

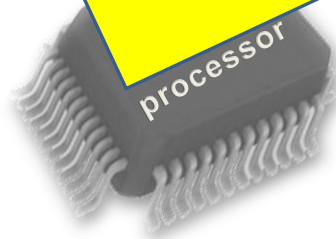


EEE4120F



OPTIONAL SLIDES

Lecture 29 Softcore Processors



MicroBlaze



PicoBlaze

The Xilinx PicoBlaze Microcontroller

PicoBlaze

Xilinx

DUGONG



(Open source)

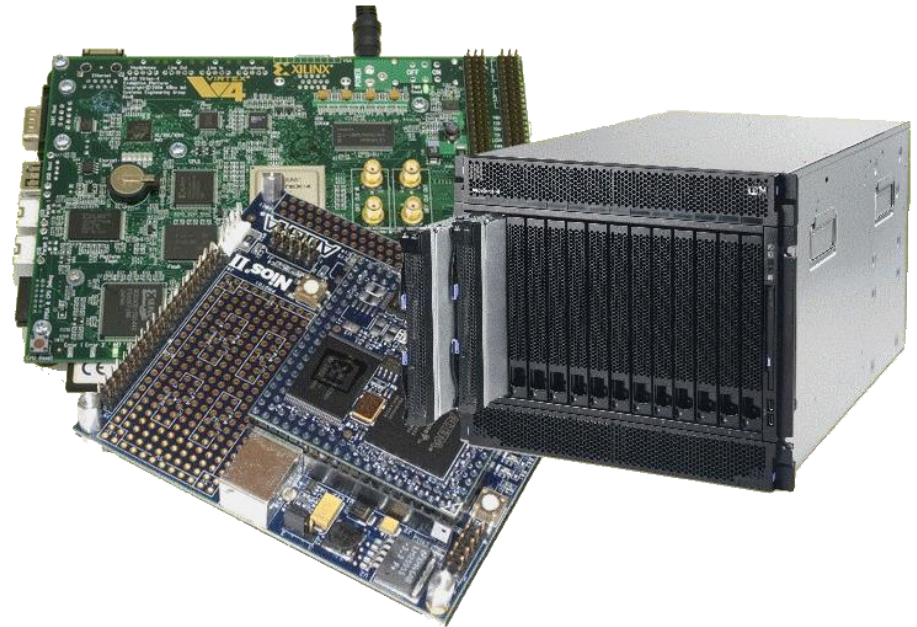
Lecturer:
Simon Winberg

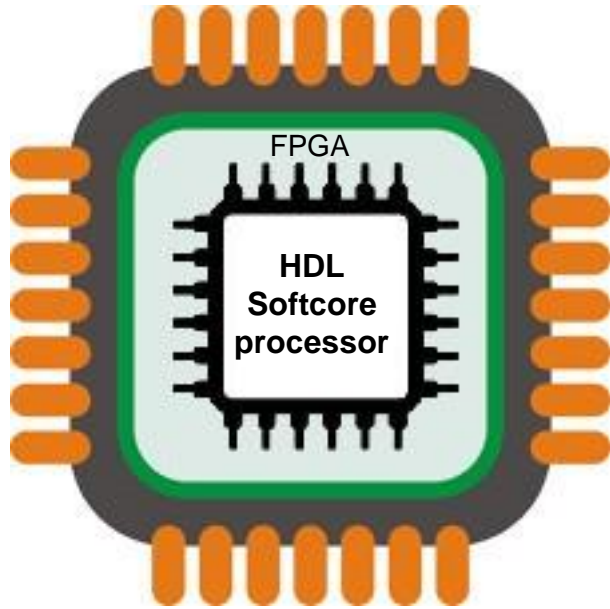


Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

Lecture Overview

- Softcore processors
- Case studies:
 - Xilinx Microblaze
 - Xilinx Picoblaze
 - DUGONG
(a configuration control 'sort of' processor)
 - [Optional:
Altera/Intel NIOS2]



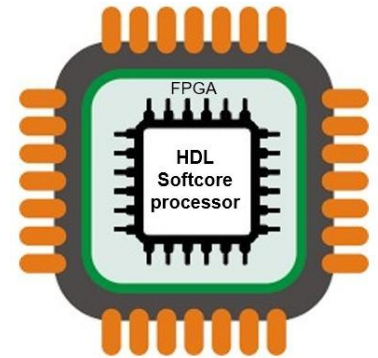


Softcore processors

EEE4120F

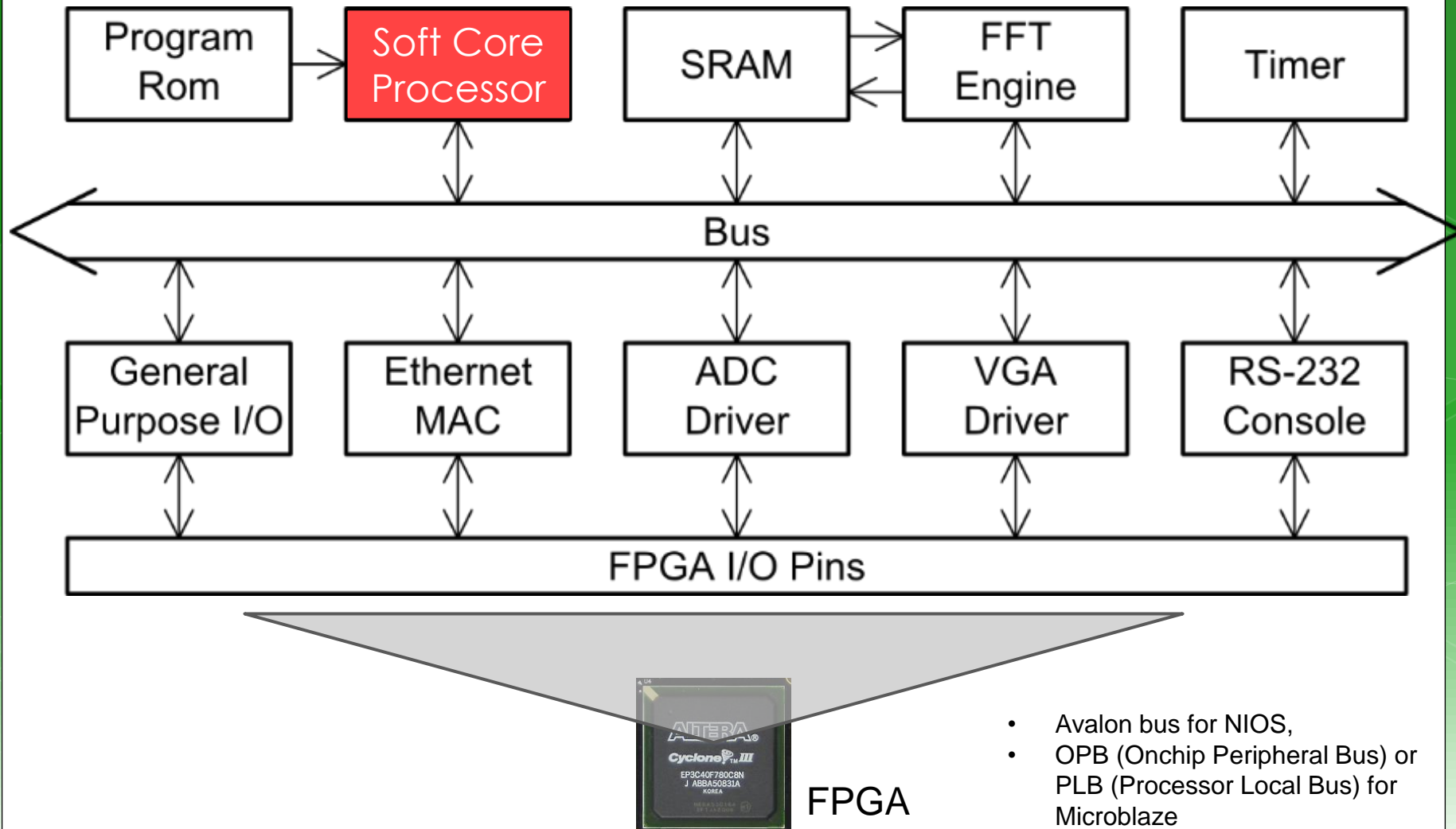
Soft core Processors:

Processors within FPGAs

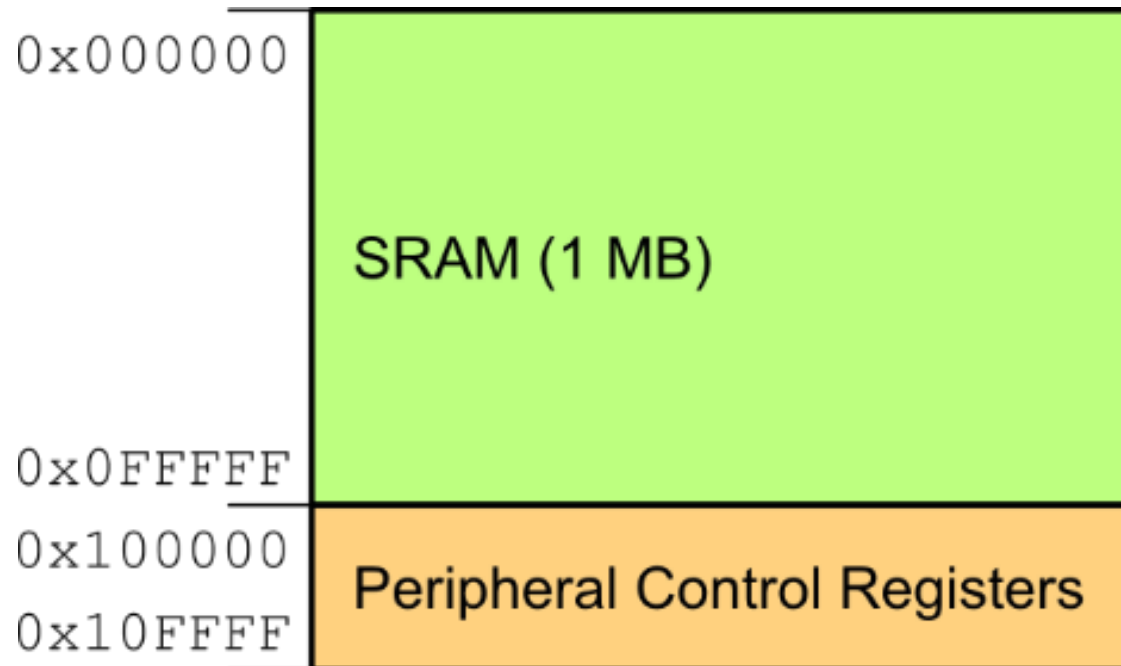


- What is a 'soft core'?
 - A soft microprocessor is a microprocessor core that can be wholly implemented using logic synthesis.
 - It is essentially an item of 'Intellectual Property' (IP) that contains the **design of a processor**.
 - A soft core processors can be incorporated into a design and then the design synthesized, place & routed to generate an executable.
 - Examples of these:
 - Altera NIOS II
 - Xilinx Microblaze & Xilinx Picoblaze
 - A variety of ARM processors
- How do these compare to hard cores? ... *see later!*

How a Soft core typically fits in to a design



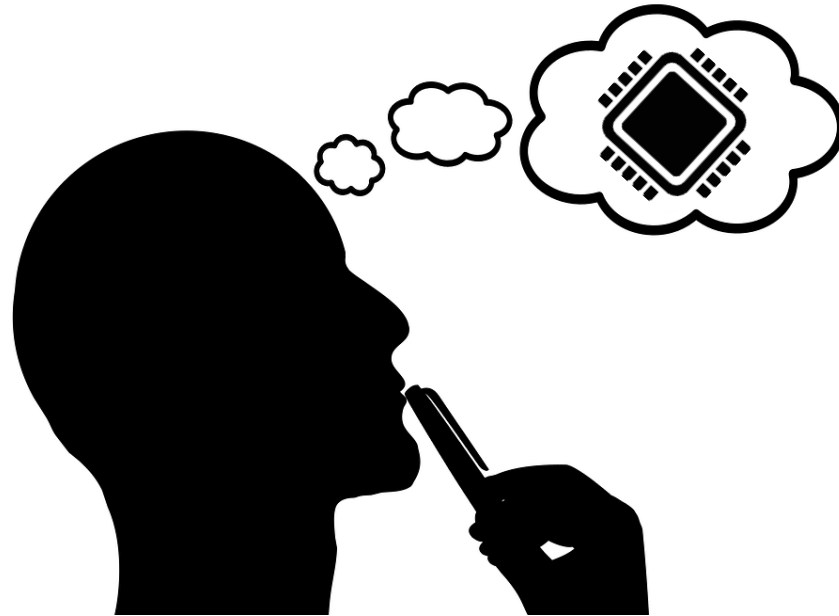
Address Bus for the Processor



- FFT Engine
- Timer
- GPIO
- Eth MAC
- VGA
- ... etc ...

Thinking Time

We might try
Zoom rooms
for this



What are some of the reasons for using a softcore processor?

*Work with a classmate and make
a list of reasons and then prioritize these*

Reasons for using a softcore processor

- Design Flexibility
 - 'best of both worlds' – allows you to choose either a software or hardware solution for implementing particular functionality
- Implementing complex (not necessarily very fast) functionality
 - Software is useful for implementing complex control logic or functions that the system may need, e.g. to implement the TCP/IP protocol stack (doing this in combinational logic is hard and takes up much logic).
- Reuse!!
 - Your underlying software is likely to outlive the specific platform it was initially designed for
- Saving logic
 - Some implementations are just more efficient to do using a processor (multiple reusable instruction approach) rather than implementing a circuit for this.
- Multiple core support
 - Choose how many cores you need, from 0 to however many you can fit on the FPGA.
- Avoiding obsolescence
 - Your software is likely to last a lot longer compared to a CL design that may be more tied to the platform.



Soft Cores Ingredients

Which are likely influence the design of FPGAs now and in the future.

Ingredients Needed for Soft Cores

- Could implement it all as gates, but that can be rather inefficient and wasteful
- On-chip resources that soft core processors typically use:
 - RAM blocks - *various configurations*
 - Full adders
 - DSP cores (e.g. integer multipliers, etc.)
 - State machines



Hard core vs. Soft core ?!

**Hardcore =
Difficult??**

**Softcore =
Lite/Easy??**



Not necessarily...

Hard Core Processor Characteristics

- Not implemented as gateware programmed into the FPGA logic
- Implemented in silicon/ASIC; unchangeable
- Benefits:
 - Usually much higher performance
 - Manufacturer typically provides optimized compiler for their chips

Reading (supplementary, not examined)

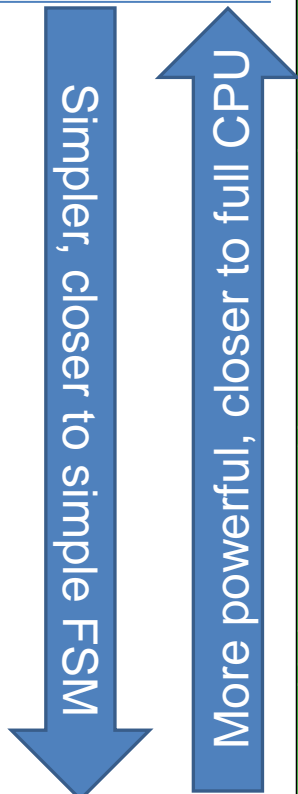
Paper L29 - 07157822 - Dual-Issue Embedded Processor for Low Power Devices.pdf

“A Dual-Issue Embedded Processor for Low Power Devices” by H Lozano, M Ito

The authors propose “an energy efficient dual-issue embedded processor that can deliver up to 60% improvement in IPC (instruction-per-cycle)...”

<http://dx.doi.org/10.4108/icst.iniscom.2015.258414>

Case Studies (shortly)

- MicroBlaze
 - 32-bit processor, RISC instructions.
 - Thirty-two 32-bit general purpose registers
 - Generate, configure and program with Xilinx SDK
 - Picoblaze
 - 8 bit, simple RISC instructions
 - Free for use on Xilinx
 - DUGONG (a scenario showing example developed by a UCT student, a state machine with instructions)
 - Planned to be a configuration control module
 - Open-source
- 
- NIOS II NIOS is recommended additional reading (it is an Altera/Intel product)
 - Altera's soft core processor, 32 bit, configurable
 - Commercial
 - NIOS II Gen2 improved, more options
 - Alternate soft cores: MP32 / MIPS32, ARM

Considerations for Using Soft Core Processors

- A major purpose is:
 - Make it easier to communicate / transfer & packetize data
 - Test designs (e.g. like reading/adjusting setting while running)
 - Set up the initial conditions of your design / HDL block configuration
 - Highly configurable processor structure – e.g. add a custom instruction to baseline processor
 - Of course also e.g. testing novel processor designs
- Should consider the soft core processor is
 - More an aid to control the system
 - facilitate comms, advantages of software running in system, e.g. O/S
- The soft core processor shouldn't
 - Be provide the main solution to you digital accelerator, unless there is good reason such as experimenting with performance of a cluster of processors on FPGA

First two points are most common use



Optional Case Study

- NIOS II
 - Altera's soft core processor, 32 bit, configurable
 - Commercial
 - NIOS II Gen2 improved, more options
 - Alternate soft cores: MP32 / MIPS32, ARM

NIOS is recommended additional reading, see last slides
It is an Altera/Intel product

Soft Core Case Studies

MicroBlaze



PicoBlaze DUGONG

The Xilinx PicoBlaze Microcontroller

PicoBlaze™

Xilinx



(Open source)

Xilinx Microblaze

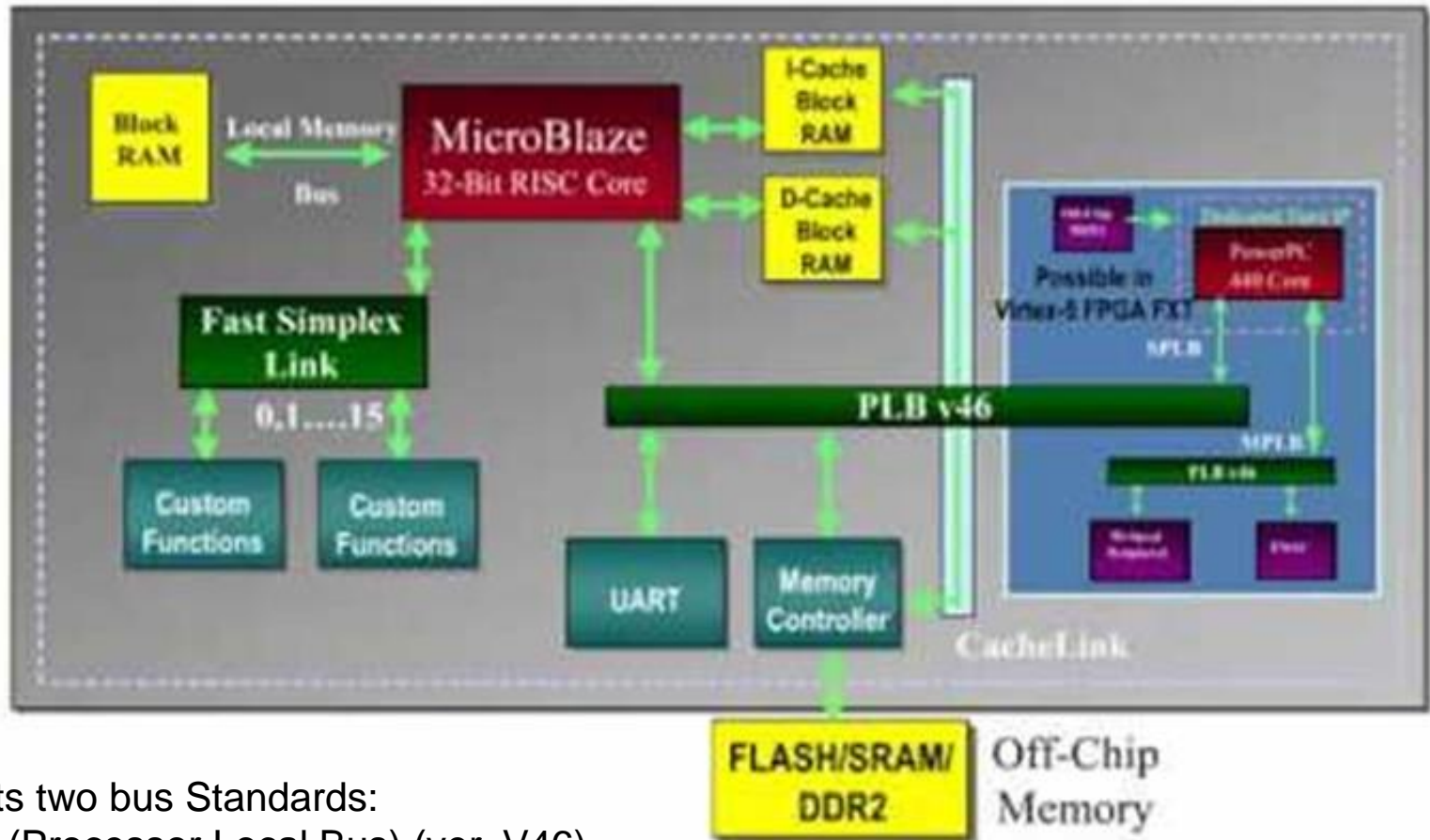


- MicroBlaze was developed by Xilinx
- It is a soft microprocessor core designed and optimized for use on Xilinx FPGAs
- MicroBlaze is a Reduced Instruction Set Computer (RISC)
- Generated, configured and programmed using Xilinx EDK (Embedded Development Kit)

FPGAs that support Microblaze

- You can't run it on any FPGA!! It works on only supported those Xilinx FPGAs. These include the FPGAs families:
 - Spartan-3/3A/3AN/3A DSP/3E FPGA
 - Spartan-6
 - Virtex-4 FX (also supports PowerPC 405 processors) and LX/SX FPGA
 - Virtex-5 FXT (alost PowerPC 440 processor) LX/LXT FPGA (MicroBlaze)
 - Virtex-6 (MicroBlaze processor)

Microblaze and its surrounding components



Supports two bus Standards:

- PLB (Processor Local Bus) (ver. V46)
for high speed & high bandwidth peripherals.
Supports 128 bits of data, 36 bit address
- XILINX Local Memory Bus (LMB) to interface local PLBs

MicroBlaze memory

- Cache
 - By default MicroBlaze has no cache.
 - Can enable cache in EDK (which will use BRAM)

MicroBlaze Architecture

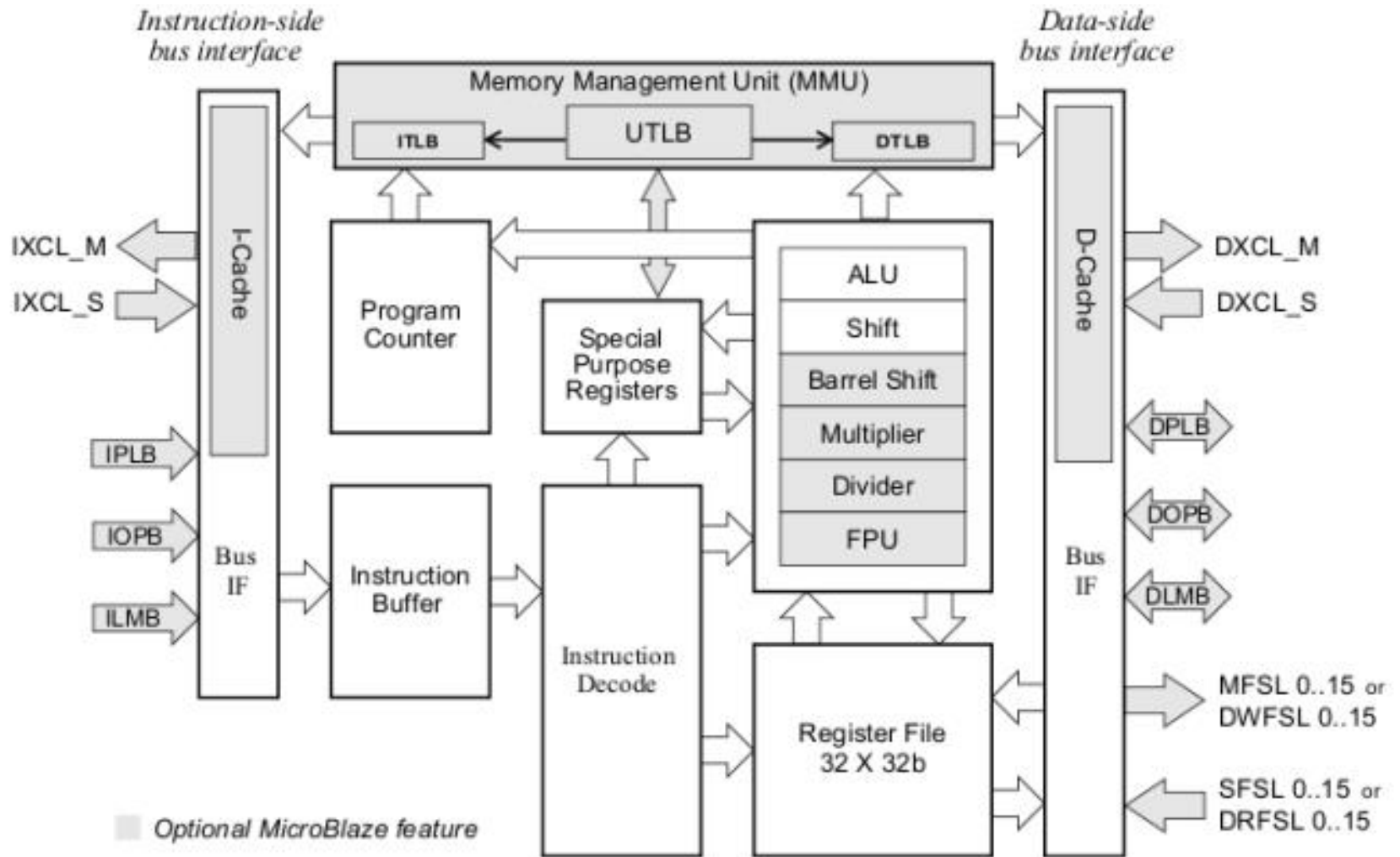


Figure 1-1: MicroBlaze Core Block Diagram

Features

- Highly configurable soft processor
- Can select a specific set of features needed
- Fixed features of the processor are:
 - Thirty-two 32-bit general purpose registers
 - 32-bit instruction word with three operands and two addressing modes
 - 32-bit address bus
 - Single issue pipeline
- The MicroBlaze can be parameterized to enable inclusion of additional functionality.

MicroBlaze instructions

- The MicroBlaze has a versatile interconnect system to support a variety of embedded applications.
- MicroBlaze instruction set is similar to that of the RISC-based DLX architecture.
- Most vendor-supplied and third-party IP modules interface to PLB* directly (or through an PLB to OPB** bus bridge.)
- For access to local-memory (BRAM in the FPGA), the MicroBlaze uses a dedicated LMB bus that reduces loading on the other buses (e.g. OPB).
- User-defined coprocessors are supported through a dedicated FIFO-style connection called FSL (Fast Simplex Link).
- The coprocessor(s) interface can accelerate computationally intensive algorithms by offloading parts or the entirety of the computation to a user-designed hardware module.

* PLB = *Processor Local Buss* (i.e. not programmable logic block in this case)

** OPB = On-chip Peripheral Bus

MicroBlaze

- High performance soft processor
 - Most of instructions completed with 1 cycles
 - Short pipelines, higher working frequency
 - FPGA optimized implementation
 - Fast carry chain MUX, hardware multiplier
 - RLOC placement constraints
- Easy (relatively) to use with Xilinx EDK
 - Linux/GCC support

MicroBlaze Limitations

- No MMU support
 - No protection among processes
 - Implies it cannot support full version of Linux
- No double precision floating point
- No atomic instructions
 - hard to implement lock
 - non-blocking FIFO instruction problem
- No cache coherent support

Micro MicroBlaze

- **A smaller, but powerful MicroBlaze:
The MicroBlaze MicroController
System (MCS)**
- MCS offers an even smaller package
- MCS consists of MicroBlaze in a fixed 3-stage pipeline configuration for the smallest footprint, and is surrounded by a system of common embedded design peripherals*.

* MCS can run from IDS Logic Edition without the need for a full embedded design license

MicroBlaze – references & where to find out more

- Xilinx MicroBlaze Wiki
 - <http://www.wiki.xilinx.com/MicroBlaze>
- Wikipedia
 - <https://en.wikipedia.org/wiki/MicroBlaze>
- Building an embedded system using a Microblaze on an Nexys3
 - http://islab.soe.uoguelph.ca/sareibi/TEACHING_dr/XILINX_TUTORIALS_dr/EDK_dr/EDK-NEXYS3-Elhossini2012.pdf
- Some of the information regarding MicroBlaze based on slides by K. Siddhapathak

Soft Core Case Studies

MicroBlaze



PicoBlaze

The Xilinx PicoBlaze Microcontroller

PicoBlaze™

Xilinx

DUGONG



(Open source)

PicoBlaze Soft Core Processor

Info available from: www.xilinx.com/picoblaze

- Size:
 - Takes on 96 slices !! (slice = group of PLBs)
 - Big advantage considering low cost FPGAs have a few 100 slices; and 1000s slices for medium range
- Not coded as behavioral VHDL
 - It is manually pre-compiled (IP block)
 - Built by instantiation of Xilinx raw primitives
 - Can still be simulated using Modelsim

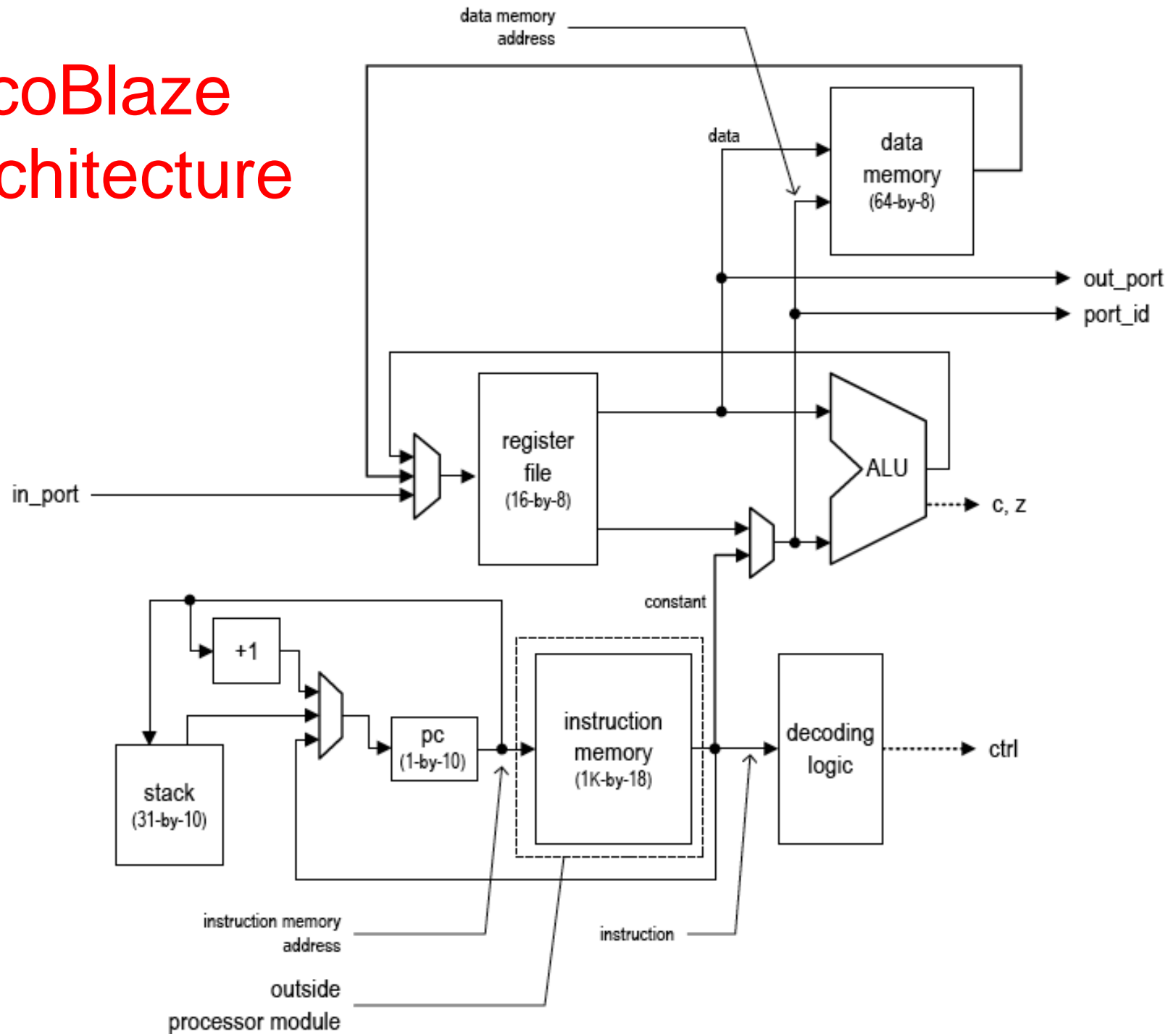
PicoBlaze Soft Core Processor

- Main Uses
 - Control / configuration of parameters
 - Some data processing (add / integer operations)
 - Comms / UI facilities (i.e. 'talk' to control PC, especially UART, USB connection)
 - Testing & debugging your design on hardware

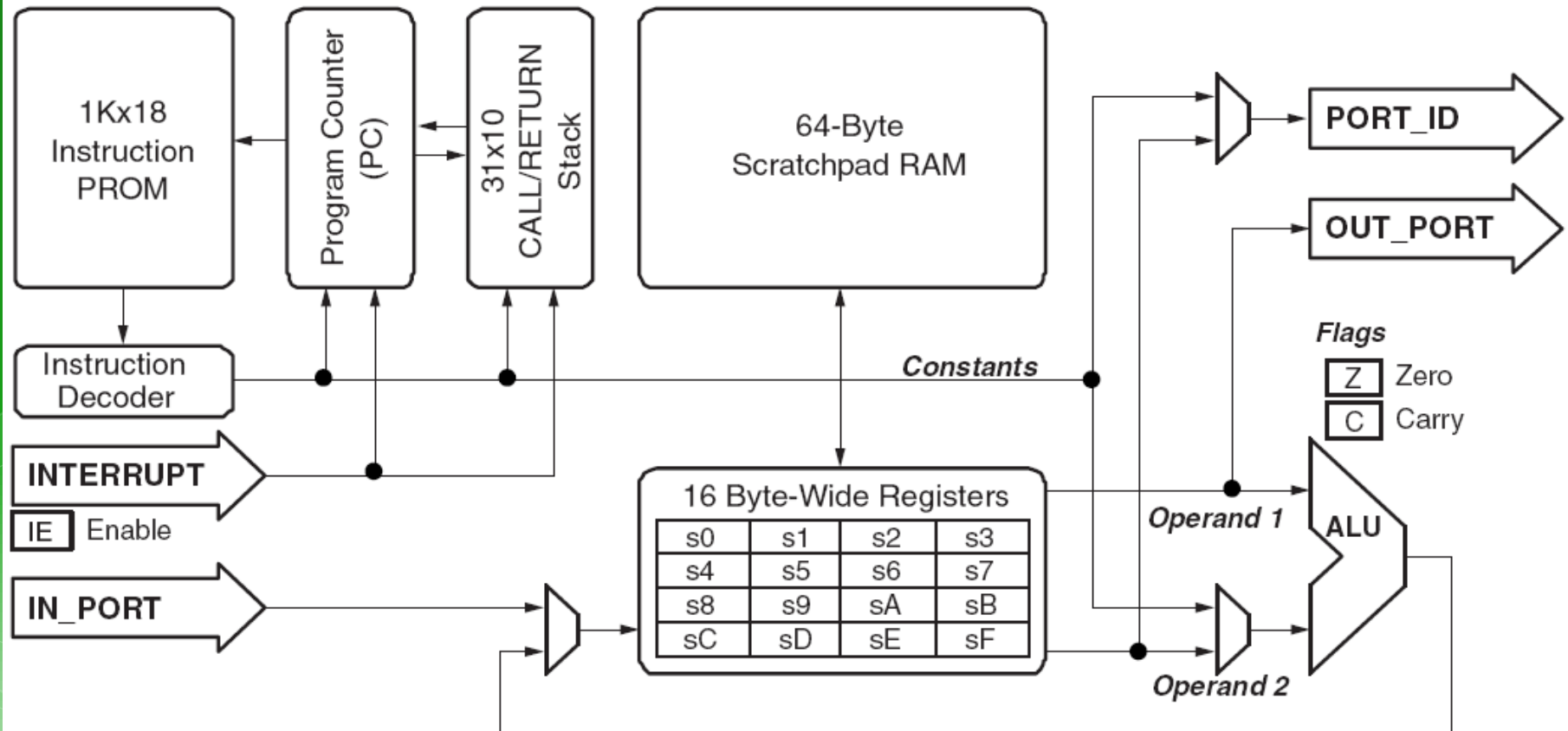
PicoBlaze Soft Core Processor

- Can be very helpful to speed-up & simplify development
 - Changing the instructions running on the softcore PicoBlaze is likely faster than changing the HDL code (especially with rebuilding time)
 - Has a simple and easy to learn instruction set, also offers opportunities for reuse of the PicoBlaze code in other systems.

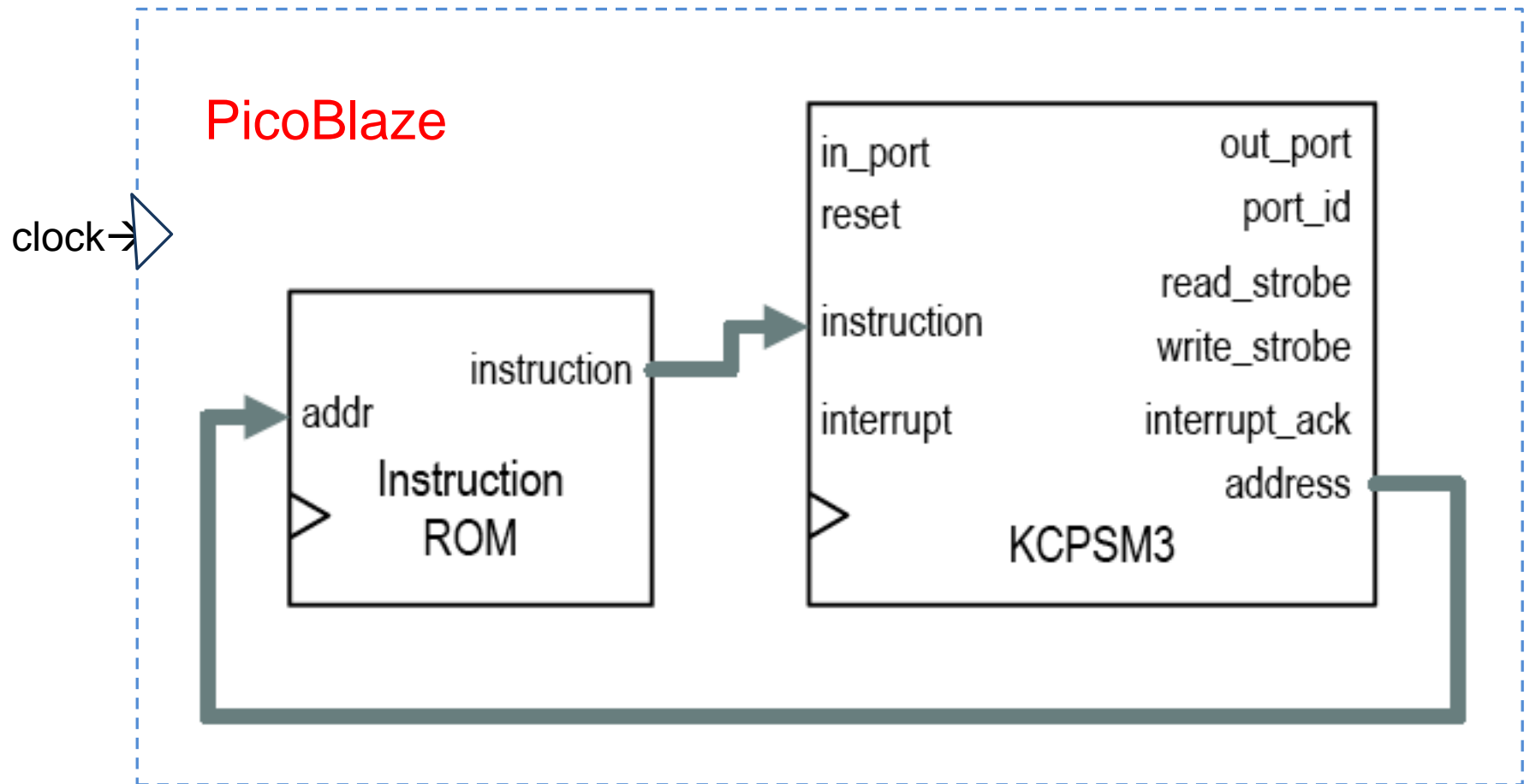
PicoBlaze Architecture



Overview of the PicoBlaze



Interfacing to the PicoBlaze



KCPSM = constant (K) Coded Programmable State Machine

PicoBlaze Interface

Name	Direction	Size	Function
clk	input	1	System clock signal in
reset	input	1	Reset signal
address	output	10	Address of instruction mem. Specifies address of the instruction to be retrieved.
Instruction	input	18	Fetches instruction.
port_id	output	8	Address of the input or output port.
in_port	input	8	Input data from I/O peripherals.
read_strobe	output	1	Strobe associated with the input operation.
out_port	output	8	Output data to I/O peripherals.
write_strobe	output	1	Strobe associated with the output operation.
interrupt	input	1	Interrupt request from I/O peripherals.
interrupt_ack	output	1	Interrupt acknowledgment to I/O peripherals

PicoBlaze Addressing Modes

Instruction example

Explanation

Immediate mode

ADDCY s2, 08

SUB s7, 7

$s2 + 08 + C \rightarrow s2$

$s7 - 7 \rightarrow s7$

Direct mode

INPUT s5, 2a

ADD sa, sf

$\text{PORT}[2a] \rightarrow s5$

$sa + sf \rightarrow sa$

Indirect mode

INPUT s9, (s2)

STORE s3, (sa)

$\text{PORT}[\text{RAM}[s2]] \rightarrow s9$

$s3 \rightarrow \text{RAM}[\text{RAM}[sa]]$

Example Program

```
; DEMONSTRATION PROGRAM
; EXERCISE THE 7 SEGMENT DISPLAYS
cold_start:
    LOAD s3, 11 ; clear all time values
    OUTPUT s3, 04
    LOAD s4, 00
    OUTPUT s4, 05
main_loop:
    OUTPUT s5,04 ; Update 4 digit 7 seg
display
    OUTPUT s3,06
    JUMP main_loop
```

Soft Core Case Studies

Nios II PicoBlaze DUGONG



Nios® II

The Xilinx PicoBlaze Microcontroller

PicoBlaze™

Xilinx



(Open source)

Why look at this one

This provides a case study of a custom-designed 'sort of' processor which provides a variety of useful facilities that are useful especially in configuring other modules in an FPGA, or being used as a junction for moving data around within the FPGA or steering the data to e.g. an output port or for doing debug inspections (kind of like a seriously cut-down boot monitor for an FPGA; you can send it various commands – in machine language – and it can carry them out to do inspections on the modules running in the system)

DUGONG

- Developed by Matthew Bridges (SDRG student), part of a larger research project
- Designed from scratch (a solution to a need)
- Design around compatibility with Wishbone interface
- Designed with a focus on simplicity with scalability

But what is it???

Available at:

<https://github.com/matthewbridges/dugong>

‘DUGONG’ name?

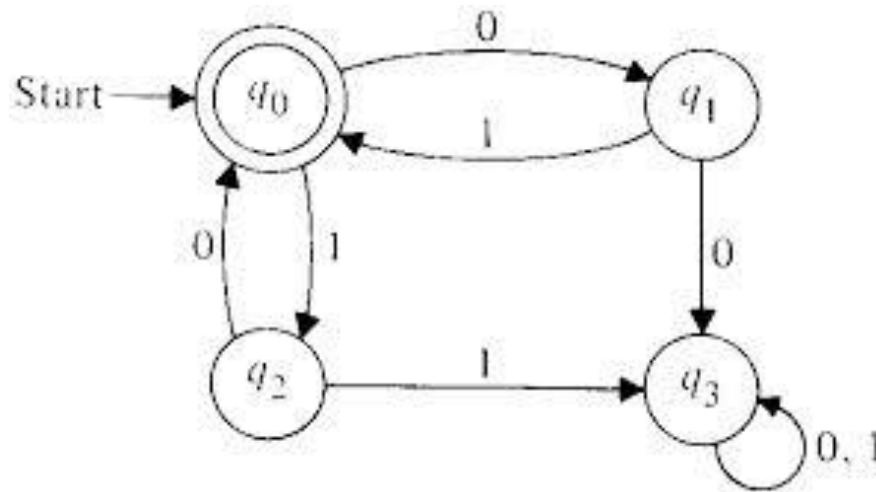


A dugong is actually a type of underwater mammal, colloquially called ‘sea cow’ that is found in the Indian Ocean (similarities to ‘manatees’)

The DUGONG controller was given this name as it’s a ‘underwater’ hidden feature, and no one else is likely to choose the name... and it is a kind of grazer like a RHINO

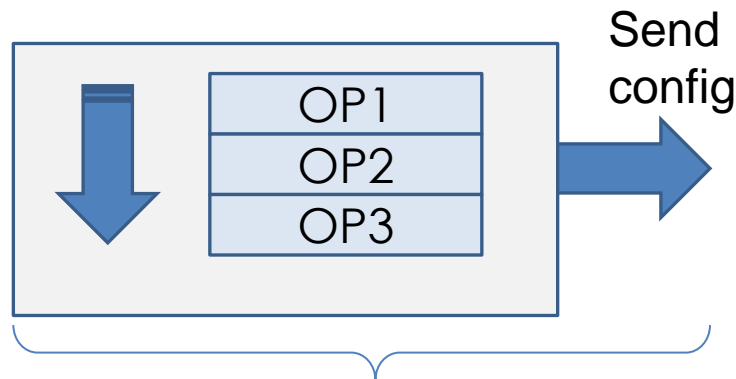
DUGONG

- Essentially it is a glorified state machine
- But has limited number of states and transition types (i.e. predefined list of triggers, operations, transitions, etc.)

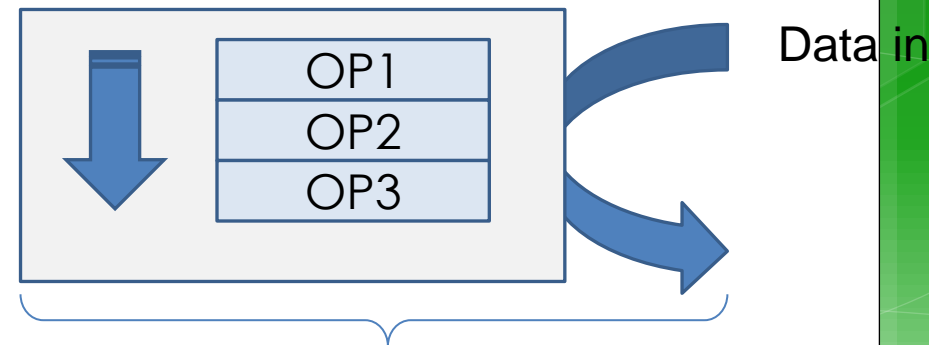


DUGONG

- Essentially it is a glorified state machine
- Works as follows:
 - A list of timing and transfer operations are loaded into it, and it runs through these sequentially



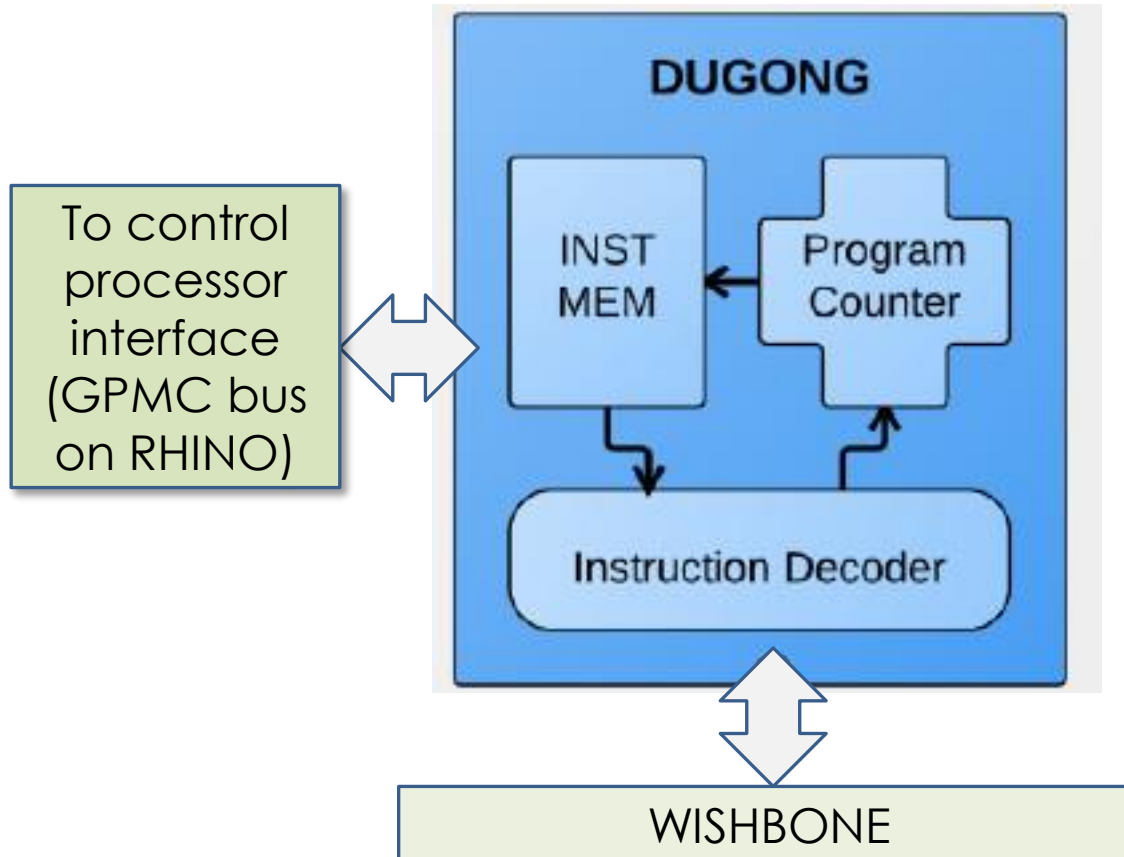
MODES: Config mode



Redirect mode

Various other modes, some simultaneous.

DUGONG Structure

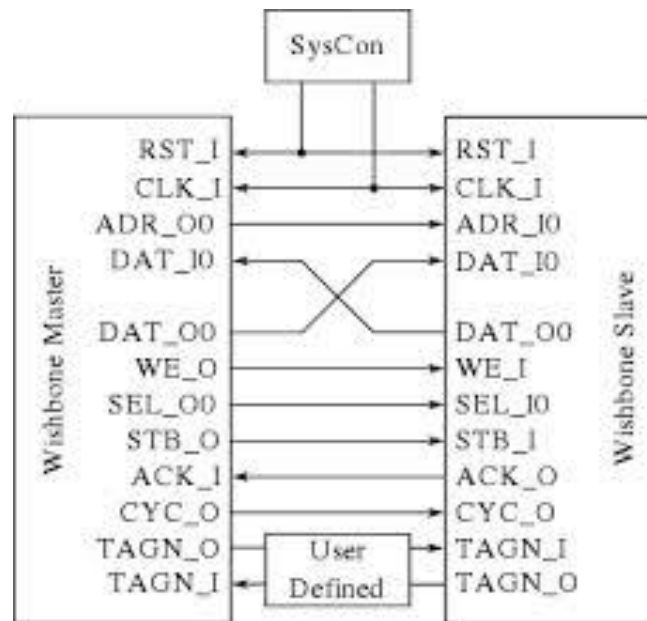


Has one register, the accumulator, through which data is transferred.

Connecting to DUGONG

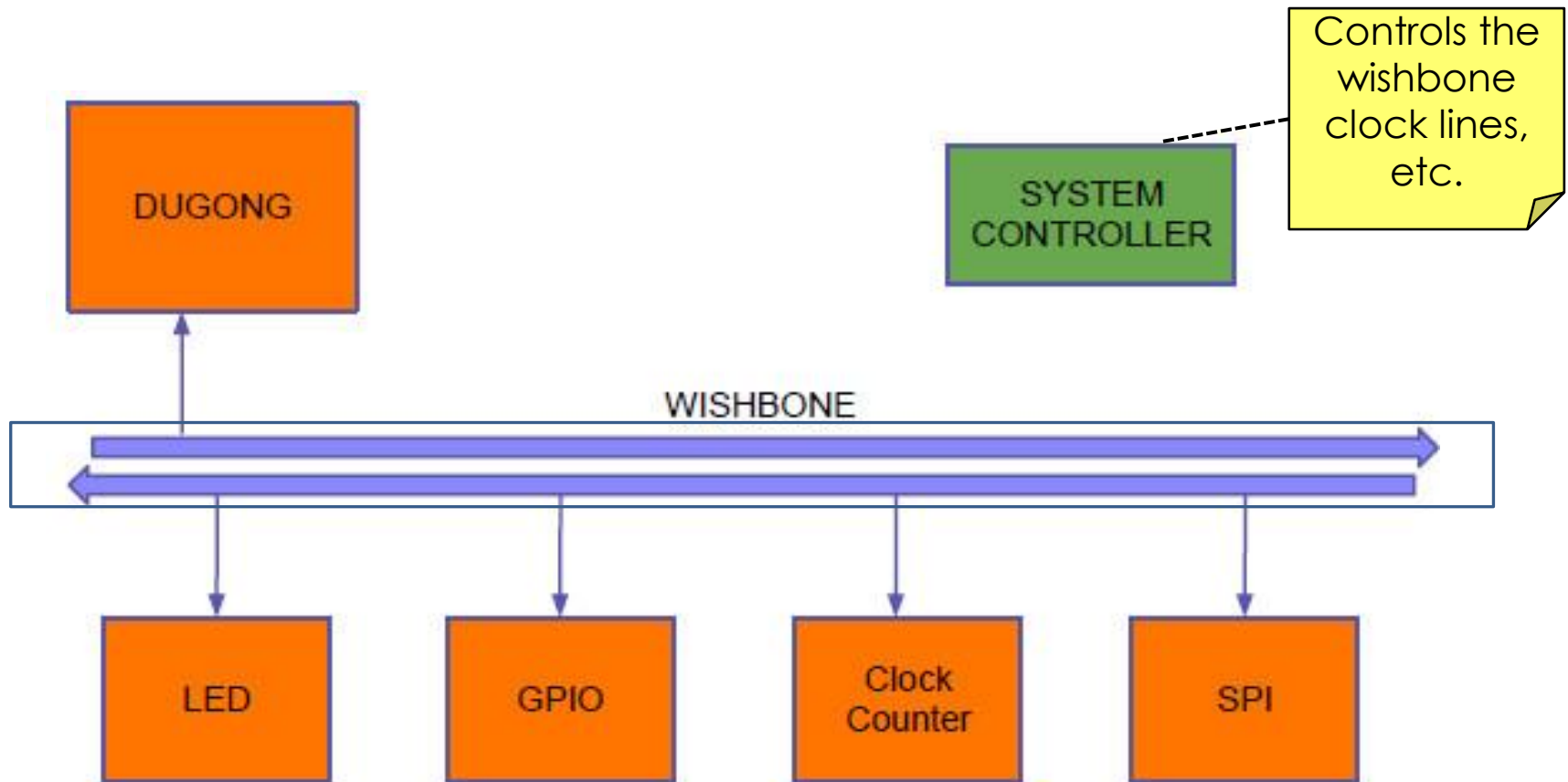
- Serves as the Master on the wishbone bus

Wishbone bus: Open source hardware interconnect bus structure for connecting computational blocks within ICs / reconfigurable computers. This bus structure is commonly used by designs provided by OpenCores.



Wishbone interfaces for
Master and Slave modules

How DUGONG fits in to a Wishbone-based design



DUGONG

- ◉ Designed around simplicity & scalability (connect up and configure small to big designs)
- ◉ Has no function unit (FU) – with good reason
- ◉ Can I use it for processing? Comparisons?
 - ◉ Sorry, no can do: that is someone else's problem (or rather use a CPU e.g. Nios II)

There are lots of other opensource soft cores available, see e.g.:

OpenCores.org: <http://opencores.org/>

Open Hardware Repository: <http://www.ohwr.org/>

DUGONG Instruction Set

- DUGONG1:
 - 4 bit Instruction Set
 - Structure: <CODE:4 bits> <DATA:28 bits>

Code	Instruction	Description
0000	NOP	No operation
0001	WRITE	Write data to addr
0010	READA	Read from addr to acc
0011	WRITEA	Write from acc to addr
0100	BRANCH	write data to pc / controller
1000	WAIT	Wait for a number of clock cycles equal to data field

DUGONG Typical Program

WRITE #0, [0xFF000000] // write 0 to address

READA [0xFF100000] // read from addr into A

WAIT #4 // wait for 4 clock cycles

WRITEA [0xFF200000] // write A to addr

BRANCH // send data in A to controller

NOP

NOP ...

Q & A

Soft Core Case Studies

Nios II



Nios® II

PicoBlaze DUGONG

The Xilinx PicoBlaze Microcontroller

PicoBlaze™

Xilinx



(Open source)

Altera Nios II Processor

- A 32-bit soft core processor from Altera
- Three standard cores versions:
 - Fast, Standard, Light
- The cores have tradeoffs in:
 - FPGA LEs needed \leftrightarrow power \leftrightarrow speed
- RISC architecture: Simple instructions
- Harvard Architecture:
 - Separated data and instruction memories (a likely safer approach to the Von Neumann arch. style)
- 32 interrupt levels
- Avalon Bus interface
- C/C++ compilers; can also use plain assembly

NIOS II Architecture

Let's have a
closer look
at the core

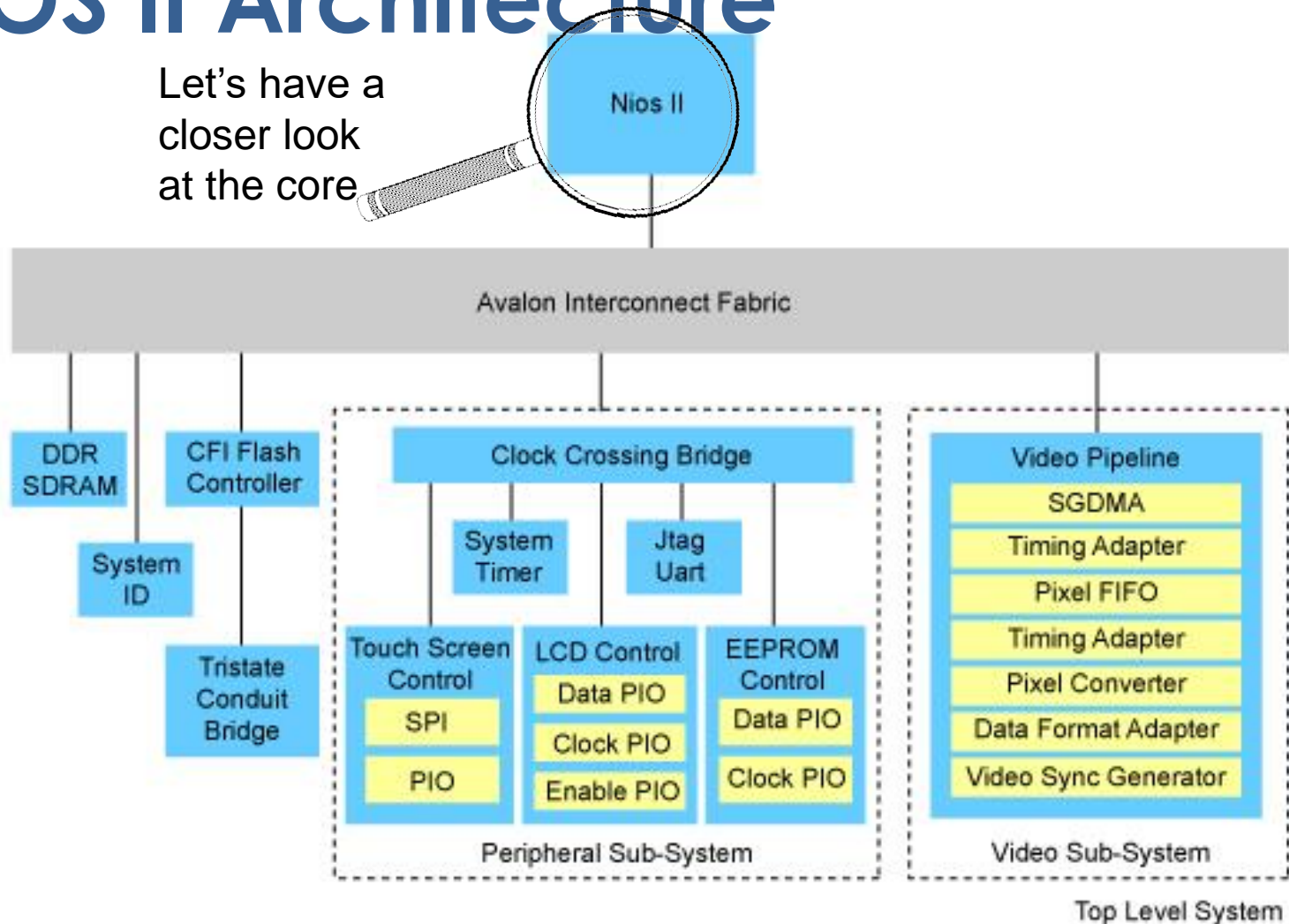
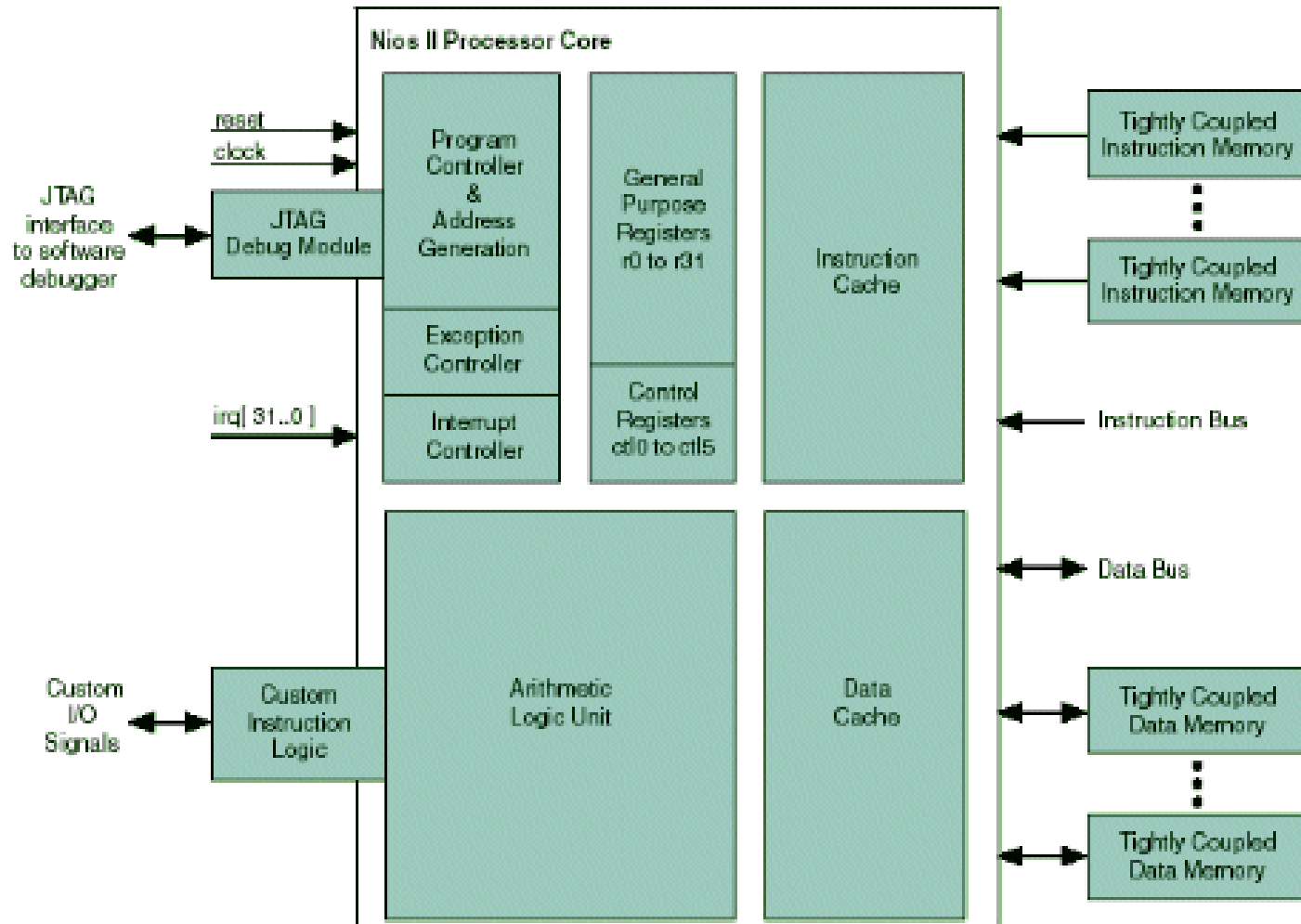


Image source: www.altera.com

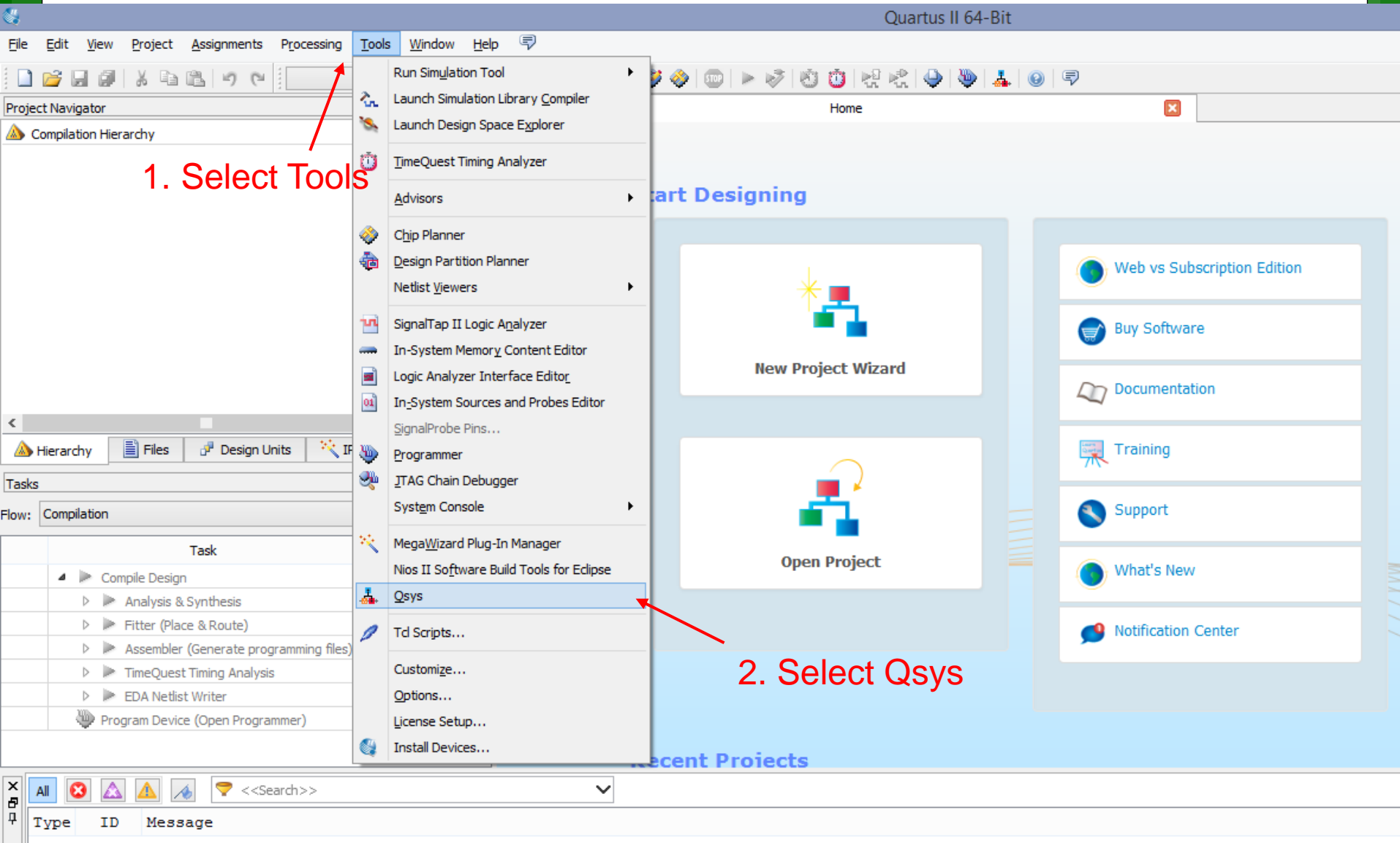
Nios II Core



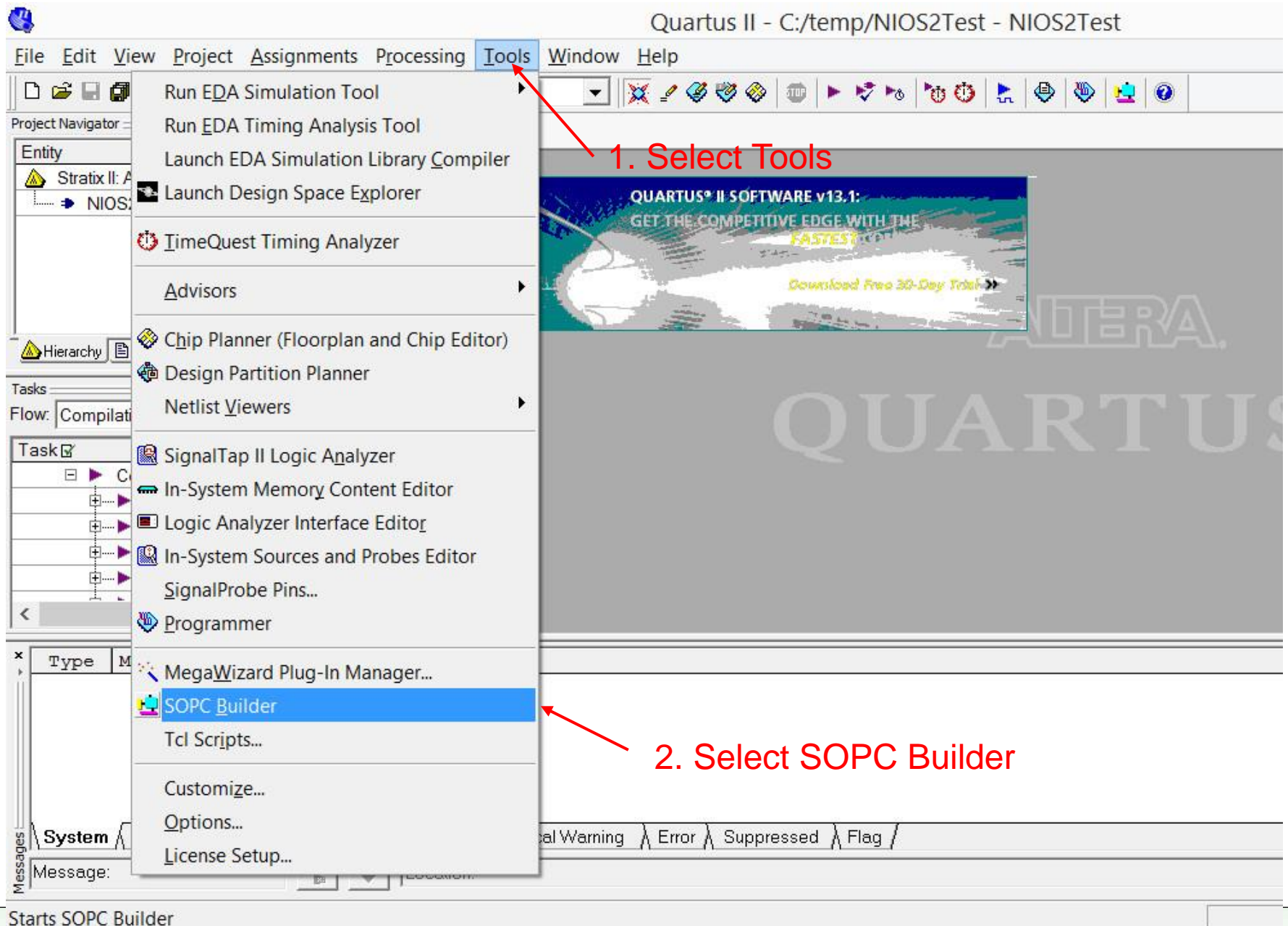


How do you get a NIOS
II soft core processor
and set it up?

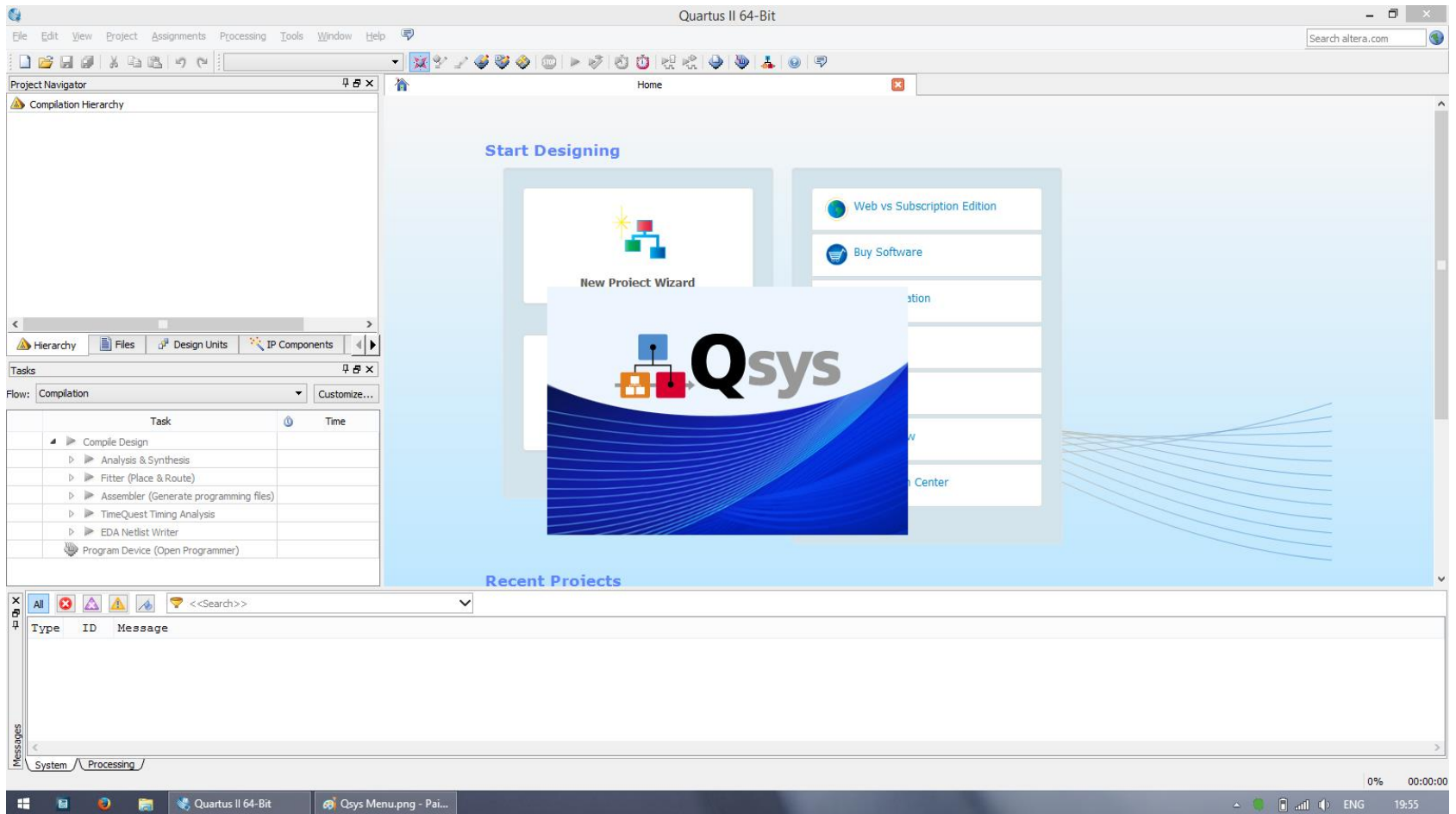
Altera Qsys: In later versions of Quartus II



Altera SOPC Builder: In earlier versions of Quartus II



Altera QSys



Disclaimers and copyright/licensing details

I have tried to follow the correct practices concerning copyright and licensing of material, particularly image sources that have been used in this presentation. I have put much effort into trying to make this material open access so that it can be of benefit to others in their teaching and learning practice. Any mistakes or omissions with regards to these issues I will correct when notified. To the best of my understanding the material in these slides can be shared according to the Creative Commons “Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)” license, and that is why I selected that license to apply to this presentation (it’s not because I particularly want my slides referenced but more to acknowledge the sources and generosity of others who have provided free material such as the images I have used).

Image sources:

man working on laptop – flickr

scroll, video reel, big question mark – Pixabay <http://pixabay.com/> (public domain)

some diagrammatic elements are from Xilinx ISE screenshots

<https://commons.wikimedia.org/wiki/Category:Images> (creative commons)

