Programming the NIOS II processor and the accelerometer

Aaron Zhao, Imperial College London

An Introduction to Lab2

What is in this lab?

- Design a NIOSII system.
- Understand the design process of a NIOSII system.
- Program the Max 10 FPGA chip on the DE10-Lite board with your soft processor.
- Write code that runs on the NIOS processor to display a message on a terminal.
- Explore and test the capabilities of your NIOS II system design.

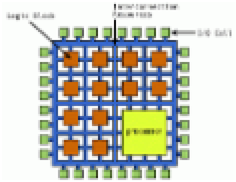
Soft Vs. Hard processors

- Hard processors normally have a fixed architecture and is normally faster
  - eg. ARM, x86
- NIOSII is a soft processor
  - customizable (*eg.* size, performance)
  - add or remove certain features (*eg.* floating-point units)
  - custom instructions or extensions of the instruction set
- Soft processors are normally on FPGAs, running at a slower clock rate

Soft Vs. Hard processors (ii)

Hard Processors (eg. Intel, ARM, AMD)

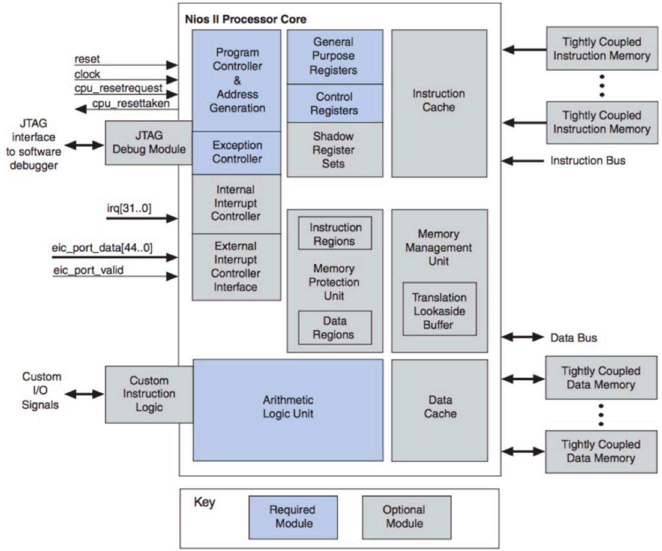
Soft Processors (MicroBlaze, NIOS II)



Processor architecture

- Register files
- Arithmetic logic unit (ALU)
- iCache and dCache
- Instruction decoding
- Instruction bus and data bus

Processor architecture (ii)

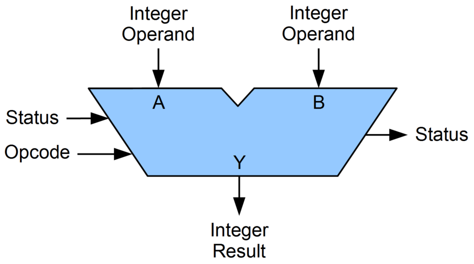


Processor architecture - Register files

- The ISA (Instruction Set Architecture) will always define a set of registers which are used to store and load data between memory and the functional units on the chip.
- Register file is a set of registers that can be accessed by the processor.
- Different types of registers
  - Data storage (eg. R0-R7 in ARM)
  - Instruction control (eg. PC, program counter)
  - Special status and control registers (eg. carry, zero and overflow flags)
- Possible optimizations
  - Register renaming

Processor architecture - ALU

- Normally a piece of hardware that performs arithmetic and logical operations, it normally takes
  - two inputs
  - an op-code
  - produces one output
  - maybe status in and outputs



Processor architecture - Cache system

- There are many design choices for the cache system
- Cache size, associativity, and replacement policy
- Data cache and instruction cache
- The general idea is to store frequently accessed data in the cache (which is closer to the processing unit) to reduce the time it takes to access the data

Processor architecture - Cache system

This can soon become very complex, especially when you consider the different levels of cache that are present in modern multi-processors.:

- Caching complicates the ordering of all operations
  - A memory location can be present in multiple caches
  - How can all processors see the same value? Or how can they see the same global order of all memory operations?
- It also affects the ordering of operations on a single memory location
  - A single memory location can be present in multiple caches
  - Makes it difficult for processors that have cached the same location to have the correct value of that location (in the presence of updates to that location)

Consistency and Coherence

- Memory consistency: Global ordering of accesses to all memory locations, it matters more to the programmer to reason about correctness in parallel systems.
- Cache Coherence: Local ordering of accesses to each cache block

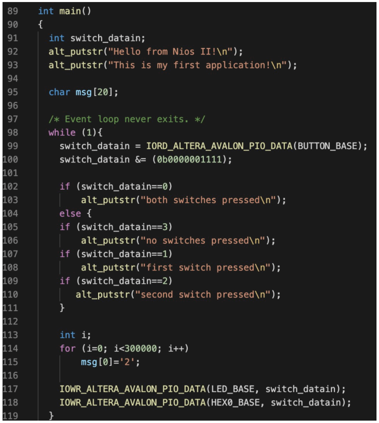
The processor you will use

- Intel’s NIOSII soft-core processor
- A NIOSII implementation is a set of design choices
- A functional unit (i.e. fp-mult) can be implemented in
  - hardware
  - emulated in software
  - omitted completely…
- Another example is division support

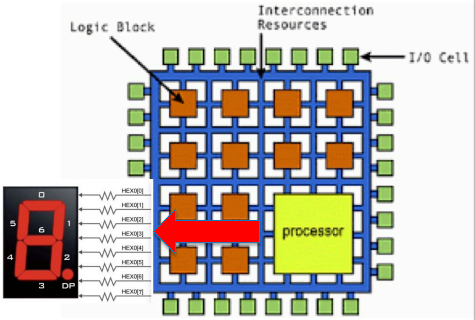
What do you do in Lab2?

- Instantiate a NIOSII processor in Qsys
- This means you use Intel's IP to create a NIOSII processor on the FPGA device. You do not need to design the processor from scratch.
- This is equivalent to writing System Verilog code to instantiate a processor in an FPGA. Actually, there are a few open-source soft processors that you can instantiate in an FPGA. One example is the RISC-V processor (Muntjac https://github.com/lowRISC/muntjac).
- Control the lighting sequence on LEDs through the NIOSII processor
- The NIOSII processor will be connected to the LEDs on the DE10-Lite board. You will write code that will control the lighting sequence on the LEDs.
- This is basically very similar to writing a program in C that controls the GPIO pins on a Raspberry Pi.

What do you do in Lab2? (ii)



```
int main()
{
  int switch_datain;
  alt_putstr("Hello from Nios II!\n");
  alt_putstr("This is my first application!\n");

  char msg[20];

  /* Event loop never exits. */
  while (1){
    switch_datain = IORD_ALTERA_AVALON_PIO_DATA(BUTTON_BASE);
    switch_datain &= (0b0000001111);

    if (switch_datain==0)
        alt_putstr("both switches pressed\n");
    else {
    if (switch_datain==3)
        alt_putstr("no switches pressed\n");
    if (switch_datain==1)
        alt_putstr("first switch pressed\n");
    if (switch_datain==2)
       alt_putstr("second switch pressed\n");
    }

    int i;
    for (i=0; i<300000; i++)
        msg[0]='2';

    IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, switch_datain);
    IOWR_ALTERA_AVALON_PIO_DATA(HEX0_BASE, switch_datain);
  }
```



An Introduction to Lab3

What is in this lab?

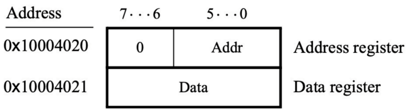
- A typical IoT scenario, where a sensor is connected to a processor.
- Design a NIOS II system that interfaces with the accelerometer on DE10-lite board.
- Understand the SPI interface.
- Learn how to read the acceleration value provided by the accelerometer.
- Design a low-pass FIR filter to process the readings.
- Investigate the impact of using low arithmetic precision to the quality of the results and the performance of your system.

Accelerometer

- Analog Devices' ADXL345 chip
- 3-axis accelerometer, it measures acceleration in three directions, which are referred to as x-axis, y-axis and z-axis.
- Serial Peripheral Interface (SPI) / I2C
- 16-bit digital output

How does NIOS interact with the accelerometer

- Add accelerometer_spi IP
  - IP controls the accelerometer and provides an SPI interface to NIOS
  - 58 internal registers
  - Memory mapped through two 8-bit registers: Address and Data
  - Memory mapped means they are mapped to specific memory addresses at the time the core is instantiated in a Qsys-developed system.



Address	Register Name	Description
0x32	*DATAX0*	Low-order byte of *x*-axis acceleration.
0x33	*DATAX1*	High-order byte of *x*-axis acceleration.
0x34	*DATAY0*	Low-order byte of *y*-axis acceleration.
0x35	*DATAY1*	High-order byte of *y*-axis acceleration.
0x36	*DATAZ0*	Low-order byte of *z*-axis acceleration.
0x37	*DATAZ1*	High-order byte of *z*-axis acceleration.

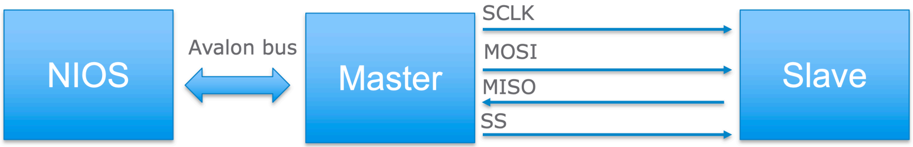
Serial Peripheral Interface (SPI)

- Synchronous Serial Communication
- Short distances
- Embedded systems
- Duplex communication and a Master-Slave architecture

Serial Peripheral Interface (SPI) SPI has four logic signals (which go by alternative namings)

- SCLK : Serial Clock (clock signal from main)
- MOSI : Main Out Sub In (data output from main)
- MISO : Main In Sub Out (data output from sub)
- CS : Chip Select

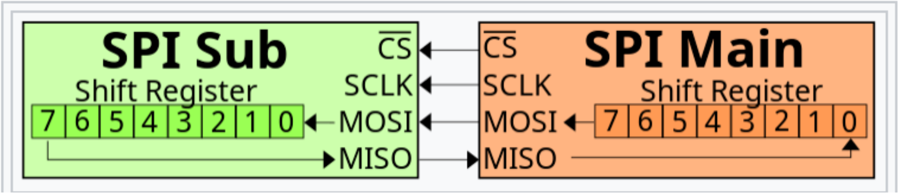
Serial Peripheral Interface (SPI) (ii)



SPI communication

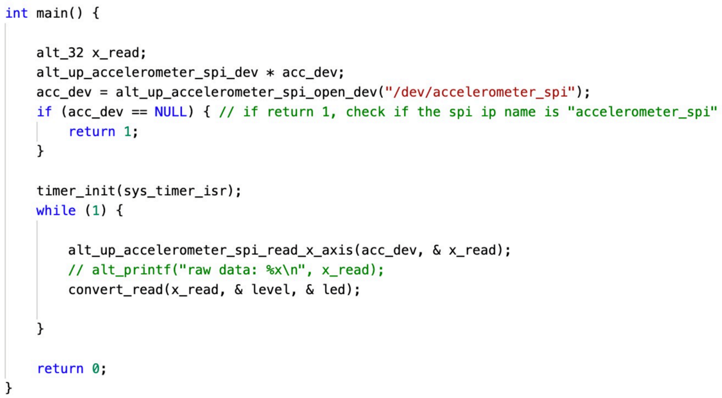
- Chip Select (CS) first, it is possible to have multiple slaves.
- Slave does not have the clock, you must provide one.
- Duplex communication, you are transferring and receiving at the same time.

Serial Peripheral Interface (SPI) (iii)



Program NIOS to read accelerometer values

- Understand code to interface with accelerometer
- Drive the LEDs with the accelerometer value



```c
int main() {

    alt_32 x_read;
    alt_up_accelerometer_spi_dev * acc_dev;
    acc_dev = alt_up_accelerometer_spi_open_dev("/dev/accelerometer_spi");
    if (acc_dev == NULL) { // if return 1, check if the spi ip name is "accelerometer_spi"
        return 1;
    }

    timer_init(sys_timer_isr);
    while (1) {

        alt_up_accelerometer_spi_read_x_axis(acc_dev, & x_read);
        // alt_printf("raw data: %x\n", x_read);
        convert_read(x_read, & level, & led);

    }

    return 0;
}
```

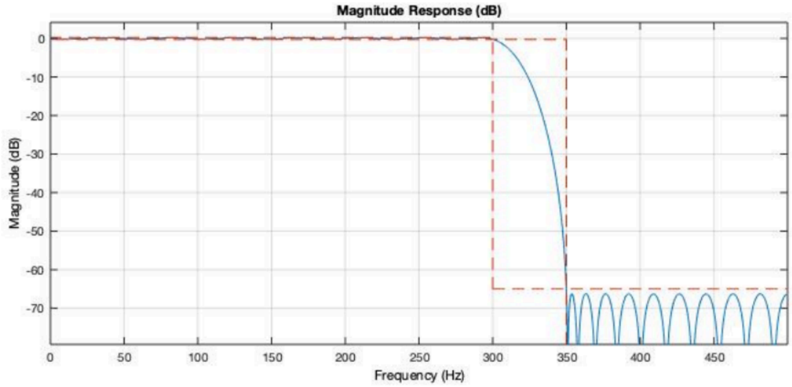
Program NIOS to read accelerometer values (ii)

- There is a main while LookUp function that reads the accelerometer values and drives the LEDs.
- The convert_read() function converts the x_value to led and level pair in order to be used to drive the lighting up of the LEDs.

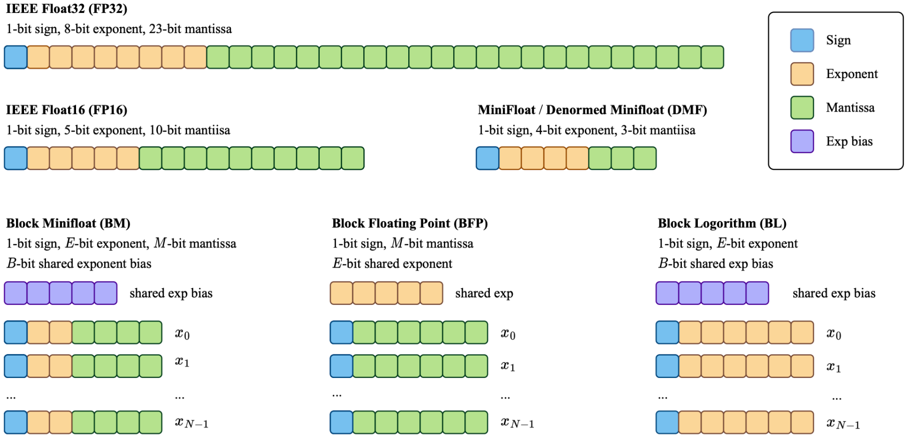
FIR filter implementation and optimisation

- The plain implementation may have glitching, this is because we sample the data at a specific rate and the data may not be smooth.
- In signal processing, we have learned that we can use a low-pass filter to smooth this.
- Moving average with a 5-tap filter
- Extend this to a low-pass N-tap filter
  - Use Matlab to design your filter
- Optimise you program
  - Convert floats to fixed-point values
  - Observe the impact on performance and results
  - Implement these in C, so you can run them on the NIOS processor

FIR filter implementation and optimisation (ii)



Computation in different arithmetic formats



IEEE Float32 (FP32)
1-bit sign, 8-bit exponent, 23-bit mantissa

IEEE Float16 (FP16)
1-bit sign, 5-bit exponent, 10-bit mantiisa

MiniFloat / Denormed Minifloat (DMF)
1-bit sign, 4-bit exponent, 3-bit mantiisa

Block Minifloat (BM)
1-bit sign, E-bit exponent, M-bit mantissa
B-bit shared exponent bias

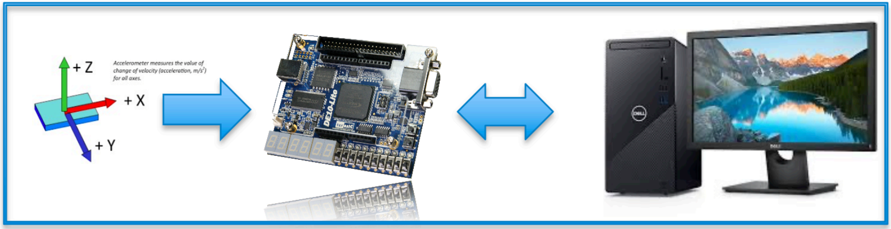
Block Floating Point (BFP)
1-bit sign, M-bit mantissa
E-bit shared exponent

Block Logorithm (BL)
1-bit sign, E-bit exponent
B-bit shared exp bias

An Introduction to Lab4

What is in this lab?

- Understand how to establish a communication process of a NIOSII system with a host PC.
- Establish a number of functions/commands that would allow you to communicate between the board and the host PC.
- Extra: Learn how to add off-chip memory to your system


+ Z
Accelerometer measures the value of change of velocity (acceleration, m/s²) for all axes.
+ X
+ Y
DE10-Lite

What is off-chip memory?

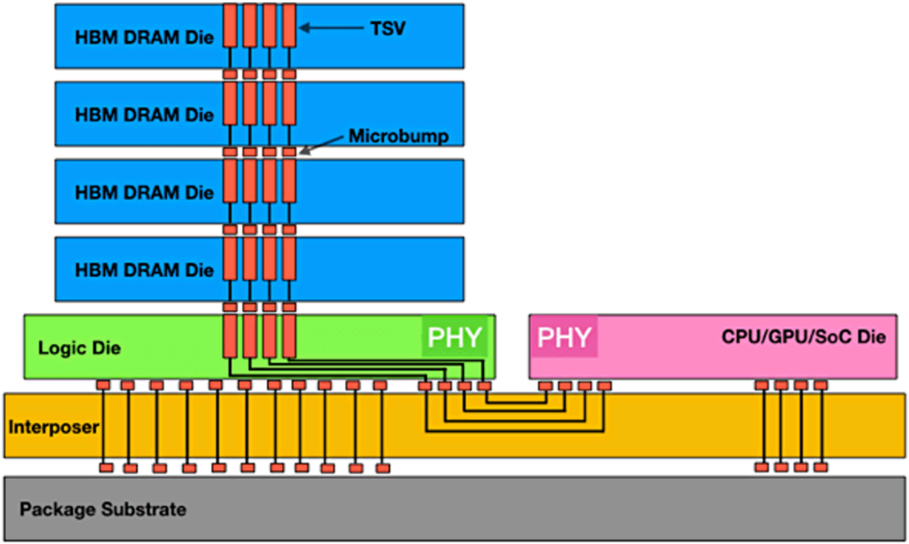
A computer at it's most basic contains a CPU and RAM connected by a bus plus some I/O.

Off-chip memory refers to any kind of memory (usually the main system RAM) that is not directly on the same die as the main CPU.

However, we have now seen different types of memory:

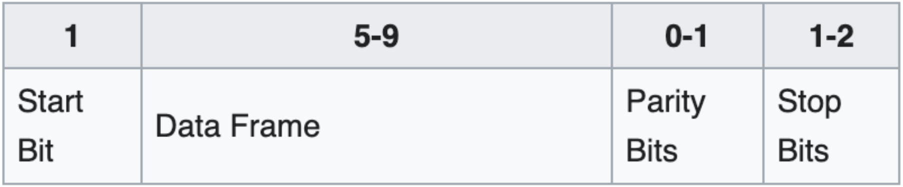
- DDR Memory: the most common one
- GDDR Memory: Graphics DDR Memory, similar to DDR but optimized for graphics cards (GPUs)
- HBM Memory: Stands for High Bandwidth Memory, and is designed for GPUs and other high-performance applications. HBM is more expensive than GDDR but offers higher bandwidth and is more power efficient.

What is off-chip memory? (ii)



What is UART communication

- UART (universal asynchronous receiver-transmitter)
- Device to Device communication
- Asynchronous Serial Communication – (2 wires)
- Agreed frequency of reading – Baud rate
- PC is the master



1	5-9	0-1	1-2
Start Bit	Data Frame	Parity Bits	Stop Bits

Lab structure

- Part 1: Give you an example to understand the communication
- Part 2: Integrate UART with Lab3
- Part 3: Add command to update the coefficient. Conversion of characters to numbers
- Part 4: Plot received accelerometer data at real time

Final Group Project

What is in the final group project?

- Use what you have learned in the labs to design a system that has information flowing from sensors to cloud server and then back to sensor or a local processing node.
- Minimum functional requirements:
  - Local processing of the accelerometer data
  - Establishing a cloud server to process events/information
  - Communicating information from the server back to the nodes in way that the local processing can be impacted.
  - Use of two nodes

What I hope you have learned?

In an abstract way, I would say 99% of the computation is about:

1. The movement of the data
2. The processing of the data

So hopefully in this course you have learned how systems can be designed to perform the above two aspects.