



EE 443

Design & Application of DSP
Spring 2016



Introduction

- Prof. Jenq-Neng Hwang, (hwang@uw.edu)
- Office Hours: EEB M426, TTh 1:00—2:30 pm
- Two courses sequence: EE442 (signal transforms and filtering) and EE443 (applications and real-time DSP microprocessor design)
- The prerequisite for this course is **EE442**
 - **DFT and FFT Computation**
 - **FIR Filter Design**
 - **IIR Filter Design**
 - **Adaptive Filter Design**
- TAs: Jounsup Klm
- TA Office Hours:
- Lab HW Demo: (15 min/group)
- Credits: 1 Science and 4 Design

Goals and Grading

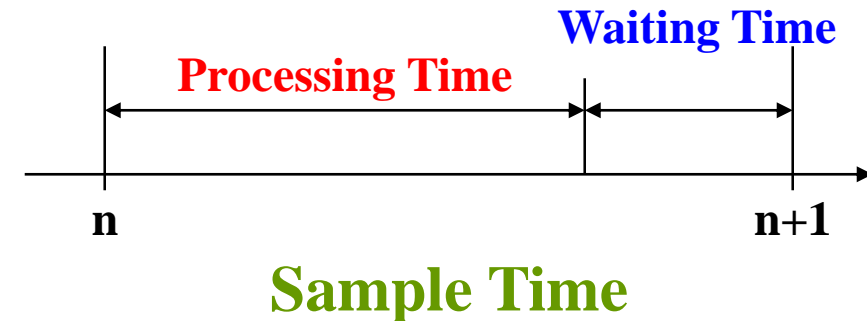
- **Goals:** The goal of this course is to introduce students, who are majoring in Signal/Image Processing and Communications, the important laboratory components of real-time DSP based on commercially available DSP microprocessors.
- **Grading:** Homework: 30%, Final Report Proposal: 10%, Final Project Presentation 10%, Final Project Demo & Report 50%.
- **Textbook:**
- **Ref:** Lori M. Matassa and Max Domeika, Break Away with Intel Atom Processors, Intel Press, 2010.

Topics

- De2i-150: NIOS II software processor and the Cyclone IV FPGA (0.5 week)
- Input/Output via AIC23 daughter card (0.5 week)
- Assembly (0.5 week)
- Intel Atom Processor and Image/Video Acquisition (0.5 week)
- FIR Filters and IIR Filters Implementations (0.5 week)
- FFT Implementations and Adaptive Filters (0.5 week)
- TI's DSP board, DSK6713 (0.5 week)
- Digital Speech Recognition (0.5 week)
- Face Detection and Recognition (0.5 week)
- Video Object Segmentation and Shadow Removing (0.5 week)

Real-Time DSP Processing

- Note that the processing can involve many past sampled data, not just the most recently sampled one
- We can say that we have a **real-time** application if:
 - $\text{Waiting Time} \geq 0$



Why DSP Processors?

- Use a **digital signal processor (DSP)** when the following are required:
 - Cost saving.
 - Smaller size.
 - Low power consumption.
 - Processing of many “high” frequency signals in real-time.
- Use a **general purpose processor (GPP)** when the following are required:
 - Large memory.
 - Advanced operating systems.

Typical DSP Algorithms

- The **Sum of Products (SOP)**, or called Multiply & Accumulation (MAC), is the key element in most DSP algorithms:

Algorithm	Equation
Finite Impulse Response Filter	$y(n) = \sum_{k=0}^M a_k x(n-k)$
Infinite Impulse Response Filter	$y(n) = \sum_{k=0}^M a_k x(n-k) + \sum_{k=1}^N b_k y(n-k)$
Convolution	$y(n) = \sum_{k=0}^N x(k)h(n-k)$
Discrete Fourier Transform	$X(k) = \sum_{n=0}^{N-1} x(n) \exp[-j(2\pi / N)nk]$
Discrete Cosine Transform	$F(u) = \sum_{x=0}^{N-1} c(u) \cdot f(x) \cdot \cos\left[\frac{\pi}{2N} u(2x+1)\right]$

Hardware vs. Microcode Multiplication

- DSP processors are optimized to perform multiplication and addition operations.
- Multiplication and addition are normally done in hardware and in one cycle.
- Example: 4-bit multiply (unsigned).

Hardware

```
  1011
x 1110
-----
10011010
```

Microcode

```
  1011
x 1110
-----
  0000  Cycle 1
 1011.  Cycle 2
1011..  Cycle 3
1011... Cycle 4
-----
10011010 Cycle 5
```


Floating vs. Fixed Point Processors

- Applications which require:
 - High precision, and wide dynamic range.
 - High signal-to-noise ratio, and ease of use.

Need a floating point processor.

- Drawback of floating point processors:
 - Higher power consumption.
 - **Can be** higher cost.
 - **Can be** slower than fixed-point counterparts and larger in size.
- For educational purposes, use the floating-point device (e.g., C6713) as it can support both fixed and floating point operations.

General Purpose DSP vs. DSP in ASIC (or SoC)

- Application Specific Integrated Circuits (**ASICs**) and System on Chips (**SoCs**) are semiconductors designed for dedicated functions.
- The advantages and disadvantages of using ASICs (or SoCs) are listed below:

Advantages of ASIC/SoC	Disadvantages
<ul style="list-style-type: none">• High throughput• Lower silicon area• Lower power consumption• Improved reliability• Reduction in system noise• Low overall system cost	<ul style="list-style-type: none">• High investment cost• Less flexibility• Long time from design to market

Commercially Available DSP Chips

- The "big four" programmable DSP chip manufacturers are **Texas Instruments**, with the TMS320 series of chips; **Motorola**, with the DSP56000 and DSP96000 series; **AT&T**, with the DSP16 and DSP32 series; and **Analog Devices**, with the ADSP2100 series.
- Also Zilog, NEC, Xilinx FPGA, etc

Company	Part	MAC	No. bits	No. bits
		time (ns)	fixed pt.	float pt.
AT&T	DSP16	55	16/36	
	DSP16A	33	16/36	
	DSP32	160	16	32/40
	DSP32C	80	16 or 24	32/40
Motorola	DSP56001	74	24/56	
	DSP96002	75	32/64	44/96
Texas Inst.	TMS320C10	114-280	16/32	
	TMS32C25	80-100	16/32	
	TMS320C50	25-50	16/32	
	TMS320C30	50-75	24/32	32/40
	TMS320C40	40-50		32/40
	TMS320C6201	5	32	
Analog Devices	ADSP2100	125	16/40	
	ADSP2100A	80	16/40	
	ADSP21065L	6	32	32/40

Texas Instruments' TMS320 Family

C2000

Lowest Cost

Control Systems

- ♦ Motor Control
- ♦ Storage
- ♦ Digital Ctrl Systems

C5000

Efficiency

Best MIPS per Watt / Dollar / Size

- ♦ Wireless phones
- ♦ Internet audio players
- ♦ Digital still cameras
- ♦ Modems
- ♦ Telephony
- ♦ VoIP

C6000

Performance & Ease-of-Use

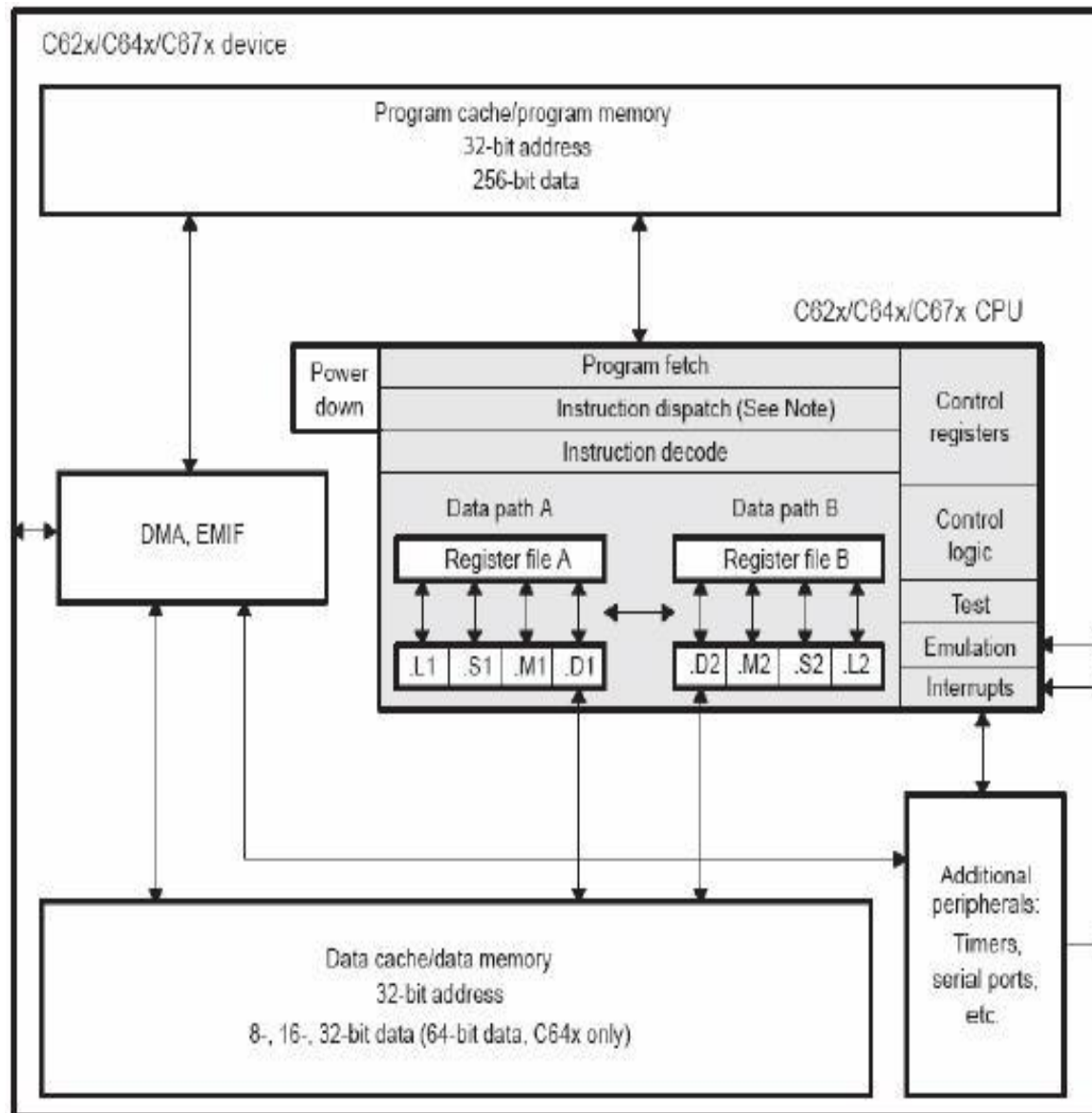
Multi Channel and Multi Function App's

- ♦ Comm. Infrastructure
- ♦ Wireless Base-stations
- ♦ DSL
- ♦ Imaging & Video
- ♦ Multimedia Servers

C6713 Floating Point DSPs

- The highest performance floating-point DSP
- Based on the high-performance, advanced VelociTI **very-long-instruction-word (VLIW)** architecture.
- **Eight** 32-bit instructions/cycle: 1800/1350 (225MHz), 1600/1200 (200 MHz), 1336/1000 (167MHz) **MIPs/FLOPs**
- The **eight functional units** provide four floating-/fixed-point ALUs, two fixed-point ALUs, and two floating-/fixed-point multipliers.
- **32** load-store structure general-purpose registers of 32-bit word length and **eight** highly independent functional units.

C67x CPU Diagram



One instruction is 32 bits. Program bus is 256 bits wide.

⇒ Can execute up to 8 instructions per clock cycle (225MHz→4.4ns clock cycle).

8 independent functional units:

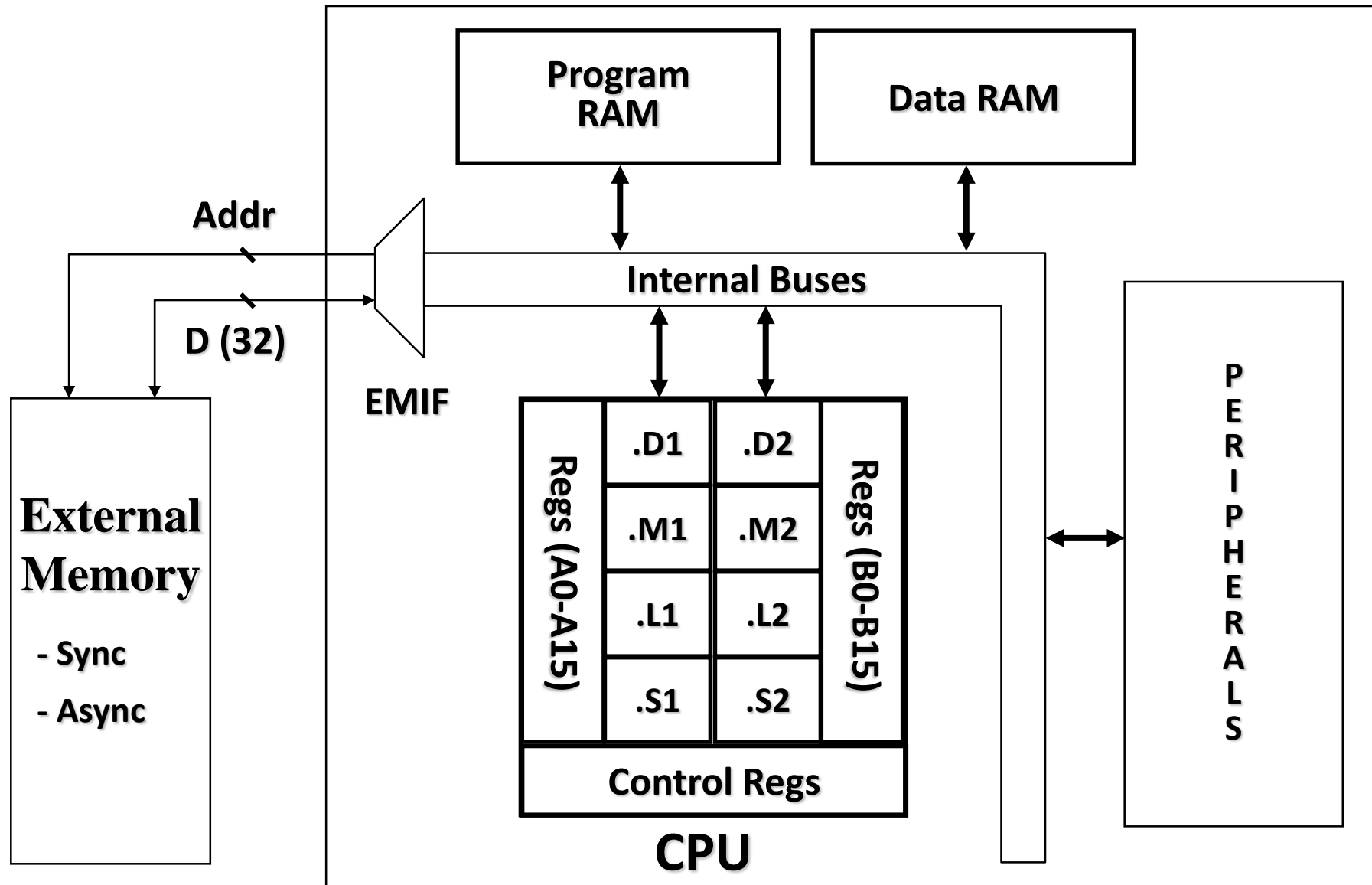
- 2 multipliers
- 6 ALUs

Code is efficient if all 8 functional units are always busy.

Register files each have 16 general purpose registers, each 32-bits wide (A0-A15, B0-B15).

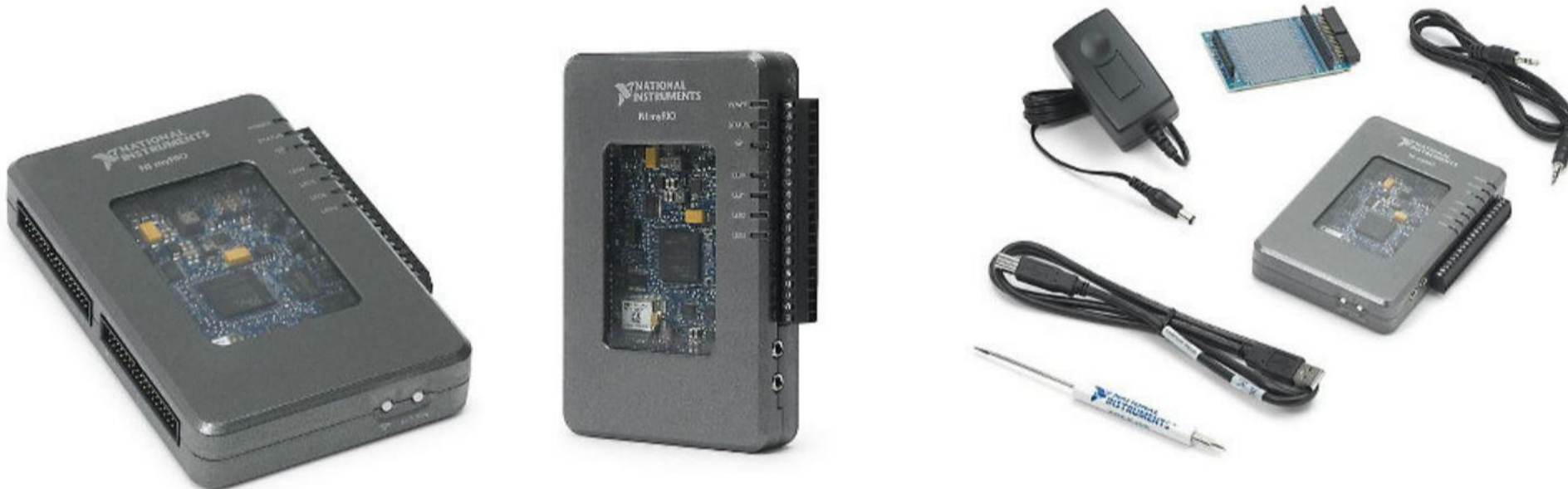
Data paths are each 64 bits wide.

C6713 CPU System Block



NI's myRIO Board

- 10 analog inputs, 6 analog outputs, 40 digital I/O lines
- Wireless, LEDs, push button, accelerometer onboard
- Xilinx FPGA and dual-core ARM Cortex-A9 processor
- Programmable with LabVIEW or C; adaptable for different programming levels



DE2i-150 FPGA Development Kit



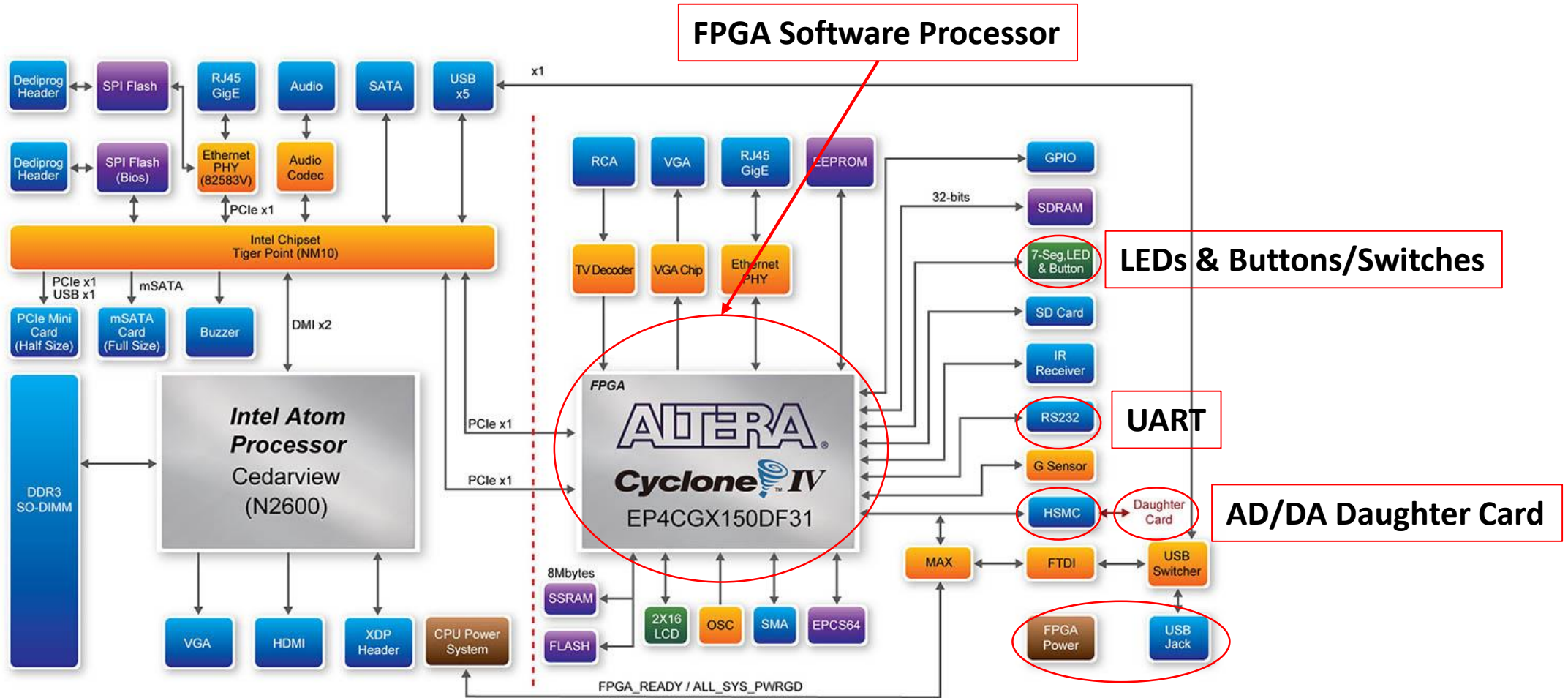
DE2i-150 FPGA Development Kit

- Donated by Intel to replace the old TI DSK DSP Development Kit in 2014
- Includes
 - Intel Atom N2600 Processor
 - Altera Cyclone IV GX EP4CGX150DF31 FPGA
- Not a traditional DSP development environment
- Will utilize a software processor (NIOS II) running on the Cyclone IV FPGA to emulate a microprocessor (for final project)
- We will have some image/video HWs using the Intel Atom Processor

DE2i-150 EE443 Supported Interfaces

- Libraries have been prepared for these peripherals:
 - 18 switches and 4 push-buttons (keys) for user input
 - 18 red and 9 green LEDs
 - UART interface to Matlab through RS232 port
 - Audio Codec (see AD/DA board)

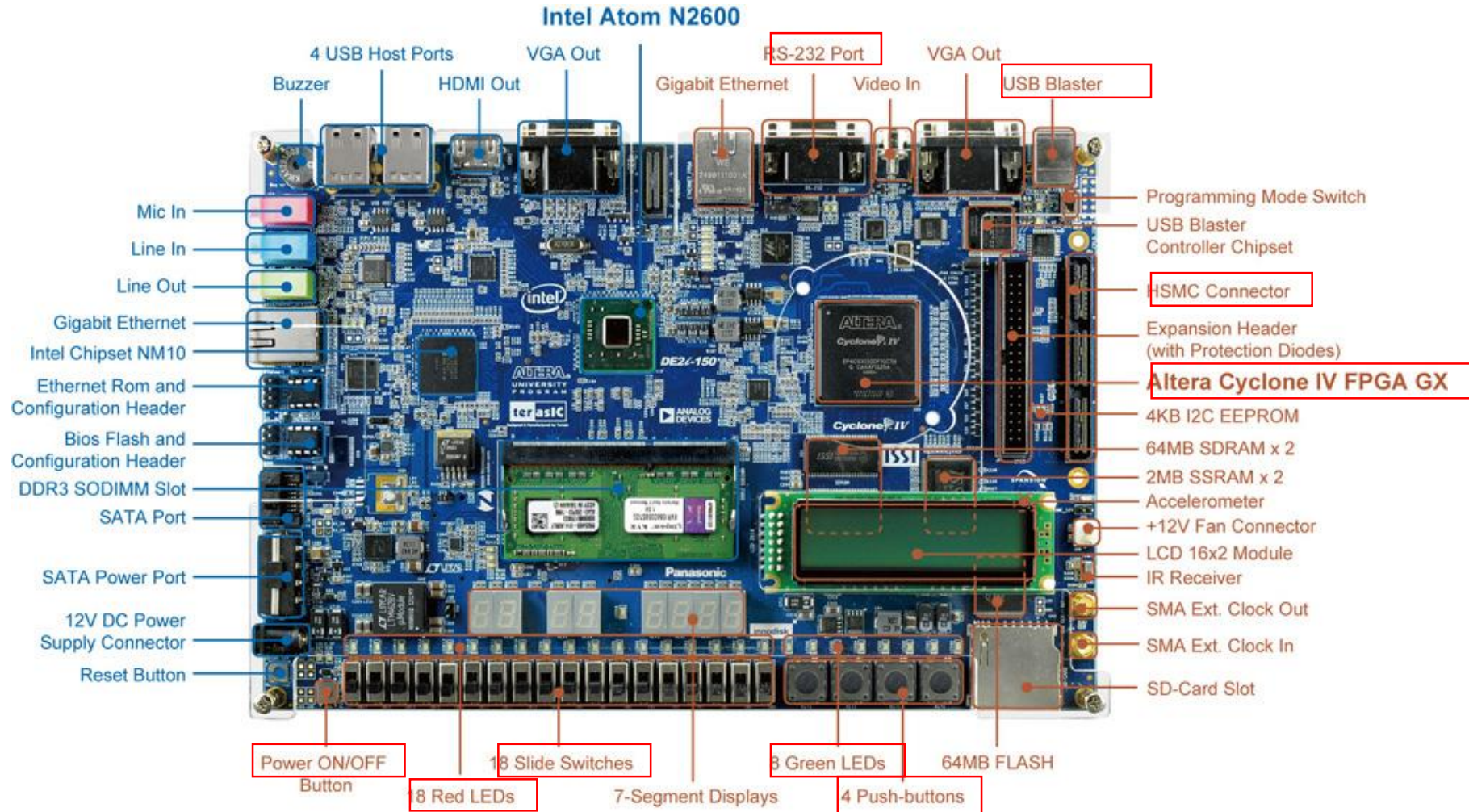
DE2i-150 Block Diagram



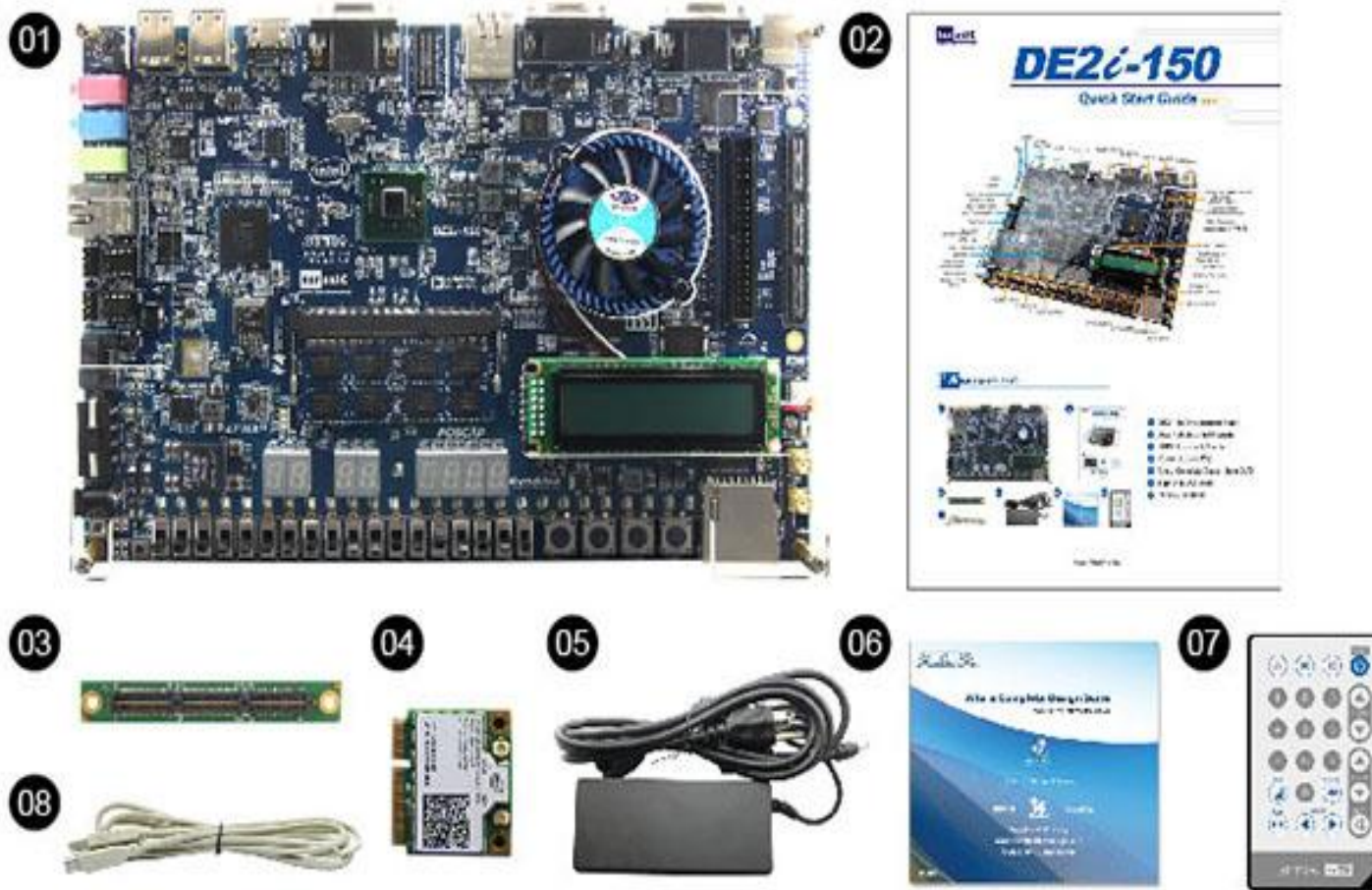
DE2i-150 EE443 Non-Supported Interfaces

- There are several components of this board from NIOS II libraries are available to you for your project. Just a few include:
 - The ENTIRE Intel Atom Processor side running Yocto (embedded Linux)
 - 7 segment displays
 - HDMI, VGA
 - Wifi, Ethernet
 - IR Receiver, Accelerometer
 - PCIe, USB

DE2i-150 Layout



What's in the Kit?



- 01 DE2i-150 Development Board
- 02 DE2i-150 Quick Start Guide
- 03 HSMC Loopback Adapter
- 04 Intel WiFi Module
- 05 Power Supply (12V)
- 06 Altera Complete Design Suite DVD
- 07 Remote Controller
- 08 Type A-B USB cable

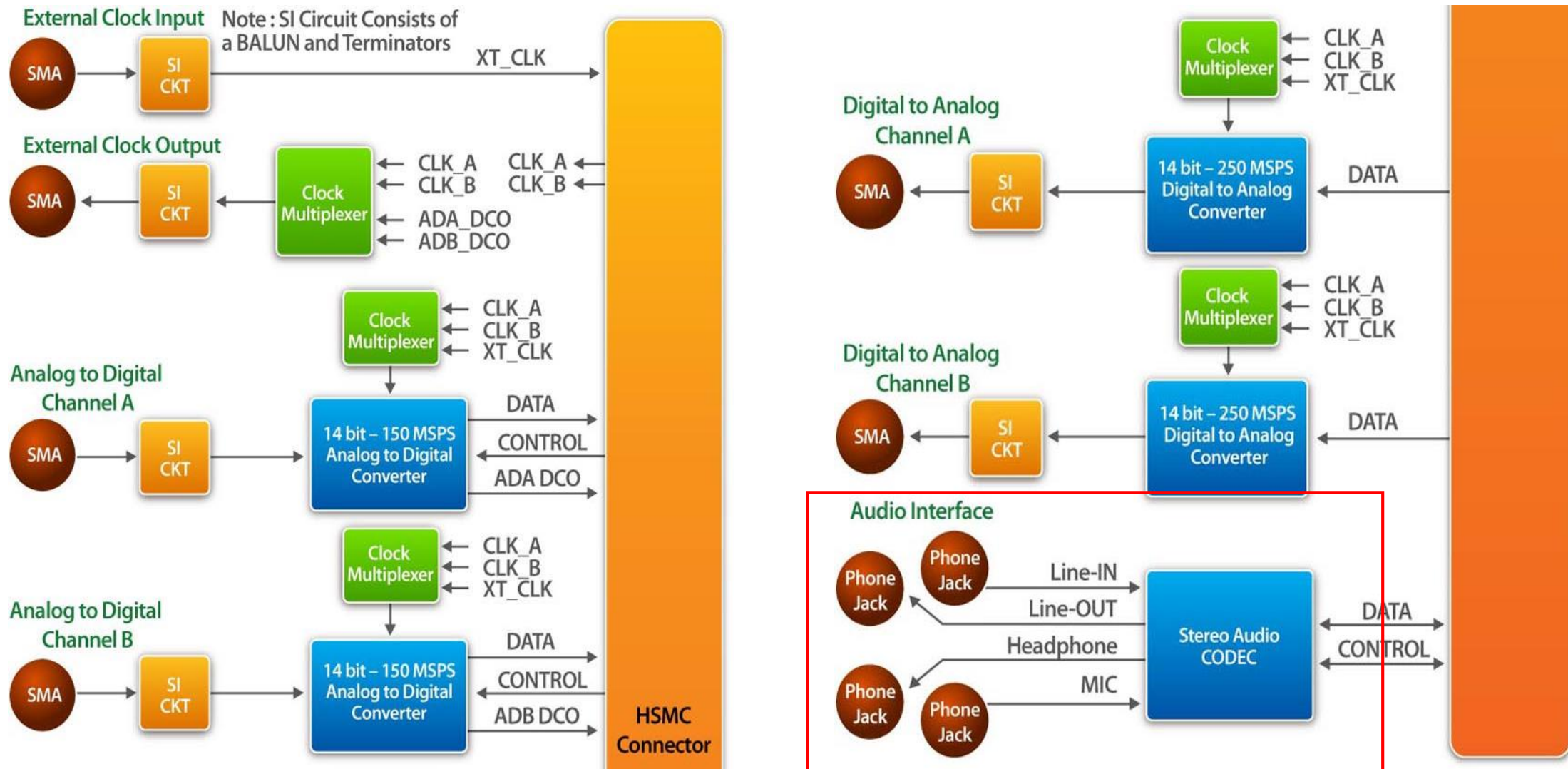
AD/DA Daughter Card: AIC23



AD/DA Daughter Card

- Plugs into DE2i-150 Development board via **High Speed Mezzanine Card (HSMC)** connector
- Adds Functionality
 - 2 Channel 14 bit ADC @ 150MS/s
 - 2 Channel 14 bit DAC @ 250MS/s
 - TI AIC23 Audio Codec
- SMA (surface-mount assembly) and Audio Jack input/output
- TI AIC23 will handle all of the audio input/output and data collection for this course

AD/DA Daughter Card Block Diagram



NIOS II Embedded Processor

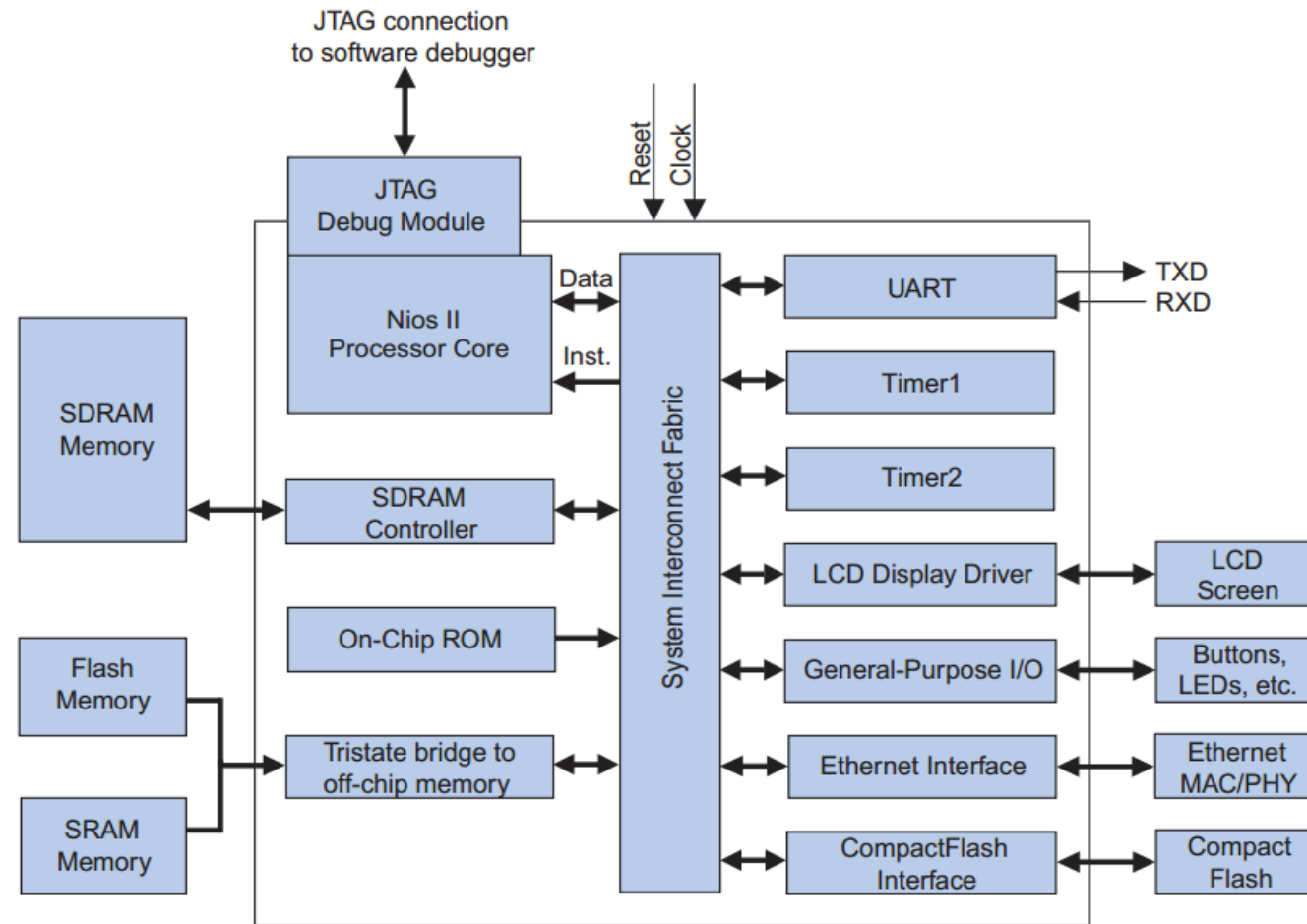
- Configurable processor unit with customizable peripherals
- Ability to program in C/C++ using Eclipse IDE
- **Nios II** is a RISC soft-core 32-bit embedded-processor architecture designed specifically for the Altera family of FPGAs. Nios II incorporates many enhancements over the original Nios architecture, making it more suitable for a wider range of embedded computing applications, from DSP to system-control.
- Nios II is comparable to MicroBlaze, a competing softcore CPU for the Xilinx family of FPGA. Unlike Microblaze, Nios II is licensable for standard-cell ASICs through a third-party IP provider, Synopsys Designware. Through the Designware license, designers can port Nios-based designs from an FPGA-platform to a mass production ASIC-device.

Development for Nios II

- **Hardware generation process**
- Nios II hardware designers use the Qsys system integration tool, a component of the Quartus-II package, to configure and generate a Nios system. The configuration graphical user interface (GUI) allows users to choose the Nios-II's feature-set, and to add peripheral and I/O-blocks (timers, memory-controllers, serial interface, etc.) to the embedded system. When the hardware specification is complete, Quartus-II performs the synthesis, place & route to implement the entire system on the selected FPGA target.
- **Software creation process**
- A separate package, called the Embedded Design Suite (EDS), manages the software development. Based on the [Eclipse](#) IDE, the EDS includes a C/C++ compiler (based on the [GNU toolchain](#)), debugger, and an instruction-set simulator. EDS allows programmers to test their application in simulation, or download and run their compiled application on the actual FPGA host.

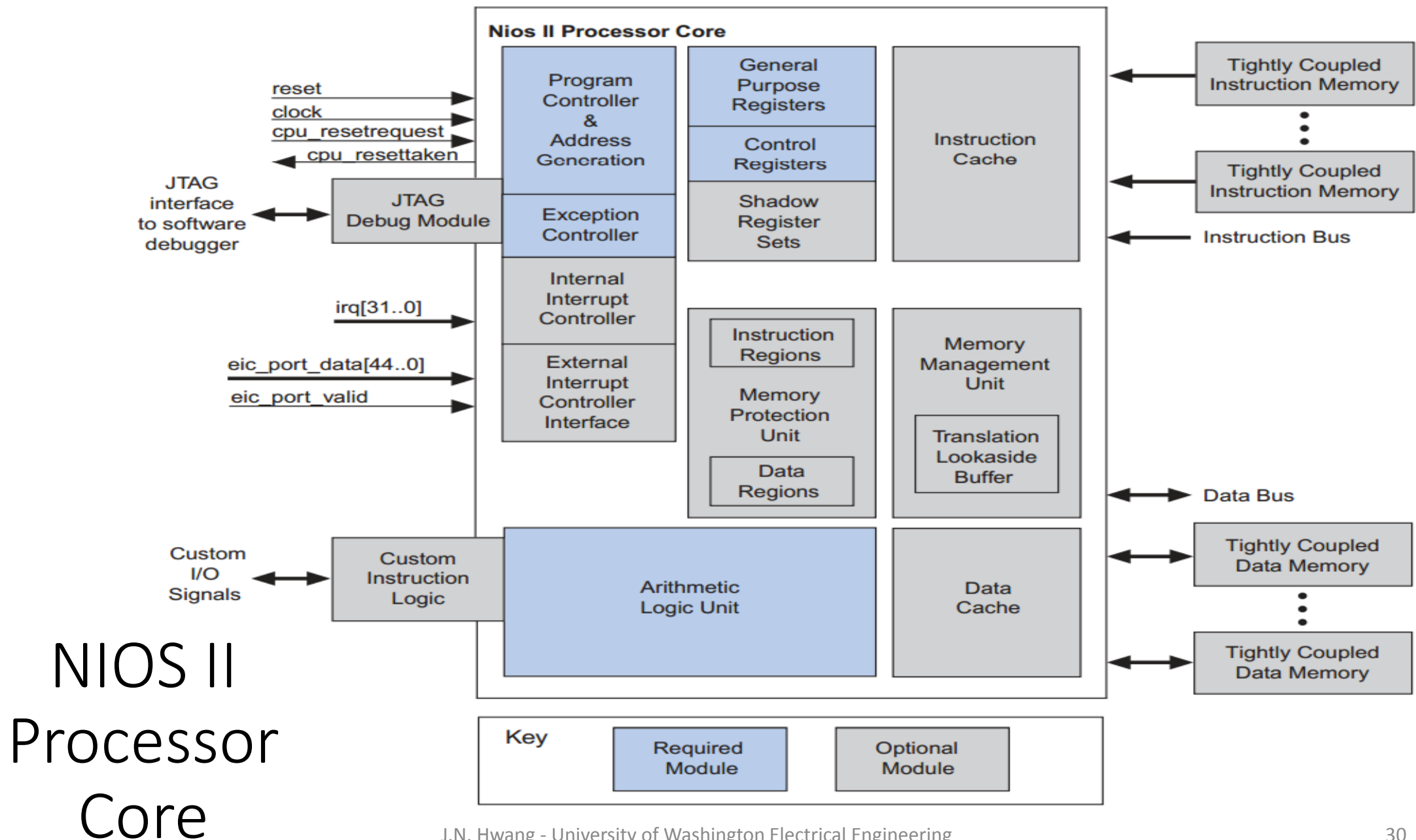
NIOS II Embedded Processor

Figure 1-1. Example of a Nios II Processor System



http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf

Figure 2–1. Nios II Processor Core Block Diagram



Basic Functions EE443

- Parallel Input/Output
 - LEDs, KEYS (buttons), SWITCHES
- AIC23 Audio Codec Control
- UART-RS232 Data Transmission
- DSP Algorithms
 - FFT, FIR, IIR, etc
- HAL API
 - Interrupts

Parallel I/O Peripherals

- Peripherals that pass data to the NIOS processor in parallel
- Common Functions:
 - `IORD_ALTERA_AVALON_PIO_DATA(BASE);` // Read
 - `IOWR_ALTERA_AVALON_PIO_DATA(BASE, DATA);` // Write
 - `IORD_ALTERA_AVALON_PIO_EDGE_CAP(BASE);` // Edge of Capture
 - `IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BASE, DATA);` // Resets Edge Capture
- BASE – The base address of the peripheral. e.g. LED_BASE, SWITCH_BASE
- DATA – The data to be sent to the base. e.g. 0x01C3F
- BASEs for the different peripherals can be found in the file “system.h”
- EDGE_CAP is to be used with interrupt applications

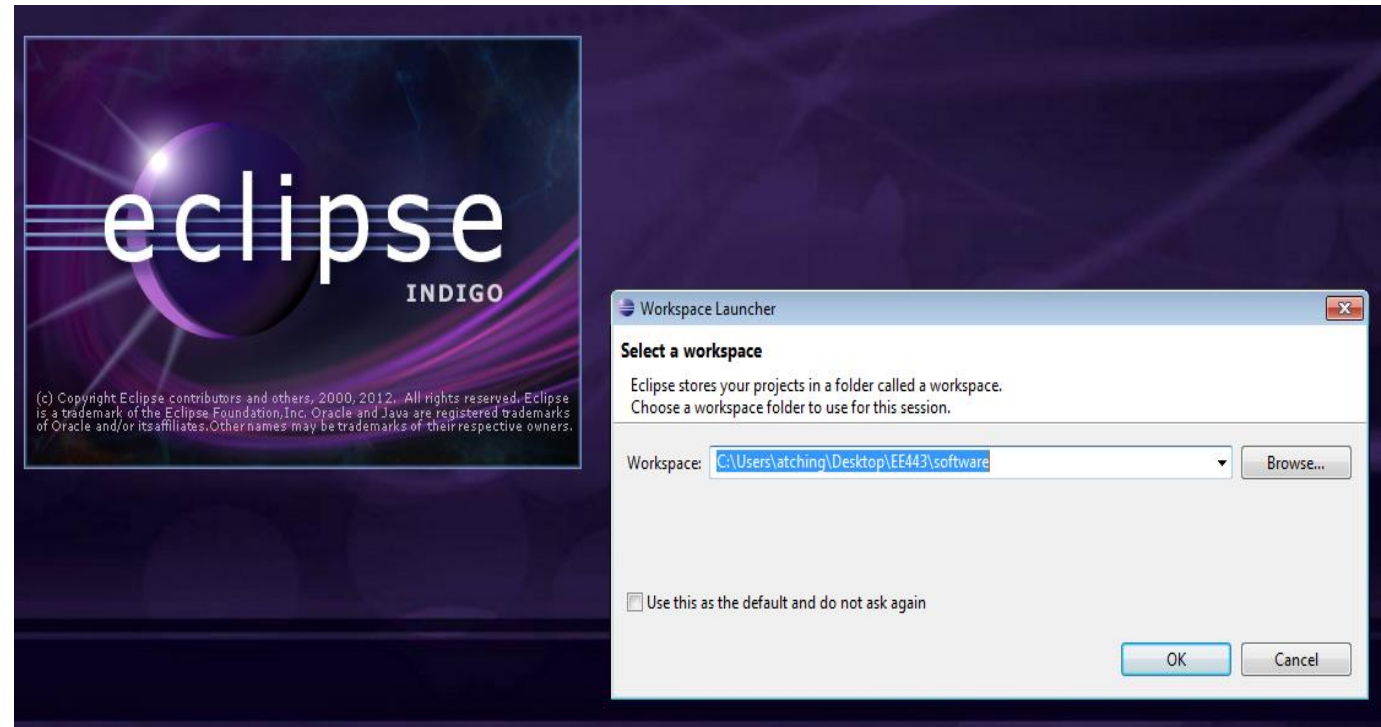
Basic Development Example

- Developing on NIOS II Platform
- Interfacing with DE2i-150 board Peripherals
- Interfacing with Matlab or LabVIEW for GUI
- Debugging

Developing on NIOS II

<http://www.altera.com/devices/processor/nios2/tools/ide/ni2-ide.html>

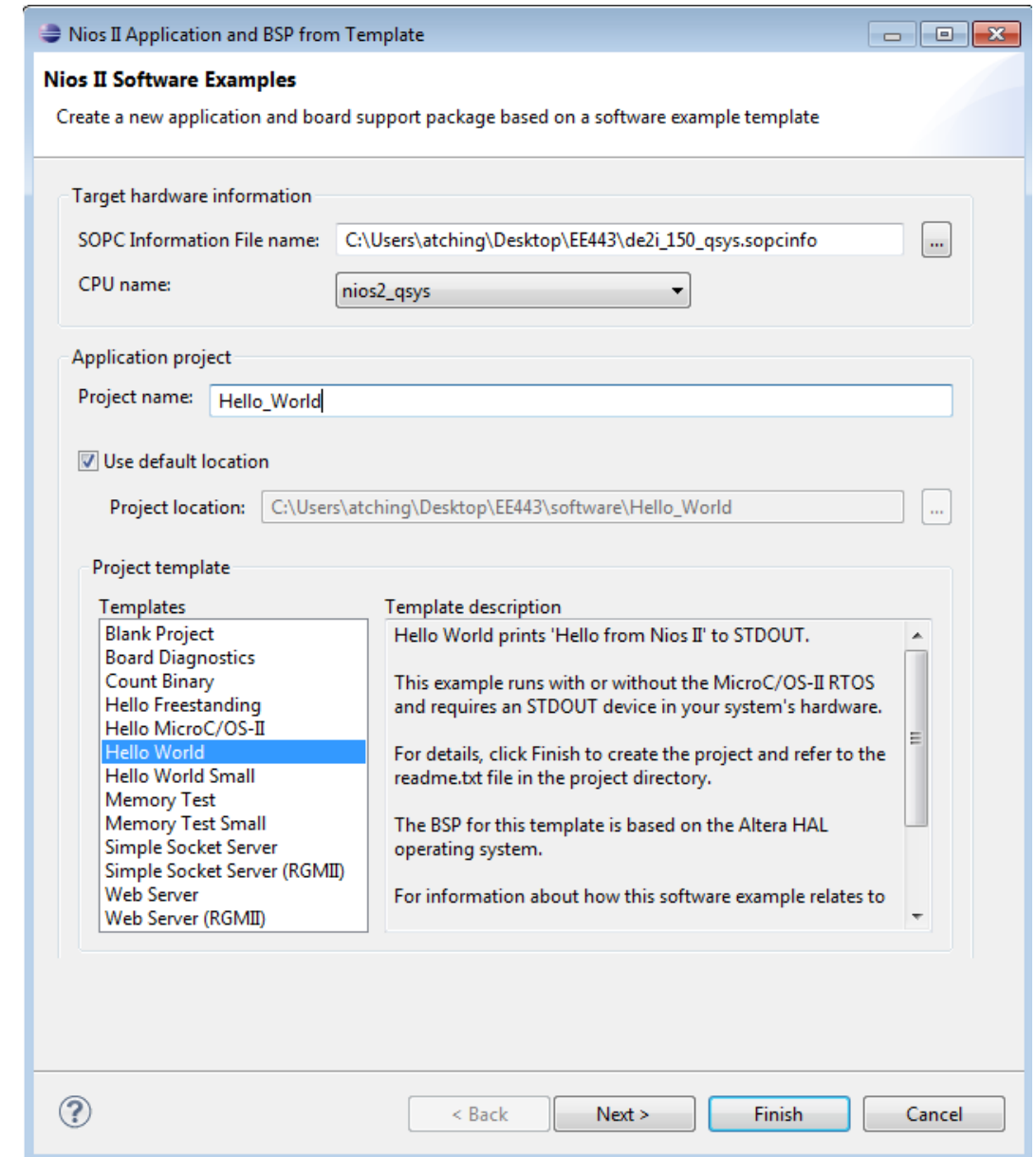
- Open **Nios II 13.0sp1 Software Build Tools (SBT) for Eclipse**
- Select **EE443\software** folder
- You will develop all software projects within this folder
- It contains all of the libraries and configurations made for this board.
- Can choose another workspace as long as you include those files



- New project wizards and software templates
- Compiler for C and C++ (GNU)
- Source navigator, editor, and debugger
- Eclipse project-based tools
- Software build tools

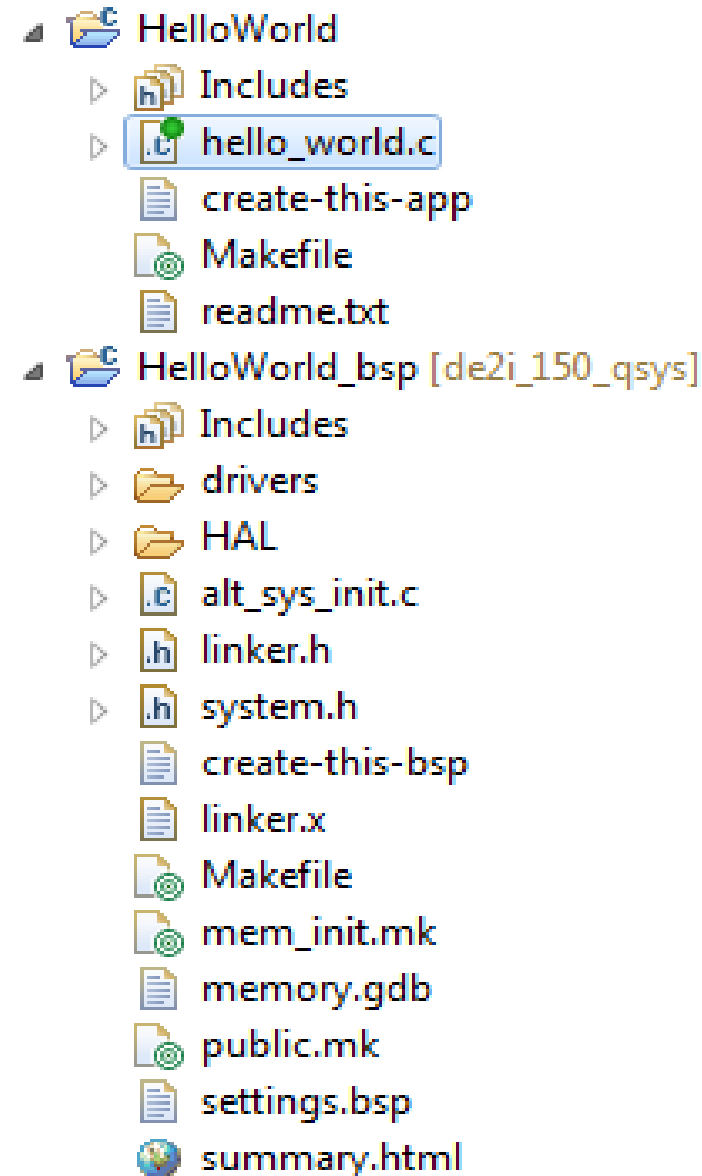
Your First Program

- File > New > NIOS II Application and BSP (board support package) from Template
- Select the **de2i_150_qsys.sopcinfo** file
- Choose the Hello World template
- Name your project
- Click finish



Setup

- Open up the **hello_world.c** file
- You can always select this template in the menu and change the file names after the project is created
- Import all libraries from ee443_libs folder
- Right click the project folder and click “Build Project”



//UART_example.c 128-length data transmission from the board to PC (Matlab)

#define PI 3.141592

//pre-define PI = 3.141592

#include "system_init.h"

//table index

void main()

{

int counter;

int Data[128];

sampleFrequency = 0x000C;

system_initialization();

for(int ii=0; ii<128; ii++) Data[ii] = (int)1000*sin(2*PI*ii/128);

while(1)

{

if(uartStartSendFlag){

for (counter = 0; counter < 128; counter++){

uart_sendInt16(Data[counter]);

}

uartStartSendFlag = 0;

}

}

}

Press the push button #0,
which should send the 128-
length integer data stored in
the Data buffer through UART.

//8k

//init switches and UART

//sample data to transmit

//infinite loop

//monitor the push button (defined in yourISR.h)

//send the data through UART

//finish the transmission

//end of while (1) infinite loop

//end of main

```
%% PC-side (Matlab): receiving data from UART

close all; clc; clear all;

s = serial('COM3', 'BaudRate',115200); % Open the serial port to receive the data
set(s,'InputBufferSize',20000); % set the size of input buffer
fopen(s); % get ready to receive the data
buffersize = 128; % set the size of instant read of buffer
a = fread(s,buffersize,'int16'); % read the buffer when data arrive

plot(a);
fclose(s);
delete(s);
'done'
```

Adding Peripherals

- Add `#include "system.h"` to your program to import the peripherals on the DE2i board
- Add the appropriate control libraries for your peripherals
- This example will be using the `switches, LEDs, and UART`
- Refer to the examples in the EE443 lab manual and online material from Altera for specifics

```
#include <stdio.h>
#include "system.h"
#include "altera_avalon_uart_regs.h"
#include "altera_avalon_pio_regs.h"
```

Example

- Your program will run in an infinite loop within `while(1)` in your `main()` function
- This example will turn the **first 3 green LEDs on/off** based on the **first 3 switches** (via PIO)
- The **4th switch** will send “**Hello World**” through the RS232 (UART) port to Matlab to print on the screen
- The **5th switch** will send “**012345**” to Matlab


```

printf("Hello from Nios II!\n");
char hello[11] = "Hello World";
int i = 0;

while(1){

    // Checks if switch 0 is on
    if(IORD_ALTERA_AVALON_PIO_DATA(SWITCH0_BASE)){
        IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, IORD_ALTERA_AVALON_PIO_DATA(LED_BASE) | 0x01); // Turns on 1st LED
    } else{
        IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, IORD_ALTERA_AVALON_PIO_DATA(LED_BASE) & ~0x01); // Turns off 1st LED
    }

    // Checks if switch 1 is on
    if(IORD_ALTERA_AVALON_PIO_DATA(SWITCH1_BASE)){
        IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, IORD_ALTERA_AVALON_PIO_DATA(LED_BASE) | 0x02); // Turns on 2nd LED
    } else{
        IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, IORD_ALTERA_AVALON_PIO_DATA(LED_BASE) & ~0x02); // Turns off 2nd LED
    }

    // Checks if switch 2 is on
    if(IORD_ALTERA_AVALON_PIO_DATA(SWITCH2_BASE)){
        IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, IORD_ALTERA_AVALON_PIO_DATA(LED_BASE) | 0x04); // Turns on 3rd LED
    } else{
        IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, IORD_ALTERA_AVALON_PIO_DATA(LED_BASE) & ~0x04); // Turns off 3rd LED
    }

    // Checks if switch 3 is on
    if(IORD_ALTERA_AVALON_PIO_DATA(SWITCH3_BASE)){
        for(i = 0; i < sizeof(hello); i++){
            IOWR_ALTERA_AVALON_UART_TXDATA(UART_BASE, hello[i]); // Send individual characters over UART to Matlab
            delay(2000); // Delay necessary to slow data transmission to within UART speeds of 115200kpbs
        }
        IOWR_ALTERA_AVALON_UART_TXDATA(UART_BASE, 0x0A); // Matlab requires an end of line character to terminate transmission
        delay(2000);
    }

    // Checks if Switch 4 is on
    if(IORD_ALTERA_AVALON_PIO_DATA(SWITCH4_BASE)){
        for(i = 0; i < 6; i++){
            IOWR_ALTERA_AVALON_UART_TXDATA(UART_BASE, (i + 48)); // Send ASCII equivalent of 012345 over UART to Matlab
            delay(2000);
        }

        IOWR_ALTERA_AVALON_UART_TXDATA(UART_BASE, 0x0A); // Matlab requires an end of line character to terminate transmission
        delay(2000);
    }

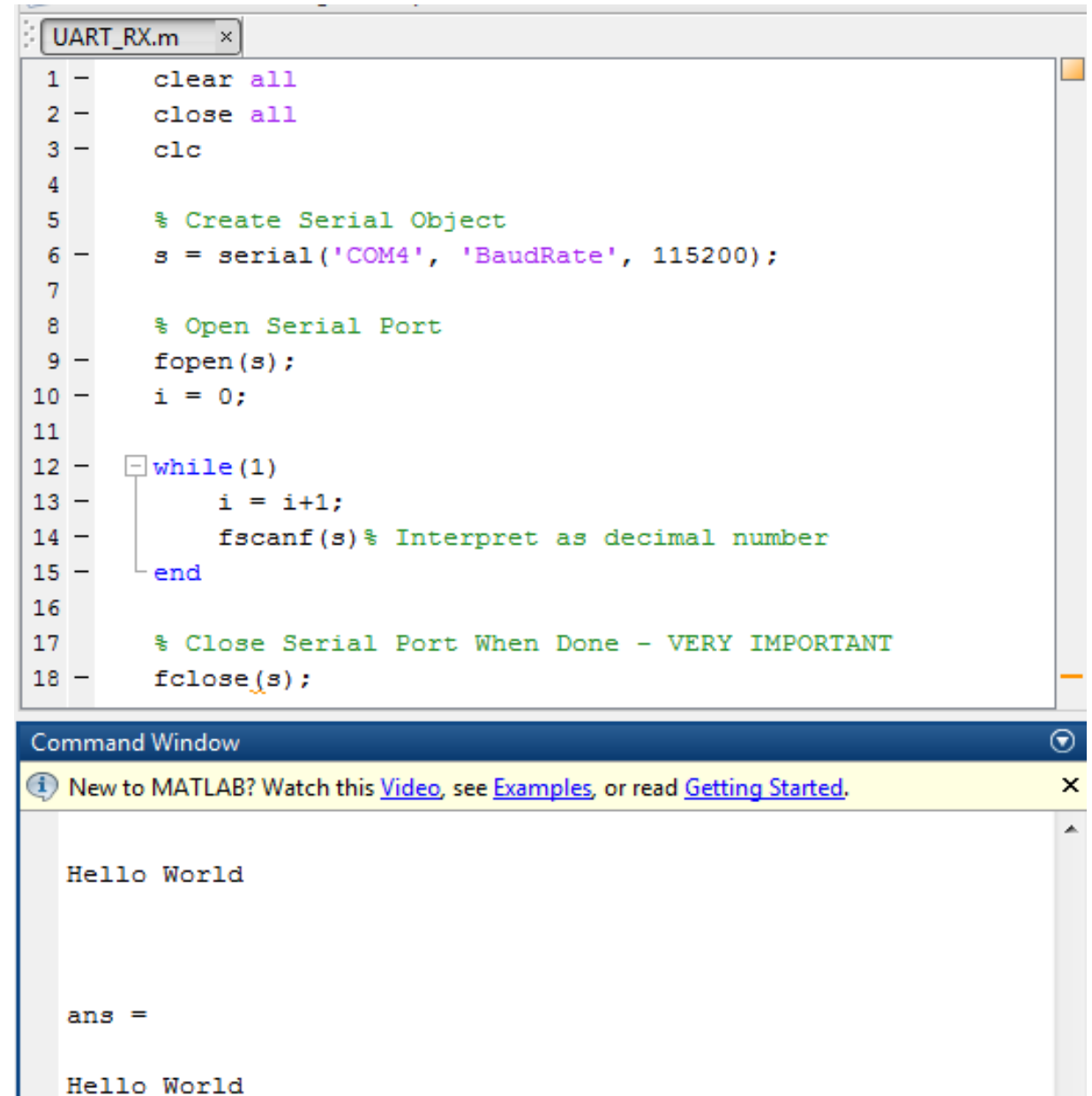
}

```

ASCII 0=48, 1=49,

MATLAB

- Create a serial object at the COM port via USB cable, connected to COM4 here, to expect the data from UART transmission
- **fopen(<serial object>)** to open the communication channel
- **fscanf(s)** reads the raw character data
- You can use MATLAB to interpret the data as decimal, float, etc using **fprintf()**
- You need a termination character in the NIOS II code (0x0A)



The image shows a MATLAB script named 'UART_RX.m' and its execution results in the Command Window. The script sets up a serial connection to COM4 at 115200 baud, opens the port, and enters a loop to read data from the serial port. The Command Window shows the output 'Hello World' and the variable 'ans' containing the same string.

```
UART_RX.m x
1 - clear all
2 - close all
3 - clc
4
5 - % Create Serial Object
6 - s = serial('COM4', 'BaudRate', 115200);
7
8 - % Open Serial Port
9 - fopen(s);
10 - i = 0;
11
12 - while(1)
13 -     i = i+1;
14 -     fscanf(s) % Interpret as decimal number
15 - end
16
17 - % Close Serial Port When Done - VERY IMPORTANT
18 - fclose(s);
```

Command Window

New to MATLAB? Watch this [Video](#), see [Examples](#), or read [Getting Started](#).

```
Hello World

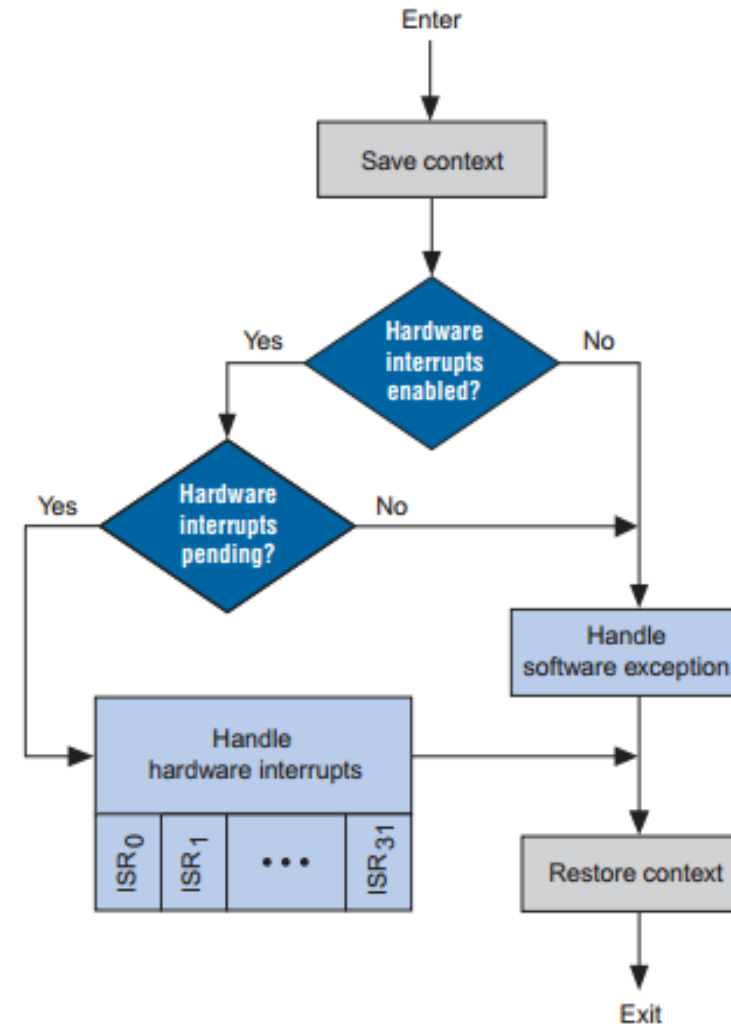
ans =

Hello World
```

Interrupt Example

- Interrupts happen outside the while(1) loop when an event is triggered (like a switch) and pause the CPU to execute, then return to normal CPU execution
- NIOS II uses the hardware abstraction layer (**HAL**) API to register and control interrupt handling
- Example will use KEY0 to turn LED 3 ON and KEY1 to turn LED3 OFF

Figure 8–1. HAL Exception Handling System with the Internal Interrupt Controller



```
#include "sys/alt_irq.h"
#include "priv/alt_legacy_irq.h"
#include "os/alt_hooks.h"
#include "alt_types.h"
#include "sys/alt_irq_entry.h"
#include "priv/alt_irq_table.h"
#include "sys/alt_irq.h"
```

Include HAL API Interrupt Libraries

```
// OR #include system_init.h for everything
```

```
// Global Variables
alt_u32 key0_id = KEY0_IRQ;
volatile int key0 = 0;
```

```
alt_u32 key1_id = KEY1_IRQ;
volatile int key1 = 0;
```

```
// Interrupt Handler for KEY0
static void handle_key0_interrupt(void* context, alt_u32 id) {
    volatile int* key0ptr = (volatile int *)context;
    *key0ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEY0_BASE);

    /* Write to the edge capture register to reset it. */
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEY0_BASE, 0);

    IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, IORD_ALTERA_AVALON_PIO_DATA(LED_BASE) | 0x08); // Turns on 4th LED
}
```

```
// Interrupt Handler for KEY1
static void handle_key1_interrupt(void* context, alt_u32 id) {
    volatile int* key1ptr = (volatile int *)context;
    *key1ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEY1_BASE);

    /* Write to the edge capture register to reset it. */
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEY1_BASE, 0);

    IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, IORD_ALTERA_AVALON_PIO_DATA(LED_BASE) & ~0x08); // Turns off 4th LED
}
```

```
int main()
{
    // Interrupt setup. Mostly DONE for you in library. This is just example so
    // you can write your own if you want to.
    alt_irq_register(key0_id, (void *)&key0, handle_key0_interrupt); // Register interrupt
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(KEY0_BASE, 1); // Enable Interrupt
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEY0_BASE, 0); // Reset Edge Capture Bit

    alt_irq_register(key1_id, (void *)&key1, handle_key1_interrupt); // Register interrupt
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(KEY1_BASE, 1); // Enable Interrupt
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEY1_BASE, 0); // Reset Edge Capture Bit
}
```

```
printf("Hello from Nios II!\n");
char hello[11] = "Hello World";
int i = 0;

while(1){
```

Cast the IRQ IDs

Interrupt Handler for KEY0

Interrupt Handler for KEY1

Register & Initialize
Interrupts

Rest of the program

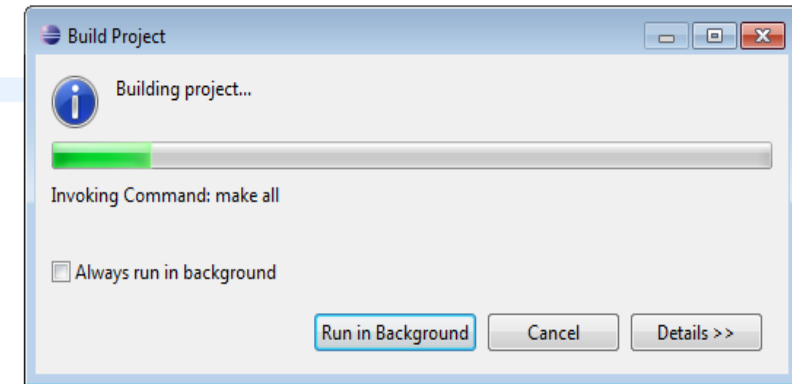
Build Project

- Build the program to compile the code so it is ready to be uploaded to the DE2i-150 board

```
#include <stdio.h>

int main()
{
    printf("Hello from Nios II!\n");

    return 0;
}
```



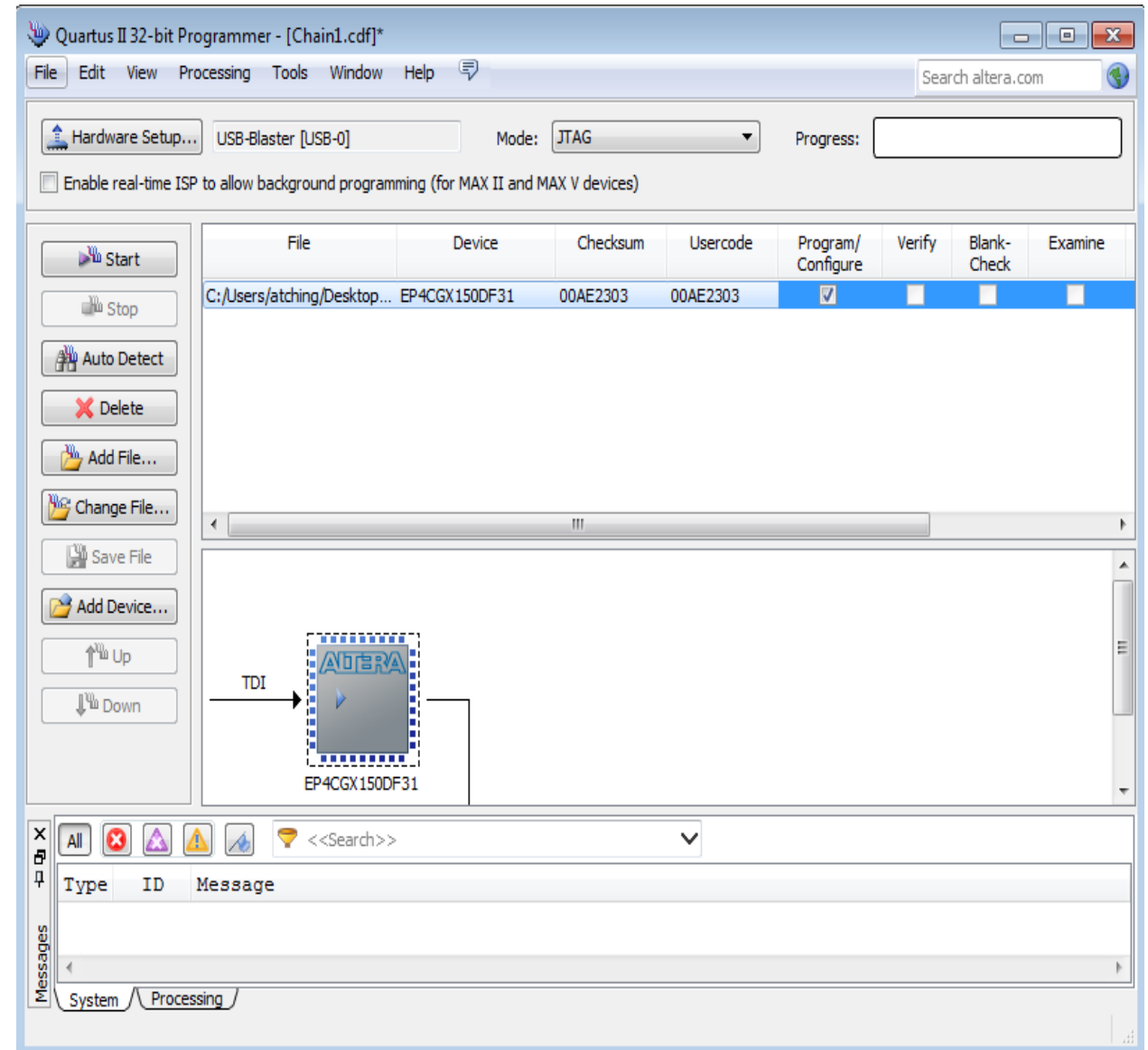
Problems Tasks Console Properties Search

Build Console [HelloWorld]

```
j/HAL/src/alt_rename.o HAL/src/alt_rename.c
mpiling alt_sbrk.c...
ps2-elf-gcc -xc -MP -MMD -c -I./HAL/inc -I. -I./drivers/inc -DSYSTEM_BUS_WIDTH=32 -pipe -D_hal__ -DALT_NO_INSTRUCTION_EMULATION -DALT_SINGLE_TH
j/HAL/src/alt_sbrk.o HAL/src/alt_sbrk.c
mpiling alt_settod.c...
ps2-elf-gcc -xc -MP -MMD -c -I./HAL/inc -I. -I./drivers/inc -DSYSTEM_BUS_WIDTH=32 -pipe -D_hal__ -DALT_NO_INSTRUCTION_EMULATION -DALT_SINGLE_TH
j/HAL/src/alt_settod.o HAL/src/alt_settod.c
mpiling alt_software_exception.S...
ps2-elf-gcc -MP -MMD -c -O0 -g -Wall -EL -mno-hw-div -mhw-mul -mno-hw-mulx -I./HAL/inc -I. -I./drivers/inc -DSYSTEM_BUS_WIDTH=32 -pipe -D_
obj/HAL/src/alt_software_exception.o HAL/src/alt_software_exception.S
mpiling alt_stat.c...
ps2-elf-gcc -xc -MP -MMD -c -I./HAL/inc -I. -I./drivers/inc -DSYSTEM_BUS_WIDTH=32 -pipe -D_hal__ -DALT_NO_INSTRUCTION_EMULATION -DALT_SINGLE_TH
j/HAL/src/alt_stat.o HAL/src/alt_stat.c
mpiling alt_tick.c...
ps2-elf-gcc -xc -MP -MMD -c -I./HAL/inc -I. -I./drivers/inc -DSYSTEM_BUS_WIDTH=32 -pipe -D_hal__ -DALT_NO_INSTRUCTION_EMULATION -DALT_SINGLE_TH
j/HAL/src/alt_tick.o HAL/src/alt_tick.c
mpiling alt_times.c...
ps2-elf-gcc -xc -MP -MMD -c -I./HAL/inc -I. -I./drivers/inc -DSYSTEM_BUS_WIDTH=32 -pipe -D_hal__ -DALT_NO_INSTRUCTION_EMULATION -DALT_SINGLE_TH
i/HAL/src/alt_tick.o HAL/src/alt_tick.c
```

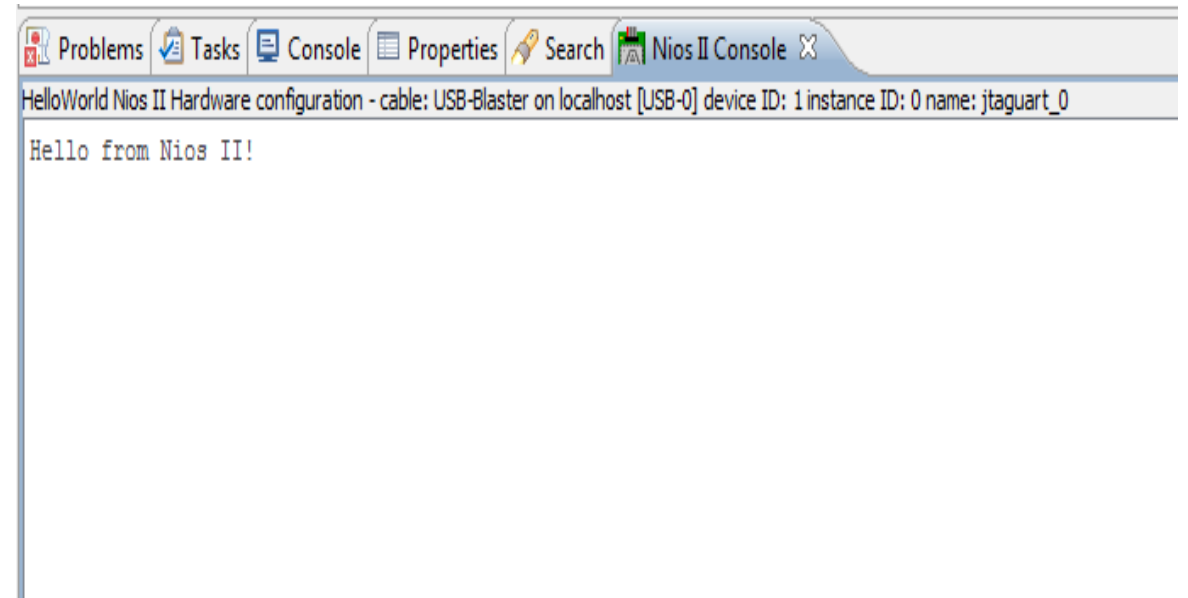
NIOS II Quartus Programmer

- NIOS II > Quartus II Programmer
- Make sure your DE2i-150 board is plugged into the USB port
- Hit Auto Detect
- Add File and select the EE443.sof file located in the EE443 folder
- Check Program/Configure
- Hit Start
- This configures FPGA to the NIOS II processor so you can use C



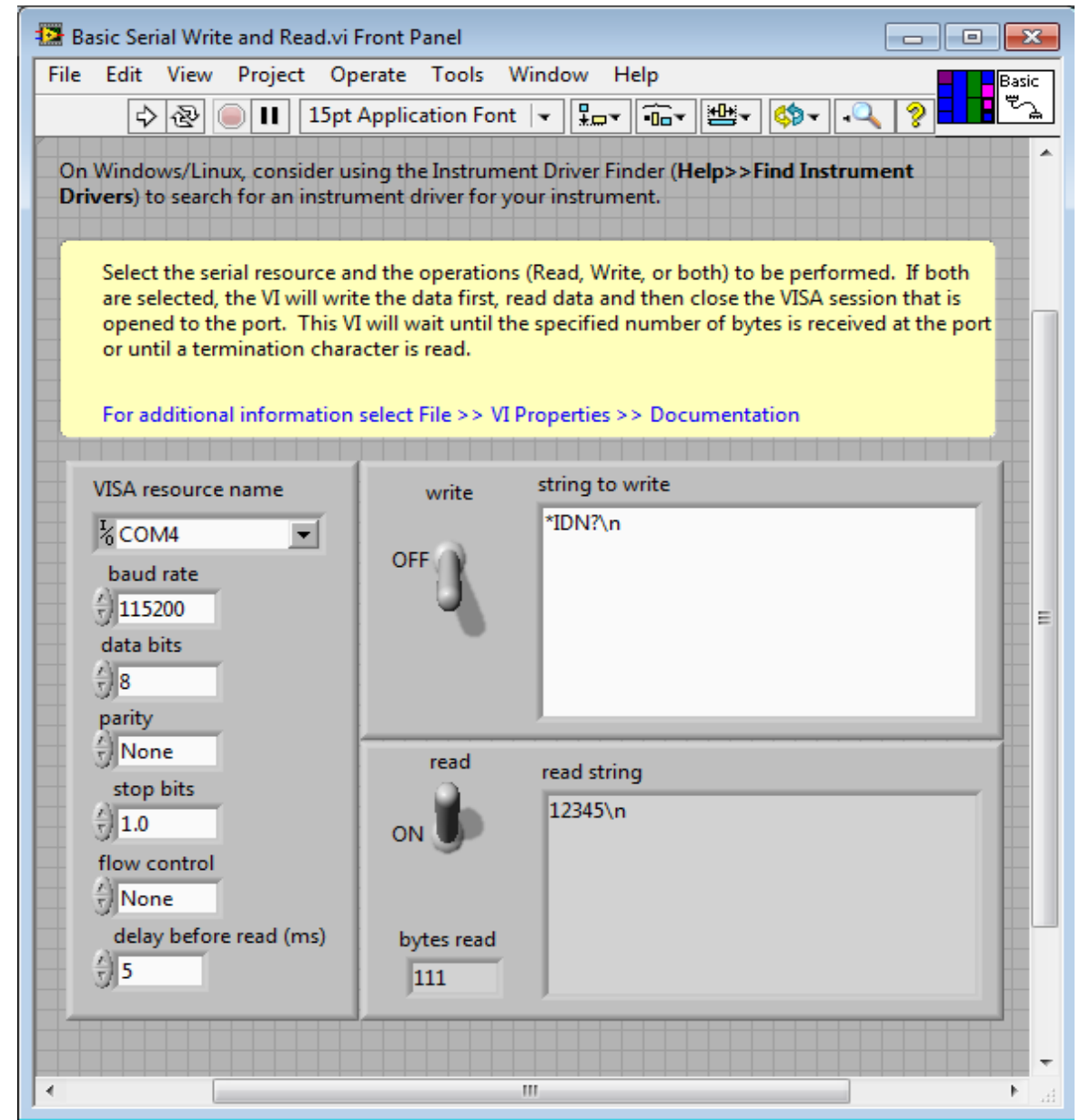
Running Program on DE2i-150

- Right click project folder
- Select Run As > NIOS II Hardware
- It should print out the words in the NIOS II Console tab at the bottom of the NIOS II Eclipse IDE
- Use the switches and keys to control the LEDs
- Run the Matlab program to print received data from the UART




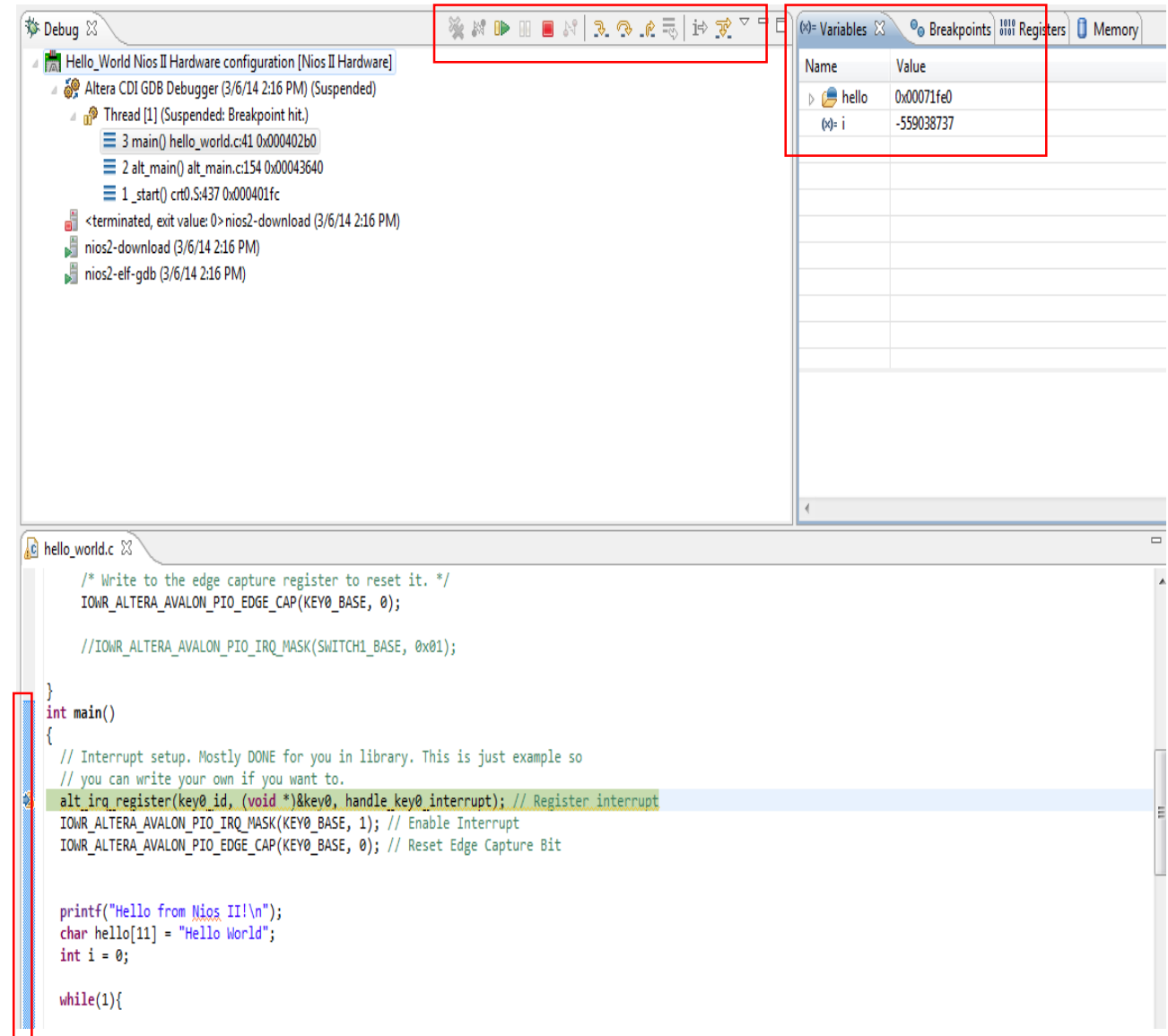
LabVIEW

- Locate the **Basic Serial Write and Read.vi** LabVIEW example from the Help Menu
- Not as accurate as MATLAB when tested but LabVIEW is much better at GUI creation and plotting support
- You can find better examples online
- Up to you to choose which one to use



Debugger

- Use debugger to **step** through the execution of your code and **check variables, registers**, etc for bugs
- Hit the  icon instead of the run icon to enter Debug Mode.
- You can then use the controls to step through your code
- Add breakpoints to pause code during execution



Bugs

- If your board becomes unresponsive, **reload the EE443.sof** file using the Quartus II Programmer to **reset** it.
- Then reprogram the NIOS II Project onto the board
- Will fix most unresponsive programs
- If you don't **close the serial COM port in Matlab** when finished, you will have to restart Matlab

Yocto



- Embedded Linux platform called Yocto on the Intel Atom
- Has a graphical interface if you plug into a monitor via VGA
- Can use mouse and keyboard
- Has **gcc compiler** for C programs and basic programs installed
- No Desktop
- Remote access via PuTTY and VNC
- Can play .wav files and input into AIC23 to process