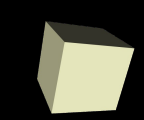
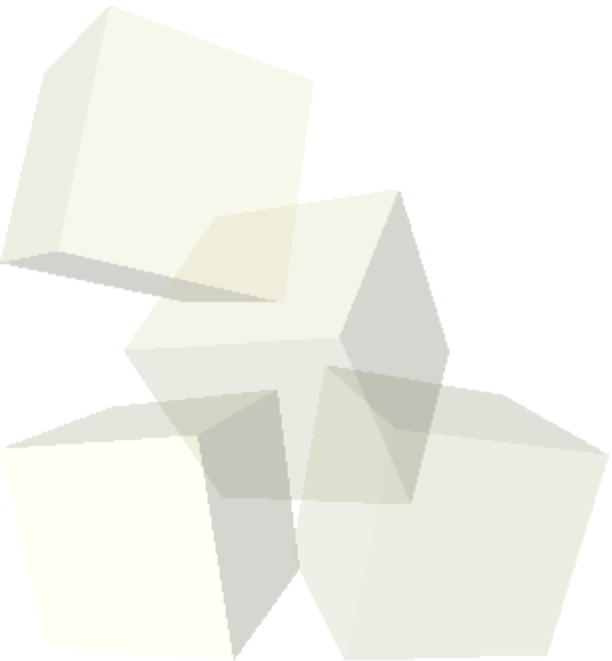


Lab 1: lowRISC tagged memory tutorial

Manuel J. Bellido Díaz

Marzo de 2017



lowRISC tagged memory tutorial

- LowRISC eligio como core del SoC la implementación de RISV64 denominada ROCKET CHIP:
  - https://github.com/ucb-bar/rocket-chip
  - https://github.com/ucb-bar/fpga-zynq
- El tutorial que vamos a desarrollar es una implementación de rocket-chip a la que se la ha introducido la extensión para soportar “Tagged Memory”. Esto implica:
  - modificaciones el hardware (se añada cache de TAGS, las cache de datos se extienden para almacenar los Tags;
  - se añaden nuevas instrucciones para manejar los tags, lo que implica dar soporte al toolchain para estas nuevas instrucciones

- **Localización del tutorial:**

  http://www.lowrisc.org/docs/tagged-memory-v0.1/

- **Organización del tutorial:**
  - Los **apartados 1, 2 y 3** son apartados para describir en que esta basado el sistema:
    - 1- visión de rocket chip;
    - 2- rocket core y su parametrización;
    - 3- como se da soporte a la “tag memory”
  - Los **apartados 4, 5 y 6** son donde se descargará y operará con el sistema:
    - 4 – configuración del sistema;
    - 5.- simulando el sistema;
    - 6.- probando sobre plataforma zedboard

- Desarrollo recomendado del tutorial:
  - Se propone comenzar con el apartado 4:
    - Descargar el repositorio lowrisc, y configurar lo necesario para trabajar: toolchains, linux kernel y rootfilesystem.
    - Emplear los tiempos de compilación para leer los apartados 1, 2 y 3.
- A continuación se incluyen transparencias por cada apartado con alguna información relevante para el desarrollo del mismo.

1.- Rocket chip overview

- En este apartado se hace una introducción rocket-chip.
- ROCKET CHIP es un generador de SoC con cores de arquitectura RISC-V desarrollado en la universidad de Berkeley
  - https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.pdf
- El proyecto lowRISC ha elegido este generador de SoC parametrizables para la implementación de lowRISC
- Se indica donde se encuentran los principales parámetros de configuración, su significado y sus posibles valores.

2.- Rocket core overview

- En este apartado se muestra la microarquitectura del core ROCKET
- Característica interesante:
  - Como se dice en el tutorial, ROCKET CORE posee una interfaz denominada ROOC para poder conectar un acelerador hardware especifico (una especie de co-procesador que se diseñe a medida). El acelerador implica:
    - Instrucciones especificas que se ejecutan en el procesador. Efectivamente, una de las características en la arquitectura RISC-V es que se pueden añadir instrucciones especificamente diseñadas para resolver un problema concreto
    - Diseño hardware del acelerador, donde se ejecuten esas instrucciones propias del mismo.
  - Existe un tutorial/documentación del trabajo desarrollado durante el verano de 2016 sobre añadir aceleradores en lowRISC con un ejemplo concreto de un codec MPEG-2:
    - http://www.lowrisc.org/docs/internship-2016/report/

3.- Tagged memory support

- Es el apartado principal del tutorial, pues en el se cuenta como se le da soporte a la “Tagged Memory” en rocket-chip.
- Se cuenta en este apartado las características de la memoria cache de tags y se introducen las nuevas instrucciones para cargar y almacenas tags
- Existe un subapartado donde se cuentas paso a paso como se da soporte en hardware y software a las nuevas instrucciones:
  - http://www.lowrisc.org/docs/tagged-memory-v0.1/new-instructions/
- Existe otro subapartado donde se cuenta los tests que estan preparados para la “tagged memory”:
  - http://www.lowrisc.org/docs/tagged-memory-v0.1/tag-tests/

4.- A guide to setting up the development environment

- Apartado para descargar el repositorio, configurar toolchains, compilar kernel de linux y crear rootfilesystem.
- “Downloading the lowRISC chip repository”
  RECOMENDACIÓN:
  - Intentaremos descargar el repositorio empleando los comandos de git que aparecen en el tutorial.
  - Si las descargas son lentas, de forma alternativa lo vamos a descargar del siguiente enlace:

    http://coria.dte.us.es/~bellido/lowRISC.tar.gz

    → http://10.1.15.78/~bellido/lowRISC.tar.gz
  - Extraer en cuenta de usuario
  - Debe de ejecutarse el comando apt-get install xxx
  - En la carpeta lowrisc-chip/ existe un script preparado para inicializar correctamente las variables, set_riscv_env.sh:
    Importante: Ejecutar el script dentro del directorio lowrisc-chip/

4.- A guide to setting up the development environment

- Building the RISC-V tools: Compilar el tootlchain modo bare metal
  - Hay que tener inicializadas las variables de entorno de RISCV (scrip set_riscv_enc.sh)
  - **Tiempo de compilación del toolchain (build.sh)**:12-14m

- Building GCC for RISC-V: Compilar toolchain en modo linux
  - Si se descargó de coria el repositorio de lowRISC ya esta hecha la descarga de riscv-gcc

    lowrisc-chip/riscv-tools/riscv-gcc
  - **Tiempo de compilación del toolchain**:15-16m

- Building the RISC-V Linux Kernel: Compilar linux para lowrisc
  - Si la descarga del kernel de linux es lenta podemos descargarlo desde:

    http://coria.dte.us.es/~bellido/linux-3.14.13.tar.gz
  - Descargarla y extraer en lowRISC-chip/riscv-tools/
  - Una vez descargada se pueden ejecutar los comando **make** que indica el tutorial.
  - **Tiempo de compilación del toolchain**:2-5m

- Build the root image (root.bin) for the Linux kernel:
  Creando el sistema de ficheros
  - Crearemos el sistema de ficheros para el kernel de linux.
  - Seguiremos el tutorial solo con una modificación:
  - **IMPORTANTE:** Hay que modificar el script $TOP/riscv-tools/make_root.sh para cambiar la línea:

    curl -L http://riscv.org/install-guides/linux-inittab > inittab
  - Por:

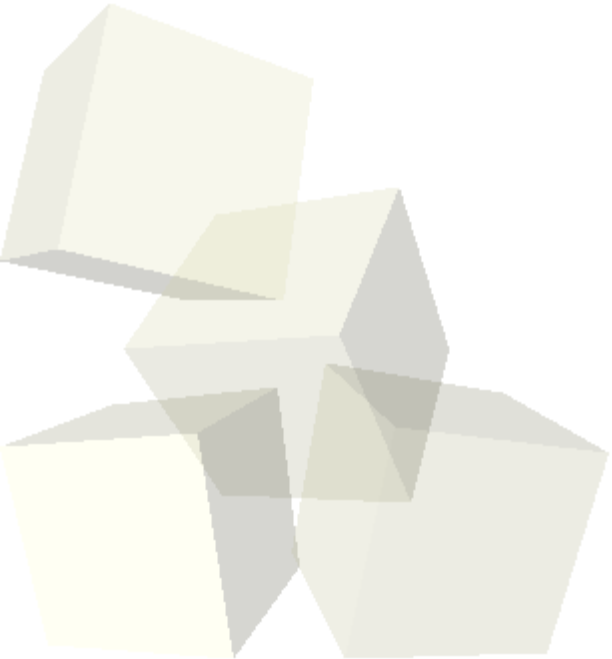
    curl -L https://www.ocf.berkeley.edu/~qmn/linux/linux-inittab >inittab
  - **IMPORTANTE**: El script make_root.sh debe ejecutarse dentro de la carpeta busybox-1.21.1/

5.- Running simulations

- Este aparatado menciona las posibles simulaciones que se pueden llevar a cabo:
  - Using the Spike Simulator
  - Using the C++ emulator generated by Chisel
  - Simulating the Verilog (ASIC target) generated by Chisel
  - Simulating the Verilog (FPGA target) generated by Chisel
- Podremos relizar simulaciones del sistema con SPIKE y con el **emulador C++**
- No realizaremos las simulaciones de código verilog ya que están preparadas para realizar con synopsys VCS.

Running simulations using Spike

- Se puede seguir el tutorial y debe funcionar todo.



Using the C++ emulator generated by Chisel

- Lo primero es compilar el emulador. El tutorial nos indica dos formas básicas: make o make debug
- Posteriormente nos indica como se pueden ejecutar los diferentes test que ya esta pre-construidos, empleando el comado make con alguna opción.
- Lo más interesante son los tres subapartados siguientes:
  - Bare metal mode
  - With proxy kernel and newlib
  - With a full linux system
- Hay que realizar las simulaciones propuestas en cada uno de estos tres casos.

6.- Running tests on the Zedboard FPGA

- En el tutorial hay una primera parte para comprobar que funciona lowRISC sobre la FPGA con imágenes preparadas, y una segunda parte donde se muestra como generar cada uno de los ficheros que se necesitan.
- Primero realizaremos la prueba con la zedboard con las imágenes o ficheros ya preparados y que deben de estar descargados en la carpeta correspondiente
- El segundo caso tiene el problema de que para generar el bit file los scripts de lowRISC están preparados para emplear VIVADO2014.4 y tenemos instalado VIVADO 2016.2
- Vamos a utilizar un pequeño “truco” que puede funcionar
  - Editar el fichero: lowrisc-chip/fpga-zynq/zedboard/src/tcl/zedboard_bd.tcl
  - Buscar la linea: set scripts_vivado_version 2014.4
    - Cambiar 2014.4 por 2016.2

Building the boot image from scratch

- Existe un script preparado para realizar todo el proceso de manera automática.
- Sin embargo, en nuestro caso, vamos a realizarlo paso a paso. Son cinco pasos:
  - 1. Generating the FPGA bitstream
    - Si el comando make rocket no va, porbar: make project
    - Si make bitstream no va, probar make vivado y seguir instrucciones de https://github.com/ucb-bar/fpga-zynq
  - 2. Export hardware information for SDK
  - 3. Building the First Stage Boot Loader (FSBL)
  - 4. Building u-boot for the Zynq ARM core
  - 5. Creating the FPGA boot image (boot.bin)
    - Este último paso lo realizaremos siguiendo las instrucciones del siguiente enlace:
      - https://github.com/ucb-bar/fpga-zynq#35--creating-bootbin
      - Lo importante para nosotros es obtener el fichero boot.bin que incorpora el nuevo bitfile de la FPGA de la Zynq

Building the boot image from scratch

- No es necesario realizar los subapartados del tutorial:
  - Modifying the contents of the RAMdisk
  - Building the front-end server
- Básicamente lo que haremos es lo mismo que se realizó con las imágenes ya precargadas, sustituyendo el fichero boot.bin por el nuevo fichero boot.bin que hemos generado