matassa, Max Domeika, "Break Away with Intel® Atom™ Processors: A Guide to Architecture Migration", Intel Press, 2010. ▪ Max Domeika, "Software Development for Embedded Multi-Core Systems", Newnes, 2008.
MATERIALS	Terasic DE2i-150-FPGA Development Kit. Ubuntu Distribution 12.04.4





DESCRIPTION

- Real-time embedded programming, analysis, and optimization using the Intel® Atom™ processor. Multi-threaded systems, multi-core software development, scalability, and industry-tailored applications.

OUTLINE OF TOPICS

Getting Started with the Hardware and Software Platform	▪ Hardware Platform: Terasic DE2i-150 Dev. Kit ▪ Ubuntu 12.04.4 ✓ Installation ✓ C++ Compiler and TBB library installation ▪ Board Setup and Examples	
C/C++ Programming Basics	C	▪ Basics, pointers, functions, structures, dynamic memory allocation ▪ Functions: compiling with different files
	C++	▪ Classes, objects, functors
Multithreading	▪ Image Convolution. ▪ Matrix multiplication. ▪ Mutexes: dot product	
Multicore software development	▪ Threading Building Blocks (TBB): *parallel_for* ▪ Threading Building Blocks (TBB): *parallel_reduce* ▪ Pipelining: Threading Building Blocks (TBB): *parallel_pipeline*	
Real-Time Applications	▪ Interrupts: software, keyboard, real-time clock ▪ Direct Memory Access	
Optimizing Real-Time Embedded Applications	▪ Power and Performance Analysis Tools ▪ Power Optimization	
Applications	▪ Convolutional Neural Network ▪ Beamforming	

LIST OF SOFTWARE APPLICATION FILES

Getting Started		`fibo.c`
		`saxpy_ex.c`
		`saxpy_fun.h, saxpy_fun.c`
			Fibonacci sequence
			SAXPY

C/C++ fundamentals	*2D convolution*	`conv2.c`
		`conv2_fun.c, conv2_fun.h`
		`input.txt, sharpen_kernel.txt`
		`Makefile`
			Matrix convolution.
			Read/write text files.
			Makefile

	Image convolution	`imgconv.c`
		`imgconv_fun.c, imgconv_fun.h`
		`edge_kernel.txt`
		`Makefile`
		`img_op.m, iss.bif, iss.jpg`
			Image convolution
			Read/write binary files.
			Makefile
			MATLAB script for verification

	Classes Basics	`class_samples.cpp`
		`basic_constrs.cpp`
		`basic_functors.cpp`
			Basic examples for classes

		`simple.cpp, simple_fun.cpp,` `simple_fun, Makefile`	Simple class split into two files (.cpp,.h). Use of Makefile for C++.
	Neuron	`neuron_imp.cpp`	Neuron implementation

Multithreading	*Basic examples*	`pthreads_example.c`
		`dot_prod.c`
		`mutex_exam.c`
			Basic thread generation. Basic mutex example
			Dot product with mutex

	2D convolution	`conv2m.c`
		`sharpen_kernel.txt, input.txt`
		`conv2m_pthreads.c`
		`conv2m_fun.c, conv2m_fun.h`
		`kernel_a.txt, kernel_b.txt`
		`kernel_c.txt`
		`Makefile`
		`img_op_m.m, iss.bif, iss.jpg`
			2D convolution (small matrix or image)
			Makefile
			MATLAB script for verification

	Matrix multiplication	`matrix_mult.c`
		`matrix_mult_pthreads.c`
		`matrix_mult_pthreads_old.c`
		`mat_fun.c, mat_fun.h`
			Matrix multiplication (non-threaded and multi-threaded)

Multicore Software Development	*TBB – parallel_for*	`basic.cpp`	Element-wise Vector operation: $a(i)^2$. Using class.
		`myceil.cpp`	Element-wise Vector operation: $\left\lfloor {}^{a(i)}/_{2} \right\rfloor$. Normal and compact lambda expressions.
		`simple.cpp`	Element-wise Vector operation: $a(i) \times 2$. Normal and compact lambda expressions.
		`mov_avg.cpp`	3-element moving average. Compact lambda expression
		`vector_op.cpp`	Vector operations: $c(i) \leftarrow a(i) + b(i), d(i) \leftarrow a(i) \times b(i)$ Compact lambda expressions.

		`morpho.cpp, morpho_fun.cpp,`
		`morpho_fun.h, Makefile`
		`morpho.m, uchip.bif, uchip.jpg`
			Grayscale morphology: dilation, erosion.

	TBB – parallel_reduce	`accum.cpp`	Accumulate a vector
		`accum_sq.cpp`	Accumulate the squared elements of a vector
		`mypi.cpp`	Compute pi. Seq. vs TBB

		`dot_product.cpp,`
		`dot_product_fun.cpp,`
		`dot_product_fun.h, Makefile`
			Dot product between two vectors
			Sequential vs. TBB (use both reduce and for)

		`vm.cpp`	Get maximum of each row of matrix (use *parallel_for* and *parallel_reduce*)
	TBB – run		
	TBB - pipelining	`pip_mod.cpp`	Element-wise modulus of 2 vectors
		`pip_sumsq.cpp`	Accumulation of squared elements
		`pip_avgvec.cpp`	element-wise powering and averaging of two vectors. Item = 2 vectors

Real-Time Programming		`basic_sigint.c`
		`basic_sigalrm.c`
		`rtctst.c`
			Using SIGINT signal
			Using SIGALRM signal
			Testing the RTC driver