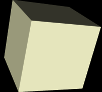
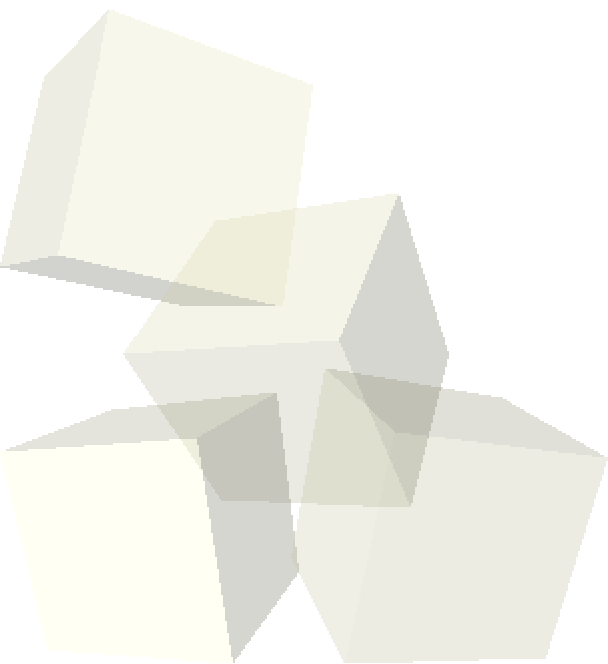
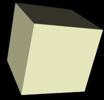


Lab 2: lowRISC Untether

Manuel J. Bellido Díaz

Febrero de 2017

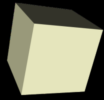




lowRISC Untether

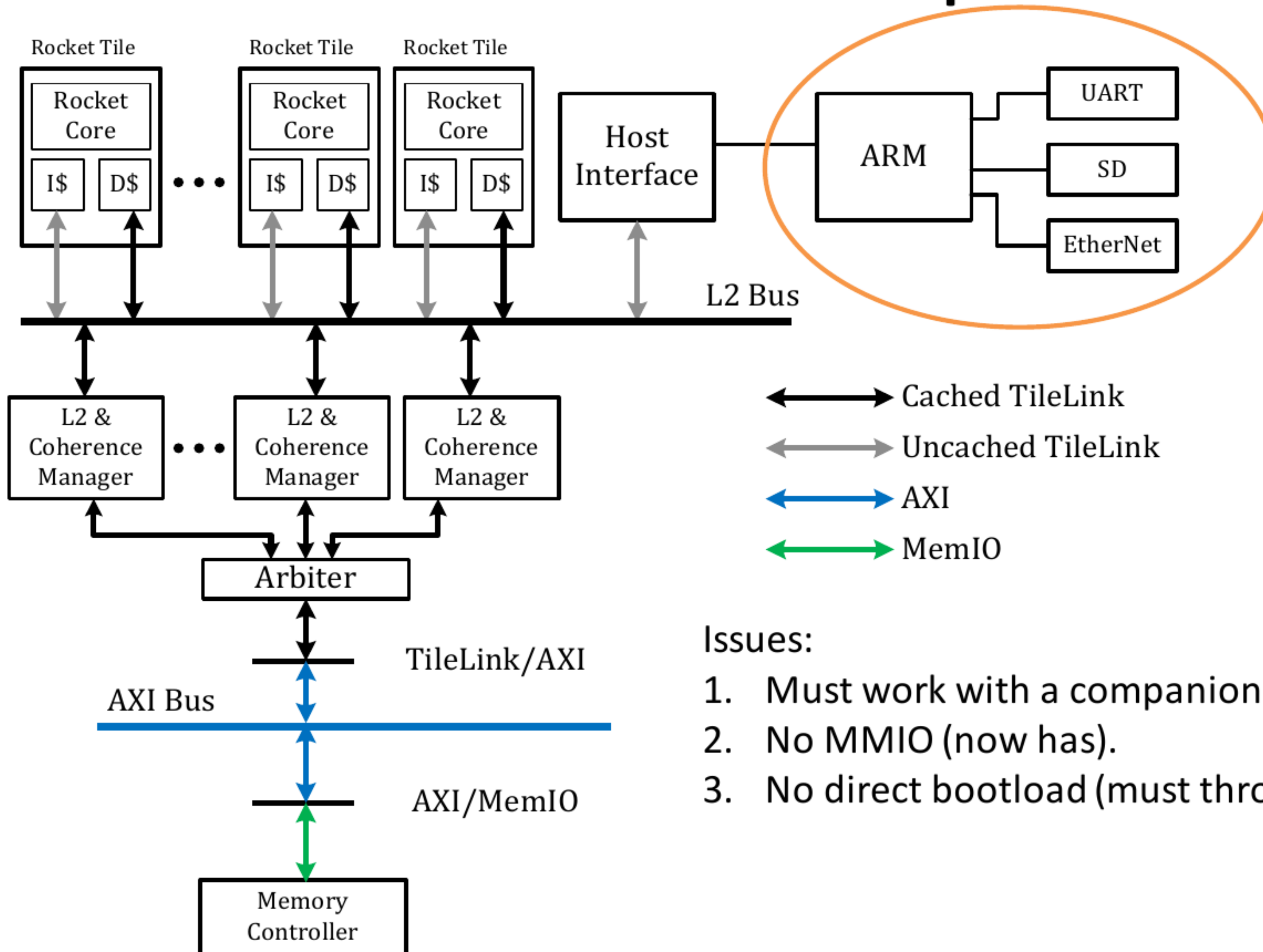
- El tutorial que vamos a desarrollar es la primera versión de lowRISC sin necesidad de estar conectado a un host, sino que es una arquitectura RISCV basada en ROCKET-CHIP que posee sus propios periféricos conectados a través de diferentes buses
- Esta es una versión sobre FPGA y, además, los periféricos son IP cores del fabricante de FPGAs (en este caso XILINX).
- Es, por tanto, una versión inicial sobre la que seguir trabajando, entre otras cosas, para ir sustituyendo los periféricos por Cores abiertos en la medida de lo posible
- **Localización del tutorial:**

  http://www.lowrisc.org/docs/untether-v0.2/



- Organización del tutorial:
  - Apartado 1: describe la arquitectura del sistema basada en Rocket-chip. Consta de varios subapartados.
  - Apartado 2: Dedicado a contar como preparar el entorno de desarrollo. Instalación de tootlchains y diferentes herramientas que se necesitan.
  - Apartado 3: Ejecución del desarrollo en sí: Simulaciones y demostración on-chip sobre FPGA.
- Presentación del tutorial en el III RISCV Workshop de Enero de 2016:
  - http://riscv.wpengine.com/wp-content/uploads/2016/01/Wed1115-untether_wsong83.pdf
  - https://www.youtube.com/watch?v=9BU5yNeyI5k
- Posteriormente, se incluyen transparencias por cada apartado con alguna información relevante para el desarrollo del mismo.

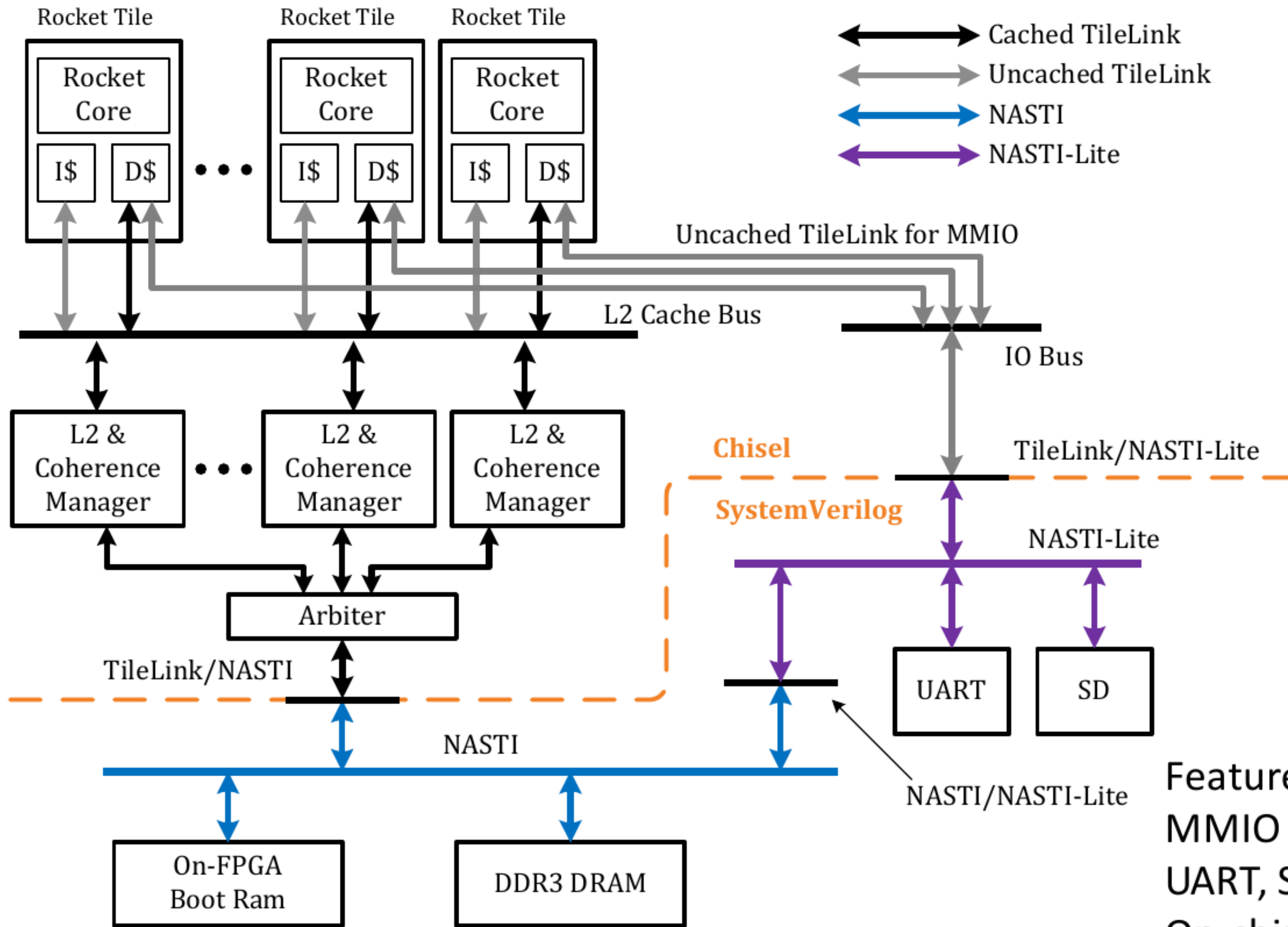
Rocket Chip



Issues:

1. Must work with a companion core (ARM).
2. No MMIO (now has).
3. No direct bootload (must through ARM).

Untethered Rocket Chip



Features:
MMIO
UART, SD, DDR3, Boot RAM
On-chip NASTI interconnect in SystemVerilog

I/O and Memory Map

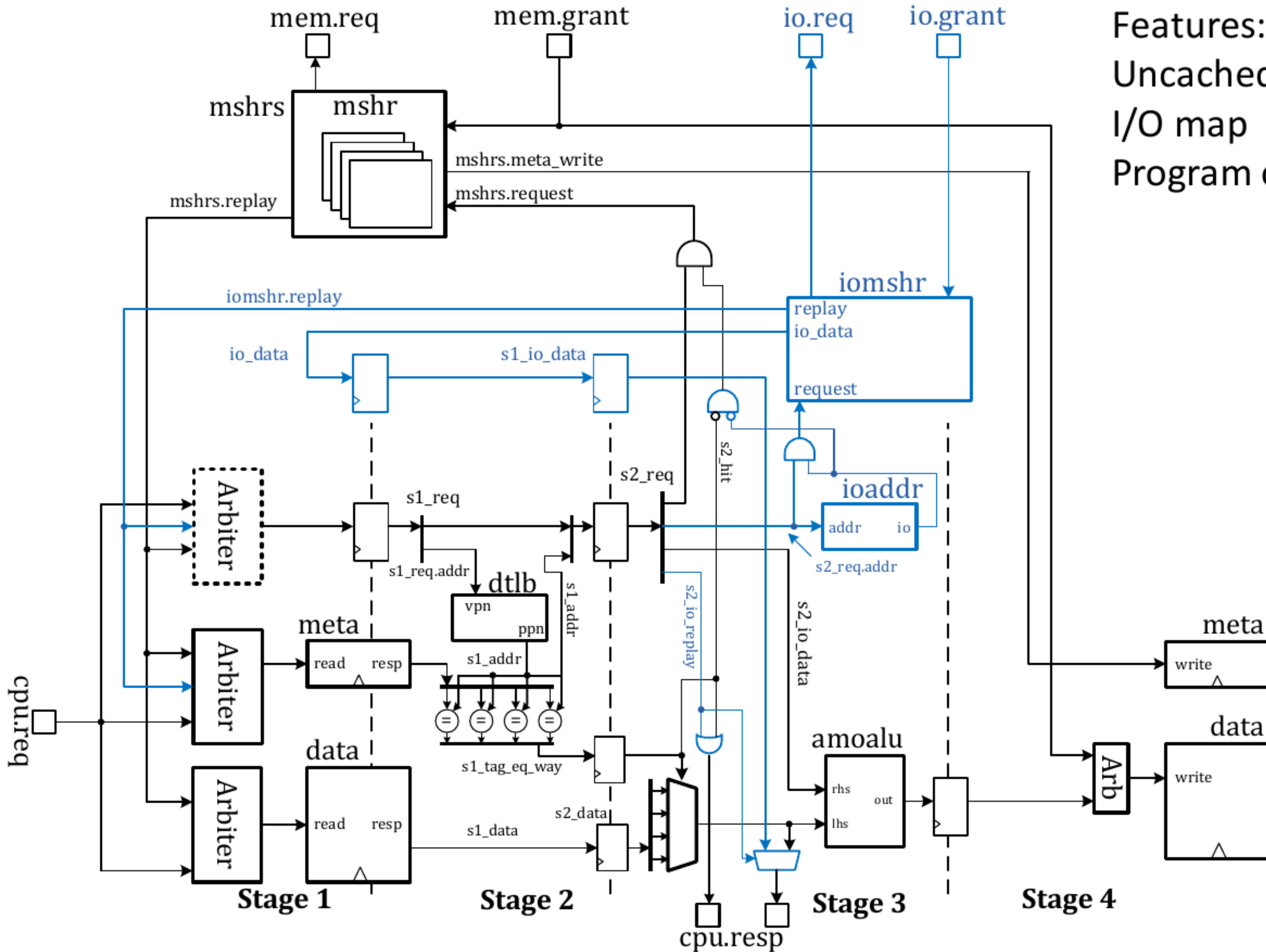
- I/O map
  - 4 I/O sections
  - CSR: io_base, io_mask, io_update

```
hit = (addr & ~io_mask) == io_base
```

- Memory map
  - 4 memory sections
  - CSR: mem_base, mem_mask, mem_phy, mem_update

```
hit = (addr & ~mem_mask) == mem_base
addr' = (addr & mem_mask) | mem_phy
```

MMIO



Features:

Uncached (bypass L1 & L2)

I/O map

Program order

Bootloader

- Two stage bootloaders
  - First stage bootloader
    - Copy the second stage bootloader to DDR RAM
    - Uncached copy (mapping DDR RAM to IO)
    - Re-map DDR RAM to memory address 0
    - Reset Rocket
  - Second stage bootloader
    - Revised Berkeley bootloader (BBL)
    - Driving I/O devices
    - Start multi-core, VM support
    - Load and boot RISC-V Linux in virtual address space

A Code Release

- The untethered Rocket chip has been released.
  - A tutorial: http://www.lowrisc.org/docs/untether-v0.2/
  - Code repo https://github.com/lowRISC/lowrisc-chip
- Key Features
  - FPGA demo with RISC-V Linux
    - Xilinx Kintex-7 KC705 suite (developing system)
    - Digilent NEXYS4-DDR board (low-end board) 320 USD
  - Up-to-date Rocket code from Berkeley
    - Merged all updates up to October 2015.
  - Nearly free development environment
    - Replace VCS with **Verilator**/ISim
    - Voucher or **WebPACK** Vivado license

Summary of the Code Release

- Remove host target interface
- Add reconfigurable I/O and memory maps
- Add memory mapped IO
- Rewrite TileLink/NASTI interfaces
- Provide on-chip NASTI interconnects
- Integrate DDR2/3 controller, SD (FAT32), UART
- 2 bootloader
- New design environment using free tools
- New make files and scripts

- Tagged memory to be re-integrated
- No support for Zedboard

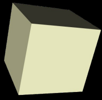
http://www.lowrisc.org/docs/untether-v0.2/release/

Future Works

- Looking for help to remove HTIF in RISC-V Linux
- Re-integrate tagged memory
- Add an interrupt controller
- Add trace debugging (with help from Stefan Wallentowitz)
- Add run-control debugger (SiFive)
- Platform spec

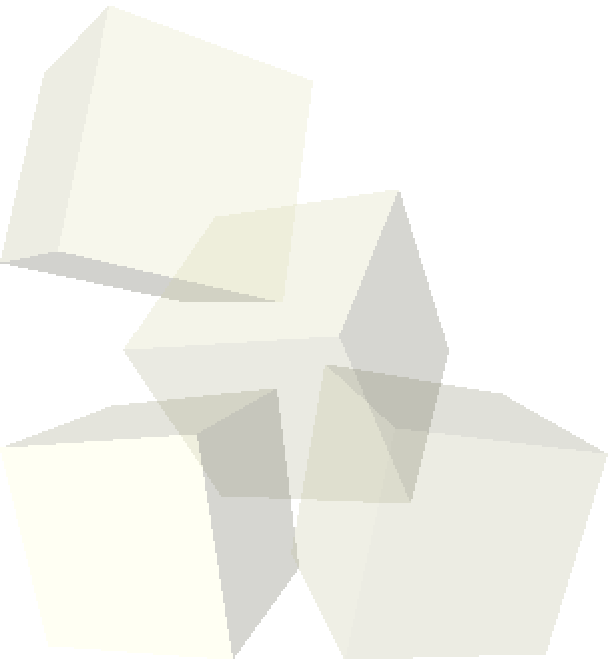
- For more information

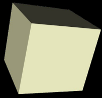
  Visit http://www.lowrisc.org/



lowRISC Untether

- APARTADO 1: Overview of the Rocket chip
  - Explica las características del diseño de Rocket-Chip en la versión “Untether”.
  - Es interesante leer esta documentación durante los tiempos de compilación de las herramientas

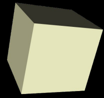




- Apartado 2: The development environment
  - Esta dedicado a descargar el repositorio y a contar la instalación de las diferentes herramientas del entorno de desarrollo.
  - En la página principal de este apartado 2 “A guide to the development environment” tenemos varios subapartados:
  - Subapart: **“System requirement”**
    - **IMPORTANTE:** Ejecutar el comando:

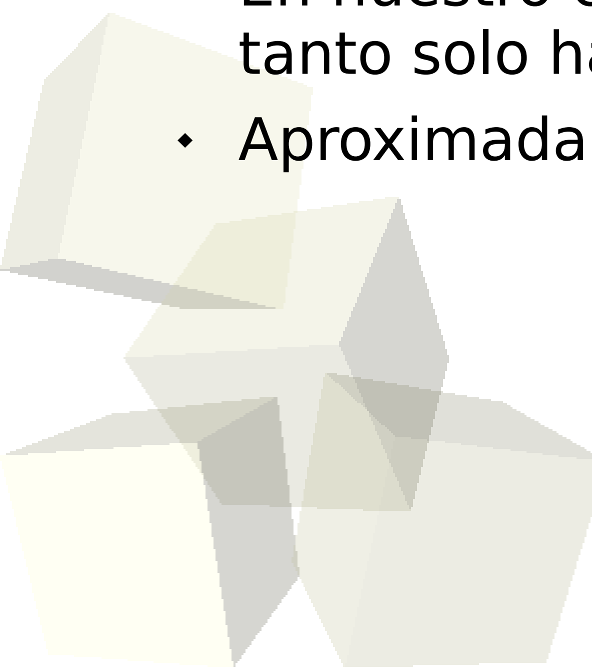
      $ sudo apt-get install XXXX
  - Subapart: **“Download the code release”**
    - Comienza con los comandos git para descargar el repositorio.
    - **IMPORTANTE:** En principio seguiremos los comandos de descarga del repositorio. Si algun proceso de instalación se enlenteciera demasiado, descargaremos el repositorio del siguiente enlace:

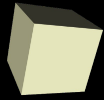
      http://10.1.15.78/~bellido/lowrisc-unthether.tar.gz
    - **IMPORTANTE:** Script para inicialización de variables de entorno: set_riscv_env.sh: cambiar kc705 por nexys4



lowRISC Untether

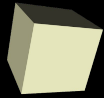
- Apartado 2: The development environment
  - Subapart: **“Compiling and installation of individual tools/packages”**
    - Nos lleva a cuatro enlaces para instalar las herramientas:
      - XILINX Vivado
      - VERILATOR (simulador verilog)
      - Toolchains para riscv (elf y linux)
      - Compilar kernel de linux y crear rootfilesystem
  - En nuestro caso, tenemos instalado XILINX Vivado y, por tanto solo haremos los otros tres apartados.
  - Aproximadamente debe tomarnos alrededor de una hora



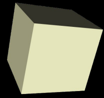


- **Install Verilator**
  - Instalaremos la ultima versión disponible según las indicaciones del enlace que aparece en la página del tutorial:
    - http://www.veripool.org/projects/verilator/wiki/Installing
  - En esta página indica dos metodos: Por git, o por tar.gz
  - Cualquiera de los dos es válido.
  - **RECOMENDACIÓN:** Instalar verilator en /opt/
  - **RECOMENDACIÓN: No** es necesario ejecutar el último comando indicado en cualquiera de los dos casos (sudo make install)
  - Volviendo al tutorial subapartado install verilator:
  - Nos indica las variables de entorno que debemos configurar adecuadamente para poder hacer simulaciones RTL
    - export VERILATOR_ROOT=<path de la carpeta instalación>
    - export PATH=$PATH:$VERILATOR_ROOT/bin

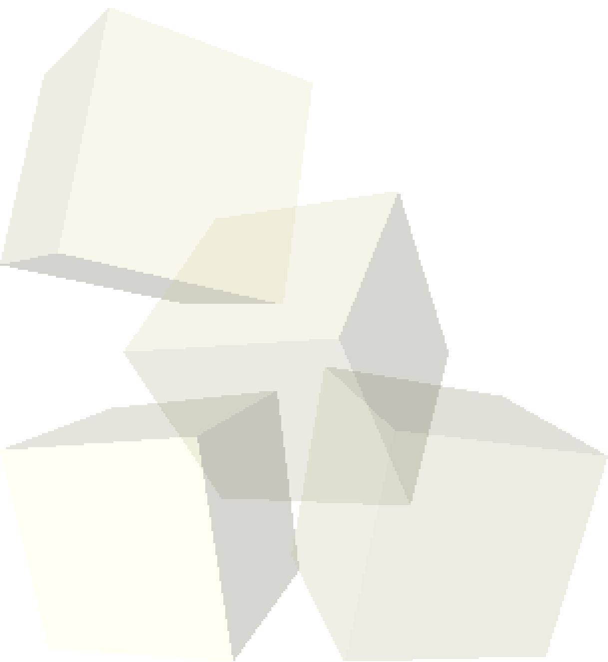
- **"Compile and install RISC-V cross-compiler"**
  - Procedimiento para instalar los toolchains modo baremetal y linux
  - **Subapart:** Building the RISC-V tools: Compilar el tootlchain modo bare metal
    - Hay que tener inicializadas las variables de entorno de RISCV (script set_riscv_enc.sh)
    - **Tiempo de compilación del toolchain (build.sh)**:14-18m
  - **Subapart:** Building GCC for RISC-V: Compilar toolchain en modo linux
    - NUNCA usar make -j .....SIEMPRE añadir un numero a opción -j (como maximo 6)
    - **Tiempo de compilación del toolchain**:15-16m

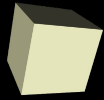


- **Compile the RISC-V Linux and the ramdisk \`root.bin\`**
  - **Subapart:** RISC-V Linux : Compilar linux para lowrisc
    - Tarda un cierto tiempo en descargar los fuentes
    - En la carpeta:
      - $TOP/riscv-tools/linux-3.14.41/
      - Ejecutar los comados make que indica el tutorial:
      - make ARCH=riscv defconfig
      - make ARCH=riscv -j vmlinux
  - **Tiempo de compilación del toolchain**:2-5m

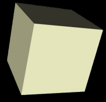


- **Compile the RISC-V Linux and the ramdisk `root.bin`**
  - **Subapart:** Ramdisk root.bin (busybox): Creando el sistema de ficheros
    - Seguiremos el tutorial
    - **IMPORTANTE**: El script make_root.sh debe ejecutarse dentro de la carpeta busybox-1.21.1/

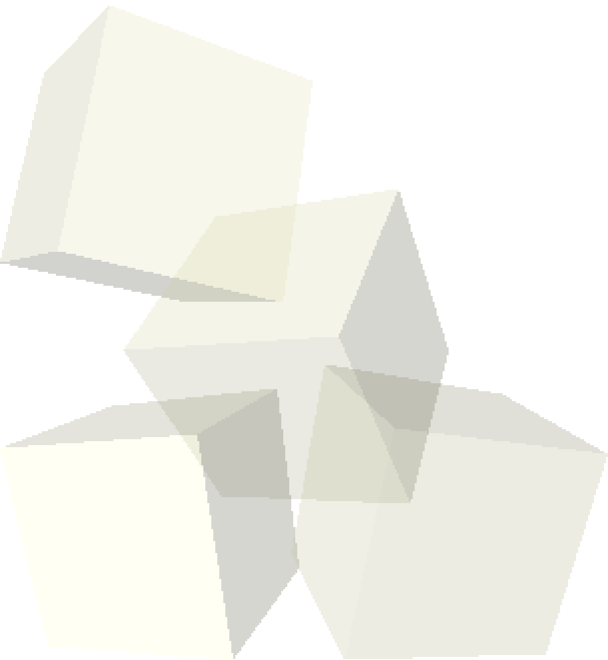


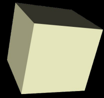


- APARTADO 3: "Simulations and FPGA Demo"
  - Consta de cuatro enlaces, tres para realizar diferentes tipos de simulaciones y otra para la implementación sobre FPGA
    - Behavioural Simulation (Spike)
      - A fast instruction level simulator. Peripheral support from the front-end server (not compatible with the FPGA implemantion).
    - RTL simulation
      - RTL-level simulation for the whole lowRISC SoC provided by Verilator.
    - FPGA demo
      - A RISC-V Linux demo on KC705/NEXYS4-DDR.
      - Peripherals (NEXYS4-DDR): 128MB DDR2 DRAM, UART, MicroSD+FAT32.
    - FPGA simulation
      - Pre-synthesis FPGA simulation for the whole lowRISC SoC provided by Xilinx ISim (a part of Xilinx Vivado).
      - Providing the full peripheral simulation with different configuration options.

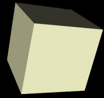


- Behavioural Simulation (Spike)
  - Subapart: Running spike
    - Funcionan las simulaciones en modo bare metal y arrancando linux

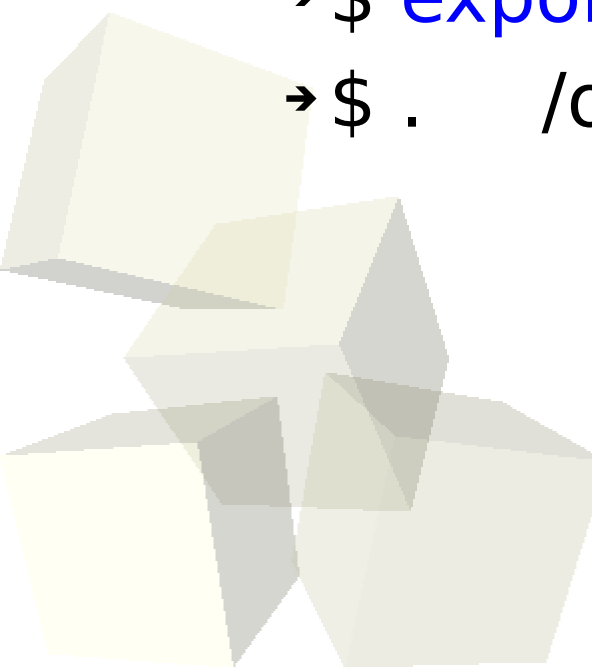


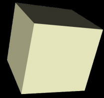


- RTL Simulation
  - Simulación con verilator
  - Subapart: Compilation
    - Ejecutar tanto make sim, como make sim-debug
    - **Tiempo de compilación total**:20-22m
    - Esto genera dos ejecutable que son los simuladores: DefaultConfig-sim y DefaultConfig-sim-debug
    - Estos ejecutables pueden copiarse a la carpeta $TOP/riscv/bin/
  - Subapart: Running simulations
    - Hay que ejecutar los comandos indicados dentro de la carpeta:
      - $TOP/riscv-tools/riscv-tests/



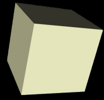
- **FPGA DEMO**
  - Debe de funcionar todo lo que nos indica el tutorial.
  - Asegurar que el scritp set_riscv.env.sh tiene como placa de FPGA la nexys4
  - Conviene activar la licencia de XILINX, además de sus variables de entorno:
    - $ export LM_LICENSE_FILE=2100@10.1.15.78
    - $.   /opt/Vivado/......../settings.sh





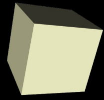
FPGA DEMO

- El proceso de generación de bitfile se realiza con:
  - $ make bitstream
  - **IMPORTANTE:** debido a que se emplea la versión de vivado 2016.2 y el tutorial esta preparado para la versión 2015.4 hay que hacer unas modificaciones en un fichero para que funcione:
    - Editar el fichero: $TOP/fpga/board/nexys4/script/make_project.tcl
      - 1.- Buscar: Memory Controller
      - En la línea del comando create_ip cambiar version 2.4 por 4.0
      - 2.- Buscar: Clock generator
      - En la línea del comando create_ip cambiar version 5.2 por 5.3
  - **Tiempo de compilación total**:20-25m



FPGA DEMO: Ejemplos sobre FPGA

- Existen varios ejemplo preparados para ejecutar en modo stand-alone sobre la FPGA: hello dram sdcard boot
- Se debe ejecutar y comprobar el correcto funcionamiento
  - Como hyperterminal puede instalarse gtkterm:
    - $ sudo apt-get install gtkterm
- Para sdcard y boot hay que preparar la tarjeta microsd adecuadamente
  - Una partición tipo FAT32 formateada.
  - Se puede hacer con la aplicación discos
- Posteriormente ejecutar LINUX:
  - Con imágenes precargadas
  - Con nuestras propias imágenes



- FPGA DEMO: Nuevo ejemplo software
  - Como ejercicio final:
    - Añadir a los ejemplos stand-alone preparados un nuevo ejemplo:
      - Hacer un programa que calcule los 100 primeros números primos
      - (se puede buscar en google el código C de cálculo de números primos y modificarlo adecuadamente)
    - Añadir el ejecutable de números primos en el root.bin para ejecutarlo en modo LINUX