Curso de Diseño de Sistemas Empotrados basados en OPENRISC (plataforma ORPSOC-FUSESOC)

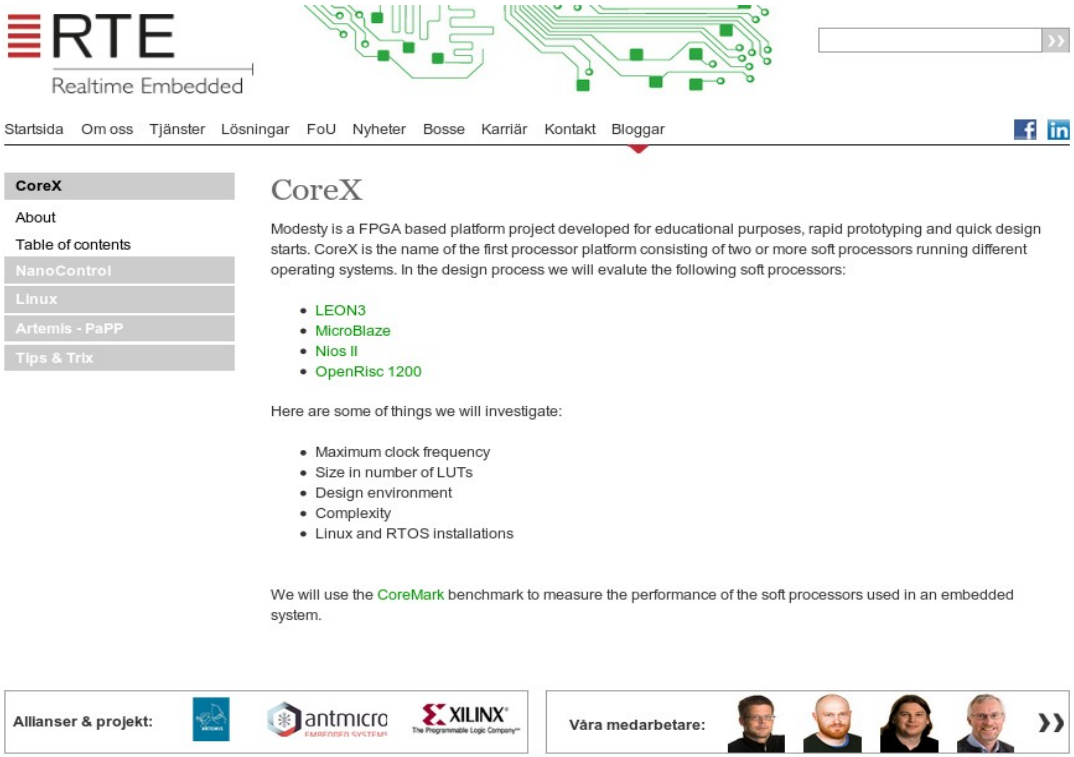
SoC Basados en Sistemas Abiertos
Máster Ingeniería de Computadores y Redes
(Esp. Sistemas Empotrados)

Manuel J. Bellido Díaz

Mayo de 2017

Curso en versión antigua de ORPSOC (v2)

- Existe un Curso On-line de diseño de sistemas empotrados con Openrisc basado en versión ORPSOCv2:http://www.rte.se/blog/blogg-modesty-corex/index
  - El curso introduce a la implementación de la plataforma ORPSOC sobre FPGA



2 OpenRISC 1200

2.1 OpenRISC 1200 soft processor
2.2 Digilent Atlys SPARTAN-6 development board
2.3 Using ORPSoC
2.4 Simulating ORPSoC using ISim
2.5 Writing an application program
2.6 Programming the SPI flash memory
2.7 Loading and executing a program
2.8 Adding a serial terminal
2.9 Installing U-boot the universal bootloader
2.10 Using U-boot
2.11 Installing Linux
2.12 Benchmarking OpenRISC 1200
2.13 Debugging the OpenRISC 1200
2.14 Using orbuild
2.15 eCos real time operating system

Aspectos generales del desarrollo en Pcs aula G0-30

- IMPORTANTE: El curso se va a ir desarrollando de forma acumulativa. Es decir, el final del trabajo de un día será el punto de partida del siguiente.
  - Seleccionar el PC sobre el que se vaya a trabajar en el resto del curso.

- Para desarrollar el curso correctamente debemos disponer de una cuenta con permisos de superusuario:
  - Aula G0-30: Hay que arrancar Linux :
    - **Usuario**: practicas **passwd**: practicas
  - El usuario practicas esta en el grupo admin lo que implica poder ejecutar comandos con “sudo” (como superusuario)

Aspectos generales del desarrollo en Pcs aula G0-30

- Además hay que reconfigurar la conexión a internet de nuestro host (PC) para poder tener pleno acceso:
  - Crear una conexión de ethernet nueva con los parámetros:
    - IP: 10.1.15.xx (cada pc un numero distinto:1,2,3,....)
    - Netmask: 255.255.252.0
    - Gateway: 10.1.15.78
    - DNS: 8.8.8.8

Curso sobre ORPSOC v3 con FUSESOC

- El curso que vamos a desarrollar es con ORPSOC v3
  - Información general: https://www.openrisc.io/
- Componentes a instalar:
  - Toolchains: herramientas de compilación de software
  - Fusesoc: herramienta para trabajar con los diseños de orpsocv3
  - ORPSOC-CORES: repositorio de cores y systemas preparados
  - OPENOCD: Herramienta de debug proxy para poder hacer el debug del sistema
  - ISE : herramienta de síntesis para FPGA de XILINX
- Preparación del sistema:
  - En los pcs del aula G0-30 en ubuntu 14.4 faltan algunos paquetes para que todo funcione:
  - $ sudo apt-get install autoconf libusb-1.0.0-dev subversion

Instalando TOOLCHAIN

- Instalaremos unos toolchains ya precompilados:
  - https://github.com/openrisc/newlib/releases -->
  - Descargar: or1k-elf_gcc6-20160228_binutils2.26_newlib2.4.0_gdb7.11.tgz
- Extraer en /opt/
  - /opt/or1k.elf/
  - Para que funcione en un terminal:
  - $ export PATH=/opt/or1k-elf/bin:${PATH}
- Se puede probar a compilar hello.c:
  - $ or1k-elf-gcc hello.c -o hello
- Compilar software para la plataforma ATLYS:
  - $ or1k-elf-gcc -mboard=atlys hello.c -o helloatlys

Instalando FUSESOC

- Informacion sobre Fusesoc:
  - https://github.com/olofk/fusesoc
  - Instalacion con PIP
    - (Hay que instalar pip en el PC:
      - $ sudo apt-get install python-pip)
- Para inicializar fusesoc se ejecuta:
  - $ fusesoc init
  - Al ejecutar este comando nos pide que indiquemos las carpetas para alojar los orpsoc-cores y los fusesoc-cores
  - Podemos crear una carpeta de trabajo en algún directorio y, en esa carpeta ir instalando los diferentes paquetes que se necesiten.
- Podemos ejecutar fusesoc list-systems para ver que sistemas estan preparados. Entre ellos esta la implementación sobre ATLYS de ORPSOC que es la que llevaremos a cabo.

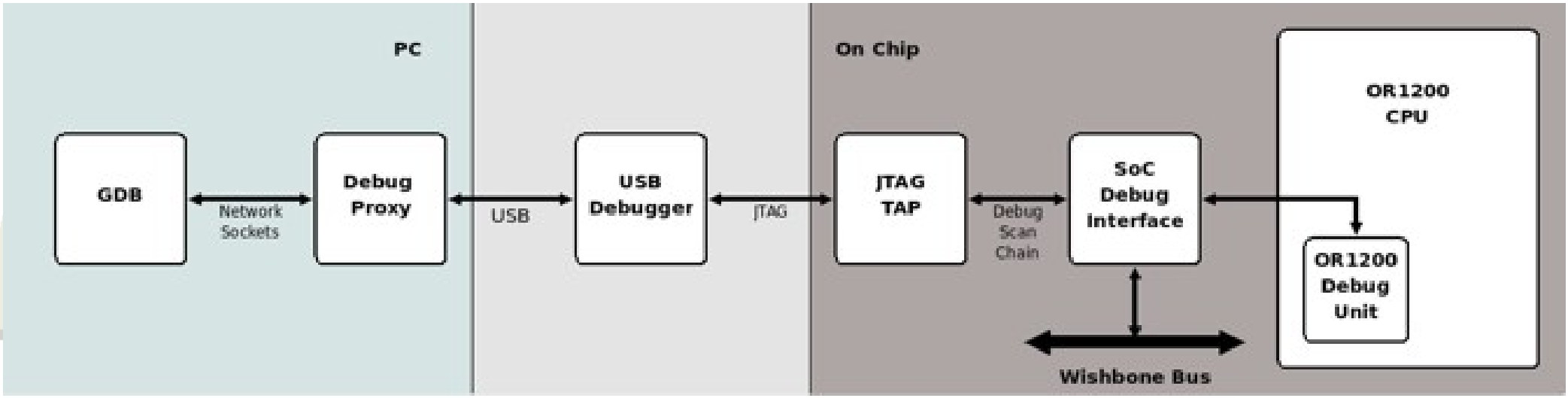
Usando Fusesoc para sistema ORPSOC sobre ATLYS

- Fusesoc esta preparado para diversas placas de desarrollo entre ellas la placa ATLYS de Digilent con FPGA de Xilinx
- Vamos a realizar algunas modificaciones en el diseño del sistema de orpsoc sobre atlys para que todo funcione correctamente:
  - Editar fichero de defines para eliminar VGA:
    - $ cd /opt/orpsoc-cores/systems/atlys/rtl/verilog/include
    - $ gedit orpsoc-defines.v
  - Comentar o borrar linea de define VGA
  - Eliminada la VGA hay que eliminar los pines de entrada salida que están asociados a la VGA en el sistema:
    - $ cd /opt/orpsoc-cores/systems/atlys/data
    - $ gedit atlys.ucf
    - Comentar todas las líneas relativas a HDMI que es la salida que se usa para la VGA en esta placa de desarrollo

Usando Fusesoc para sistema ORPSOC sobre ATLYS

- Ejecutar fusesoc para construir SoC para ATLYS:
  - Activar las herramientas de XILINX:
    - $. /opt/Xilinx/14.7/ISE_DS/settings64.sh
    - $ export LM_LICENSE_FILE=2100@10.1.15.78
  - Ejecutar fusesoc:
    - $ cd <carpeta_de_trabajo>/fusesocbuild
    - $ fusesoc build atlys
- **TIEMPO DE COMPILACIÓN: 15-20 minutos**

- Sistema de debug para SoC
  - Unidad de debug conectada al bus principal del SoC (hw)
  - Cable de interconexión entre UD y host (cable JTAG)
  - Protocolo de comunicación entre el cable y la UD
  - Sistema software de debug, habitualmente GDBserver



- Sistema de debug para ORPSOC v3 en placa ATLYS
  - Unidad de debug : Advanced Debug Unit
    - http://opencores.org/project,adv_debug_sys
    - JTAG TAP: Unidad MOHOR
  - Cable de interconexión entre UD y host (cable JTAG):
    - C232HM-DDHSL
    - http://www.ftdichip.com/Support/Documents/DataSheets/Cables/DS_C232HM_MPSSE_CABLE.PDF
  - Protocolo de comunicación entre el cable y la UD:
    - Emplearemos OPENOCD
      - http://openocd.org/
  - Sistema software de debug, habitualmente GDBserver

Instalando y preparando OPENOCD

- Instalaremos y compilaremos la última versión de OPENOCD: ir a http://openocd.org/ y desde el sitio adecuado descargar openocdxxxx.tar.gz. Extraer en /opt
- Compilar OPENOCD:
  - $ cd <carpeta_de_trabajo>/openocd-0.10.0
  - $ ./configure --enable-ftdi
  - $ make
  - Opcional: $ make install
- Configurando cable C232HM:
  - Fichero de configuración del cable:
    - Se encuentra en la carpeta:
      - tcl/interface/ftdi :
        - um232h.cfg

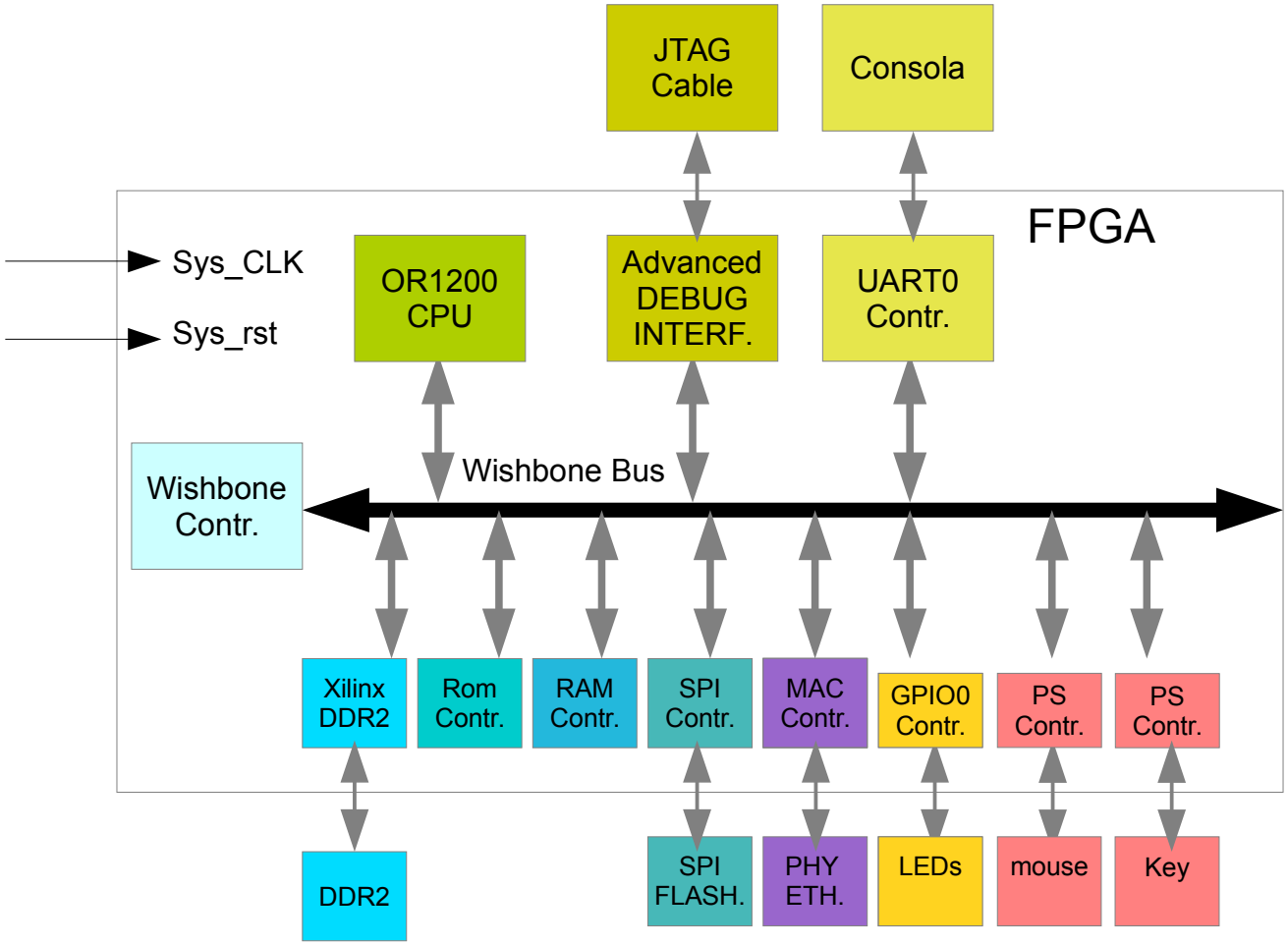
Instalando y preparando OPENOCD

- Configurar board para ORPSOC en OPENOCD:
  - $ cd /opt/openocd-0.9.0/tcl/board
  - $ gedit or1k_generic.cfg
  - Modificar dos cosas:
    - Cambiar TAP de VJTAP a MOHOR
    - La ruta del fichero or1k.cfg no es correcta hay que modificarla para que encuentre el fichero
- Comprobar que OPENOCD funciona:
  - $ cd <carpeta_de_trabajo>/openocd-0.10.0/
  - $ sudo src/openocd -f tcl/interface/ftdi/um232h.cfg -f tcl/board/or1k_generic.cfg
- Procedimiento de debug con OPENOCD:
  - https://github.com/openrisc/tutorials/blob/master/docs/Debugging.md

Utilizando ORPSOC sobre ATLYS

- Conectado cables a Atlys:
  - USB-microusb en puerto de programación
  - USB-microusb en puerto UART (para hyperterminal)
  - Cable C232hm en puerto PMOD (mirar fichero ucf de atlys para conocer los pines de conexión del cable)
  - Cable de alimentación
- Programando ATLYS:
  - Programa Impact con cable usb en puerto microusb de programación
- Ejecutar software:
  - En un terminal ejecutar OPENOCD
  - En otro terminal conectarse a ORPSOC:
    - $ telnet localhost 4444
    - Descargar software y ejecutar (abrir hyperterminal por ejemplo gtkterm -115200 baudios-)
  - Ejecutar: hello.c numeros primos.c

Arquitectura de ORPSOC sobre ATLYS



Arquitectura de ORPSOC sobre ATLYS

- La arquitectura de un sistema queda definida en la carpeta:
  - <carpeta_de_trabajo>/orpsoc-cores/systems/<nombresistema>
- Arquitectura de atlys definida en:
  - <carpeta_de_trabajo>/orpsoc-cores/systems/atlys/
  - **Ficheros:**
    - Atlys.core: Donde se definen los cores incluidos en el sistema
    - Atlys.system: 1.) Fichero UCF; 2.) FPGA especifica; 3.) Fichero con el diseño TOP
  - **Carpetas:**
    - data: con ficheros específicos del sistema
      - Atlys.ucf: Fichero UCF;
      - wb_intercon.conf: Fichero de configuración de la interconexión al bus WISHBONE
      - Atlys.dts, atlys_defconfig:Configuración de kernel de linux
    - rtl/verilog/: carpeta con ficheros específicos del diseño para este sistema

Arquitectura de ORPSOC sobre ATLYS

- Los Cores generales (que se emplean indistintamente en los diferentes sistemas) se encuentran en la carpeta:
  - /opt/orpsoc-cores/cores/
- Fichero **wb_intercon.conf:**
  - Define la interconexión del bus wishbone:
    - Interconexión del bus de instrucciones
    - Interconexión del bus de datos
    - En la interconexión del bus de datos queda definido el mapa de memoria del sistema (parámetro offset de cada periférico)
    - $ gedit /opt/orpsoc-cores/systems/atlys/data/wb_intercon.conf
    - Ejemplo: El GPIO0 esta posicionado en le mapa de memoria en:
      - 0x91000000
      - Y ocupa un total de dos direcciones:
        - 0X91000000 y 0x91000001

Manejando GPIO en ORPSOC

- Existe un periférico que maneja GPIOs en ORPOSOC:
  - Código de GPIO:
    - $ gedit /opt/orpsoc-cores/cores/gpio/gpio.v
  - Mapa de memoria en ORPSOC:
    - $ gedit /opt/orpsoc-cores/systems/atlys/data/wb_intercon.conf

- Cabecera para programas C de manejo de GPIO0 en ATLYS:

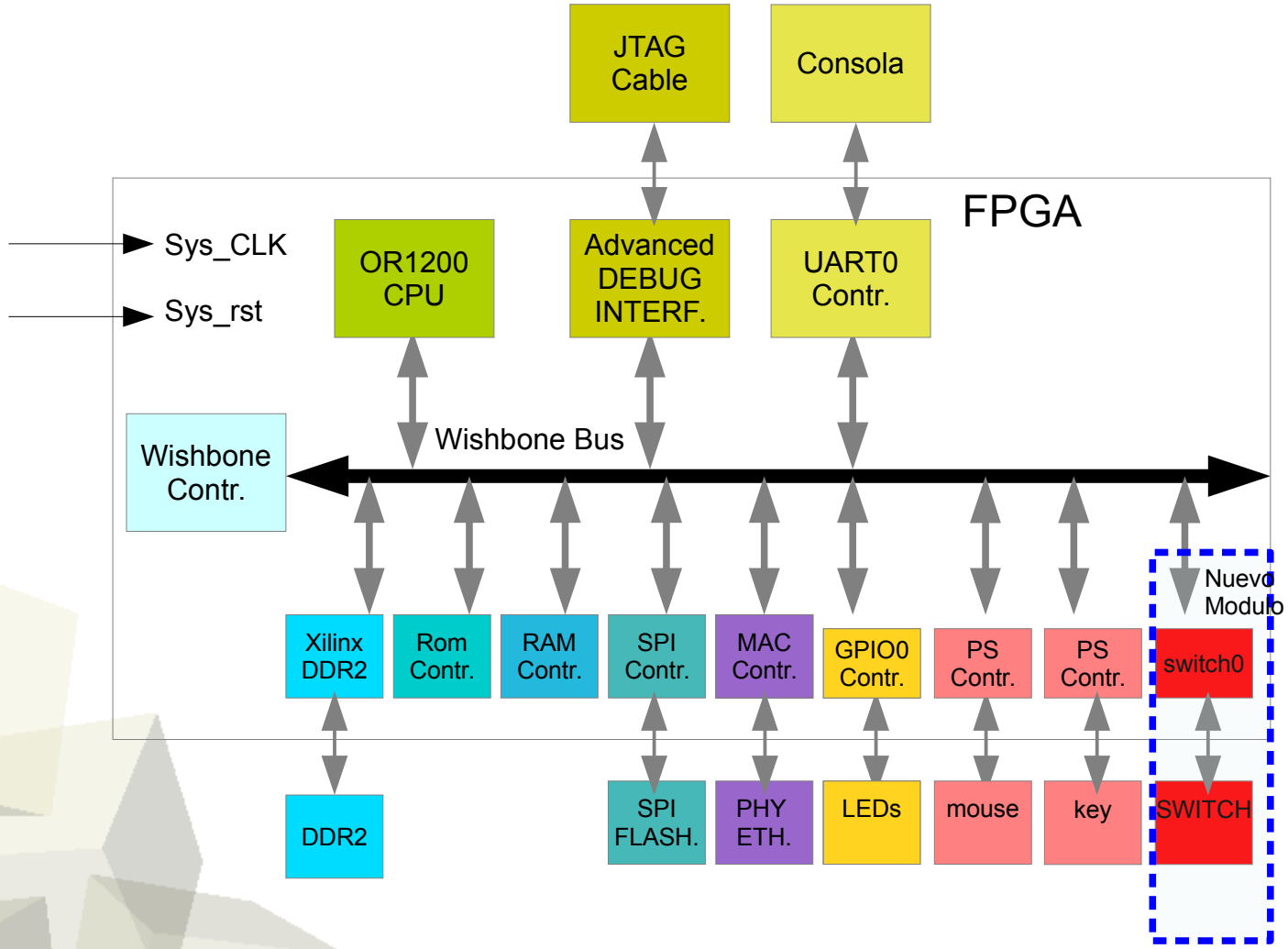
```
#define GPIO_0_BASE  0x91000000
#define writeGPIO(addr,val) (*(unsigned char*) (addr)=(val))
#define readGPIO(addr)      (*(unsigned char*) (addr))

int main (int argc, char *argv[])
{
```

  - **Desarrollar un código que encienda los leds impares**

Añadiendo un módulo RTL a ORPSOC

- Una de los cosas necesarias a la hora de adaptar la plataforma base de referencia, en nuestro caso ORPSOC, a las necesidades del sistema es ser capaz de añadirle los módulos RTL adecuados.
- Vamos a ver el proceso para poder añadir un nuevo módulo al SoC
- En nuestro caso vamos a desarrollar ese ejemplo añadiendo un nuevo módulo de control de switches (switch0) al SoC.

Añadiendo un módulo RTL a ORPSOC



Añadiendo un módulo RTL a ORPSOC

- Pasos para incluir nuevo módulo GPIO:
  - Se necesita crear el controlador con interfaz wishbone:
    - Crearemos un nuevo fichero de nombre switch.v:

      /opt/orpsoc-cores/systems/atlys/rtl/verilog/switch.v
    - Para hacer el controlador (switch.v) podemos copiar el controlador de GPIO (gpio.v) y modificarlo adecuadamente.
  - Diseñaremos un controlador de switch con un ancho de datos fijo de 8 (8 leds) y que solo tenga un único registro, el registro de lectura de los switches
  - Partiendo del código del controlador de GPIO es relativamente facil diseñar el código del controlador de switches

Añadiendo un módulo RTL a ORPSOC

- Una vez hecho el controlador hay que modificar un conjunto de ficheros:
  - En carpeta /opt/orpsoc-cores/systems/ atlys/rtl/verilog/
    - orpsoc_top.v: Incluir en module, señales externas de switch.v
      - Incluir señales internas de conexión al bus wishbone (similar a como se define para GPIO0)
      - Incluir el componente switch0 (similar a gpio0 pero con los cambios adecuados)
  - En carpeta /opt/orpsoc-cores/systems/atlys/data/
    - wb_intercon.conf:
      - Incluir modulo switch0 en interconexión con buses tanto con OR1k como con debug (similar a como esta gpio0)
      - Inlcuir slave para switch0:
        - Direcciones: 0xb8000000

Añadiendo un módulo RTL a ORPSOC

- Pasos para incluir nuevo módulo GPIO:
  - En carpeta /opt/orpsoc-cores/systems/atlys/
    - atlys.core:
      - Incluir ruta del código swtch.v
        - rtl/verilog/swicth.v
  - En carpeta /opt/orpsoc-cores/systems/atlys/data/
    - atlys.ucf
    - Conectaremos el nuevo switch0 a los switches

Añadiendo un módulo RTL a ORPSOC

- Pasos para incluir nuevo módulo GPIO:
  - Por último hay que regenerar la interconexión al bus wishbone añadiendo el nuevo módulo switch0 a esa interconexión.
  - La interconexión al bus esta en dos ficheros:
    - /opt/orpsoc-cores/systems/atlys/rtl/verilog/
      - wb_intercon.v
      - wb_intercon.vh
    - Si se editan estos ficheros puede comprobarse que son ficheros generados automáticamente.
    - Efectivamente hay un código de python que permite generar automáticamente estos códigos a partir del wb_intercon.conf
    - Se debe ejecutar en la carpeta atlys/rtl/verilog/

$ <Path_completo_carpeta_trabajo>/orpsoc-cores/cores/wb_intercon/sw/wb_intercon_gen <ruta_completa_wb_intercon.conf> **wb_intercon.v**

Añadiendo un módulo RTL a ORPSOC

- Una vez modificados todos los ficheros habrá que realizar el proceso de síntesis y generación de bitfile.
  - $ cd <carpeta_de_trabajo>
  - $ fusesoc build atlys
- Para comprobar el buen funcionamiento del nuevo GPIO se propone desarrollar un programa que controle el encendido de cada led con el switch correspondiente.
- Una vez hecho el código y compilado podemos comprobar el funcionamiento cargando el software por la unidad de debug (openocd)

Ejecutando Linux

- La arquitectura OPENRISC esta incluida en el mainline del Kernel de LINUX desde 2013
- El proyecto openrisc mantiene una versión de los fuentes del kernel en github:
  - https://github.com/openrisc/linux
- Para crear la imagen de LINUX para un determinado sistema-plataforma de openrisc existe un tutorial que cuenta los pasos a seguir:
  - https://github.com/openrisc/tutorials/blob/master/docs/Linux.md

Ejecutando Linux

- Aspectos a tener en cuenta en el tutorial:
  - La descarga del kernel con git puede ser lenta. Alternativa:
  - https://coria.dte.us.es/~bellido/linux-openrisc.tar.gz
  - En el tutorial hace referencia a dos ficheros *.dts y *_config.
  - Para el sistema ATLYS están en:

    <carpeta_trabajo>/orpsoc-cores/systems/atlys/data/
  - Para resolver el problema de Kernel Panic que puede ocurrir en la ejecución del kernel de LINUX hay que activar la opción de soporte de compresión gzip:
  - $ make menuconfig --> general setup y buscar..
  - Y volver a compilar el kernel

Toolchain para openrisc con Linux

- Si queremos desarrollar aplicaciones que se ejecuten sobre Linux es disponer del toolchain adecuado.
- Se pueden descargar versiones del toolchain precompiladas en el enlace:
  - https://github.com/openrisc/musl-cross/releases
  - Alternativamente:
    - https://coria.dte.us.es/~bellido/or1k-linux-musl_gcc5.3.0_binutils2.26_musl1.1.14.tgz
- Se debe extraer en /opt
- Para ejecurtarlo en un terminal:
  - $ export PATH=/opt/or1k-linux-musl/bin:${PATH}

- Una vez instalado el toolchain es posible compilar una aplicación con **or1k-linux**-musl-gcc

Instalando una aplicación propia en la imagen de Linux

- Una vez que se ha desarrollado la aplicación definitiva para el sistema empotrado nos queda como incrustarla en la imagen de Linux y que se ejecute automáticamente.
- Hay que tener en cuenta que, hasta ahora el sistema de ficheros principal (/) se monta en RAM (ram filesystem)
- En el kernel de linux, para arquitectura (~linux/arch/<arquitectura>/) existe una carpeta (support/initramfs/) con la estructura de carpetas y archivos que va a tener el rootfilesystem

Instalando una aplicación propia en la imagen de Linux

- Así, un procedimiento que puede seguirse para incluir una aplicación en la imagen de Linux consiste en copiar el ejecutable de la aplicación, por ejemplo, dentro de la carpeta usr/bin del initramfs
- Si pretendemos que se ejecute desde el momento de arranque de linux tendremos que editar el fichero donde se inicializan una serie de aplicaciones e incluir al final del mismo nuestra aplicación en modo background (& al final):
  - Fichero etc/init.d/rcS en initramfs

- Analizando el fichero dts (Device Tree System), esta configurado el siguiente hardware:
  - La consola de inicio
  - La memoria ram DDR
  - La cpu OPENRISC
  - El puerto serie
  - La interfaz ethernet
  - El controlador de interrupciones
- Sin embargo, no están configurados los GPIOs que si están implementados en el hardware y accesibles en el mapa de memoria (gpio0 en 0x91000000 y switch0 en 0xb8000000)
- **¿Es posble acceder a los GPIOs desde LINUX?**

Manejando GPIOs en LINUX: Accediendo a posiciones de memoria física en LINUX

- En Linux es posible acceder a la memoria física a través del dispositivo /dev/mem
- Existe un comando que permite acceder a posiciones físicas de memoria para leer y/o escribir:

  devmen
- Por otra parte, la función mmap permite asociar un puntero del espacio de usuario en linux (memoria virtual) a memoria física.

  http://elinux.org/EBC_Exercise_11b_gpio_via_mmap
- Un ejemplo de uso se encuentra en el siguiente código en C:

  https://coria.dte.us.es/~bellido/readswwriteleds-prv2-linux-memorydirectacces-g1y0.c

Manejando GPIOs en LINUX: Incluyendo GPIO en Device Tree

- El módulo hardware de GPIO que se emplea es un módulo sencillo para el que se ha desarrollado un driver para el kernel de linux: **jb-trivial**
- Para poner en marcha este módulo del kernel de linux son necesarias dos cosas:
  - Añadir en el fichero *.dts el gpio de la manera adecuada
  - Mirar como esta puesto en el fichero de0_nano.dts
  - Añadir el módulo jbtrivial a la hora de compilar el kernel de linux (y el sysfs)
- En los fuentes del kernel de linux existe un fichero dts para openrisc plataforma de0_nano que incluye la descripción hardware de GPIO. Se puede copiar al fichero atlys.dts.
- Con “make menuconfig” hay que seleccionar el driver **jb-trivial:** Device Driver ---> GPIO Support --> activar Opencores Jb-Trivial y sysfs interface

Manejando GPIOs en LINUX: Incluyendo GPIO en Device Tree

- Una vez compilado y ejecutado el kernel de linux los GPIOS están accesibles en el espacio de usuario. La forma de acceder a ellos se pueden encontrar en el siguiente enlace:

  http://wiki.lemaker.org/HiKey%28LeMaker_version%29:How_to_control_the_GPIO_on_the_SBC_boards

- También podemos ver los GPIOS que se crean en la capeta /sys/class/gpio/ de linux