Desarrollo y Depuración de Software

Zynq
Vivado 2018.1

Objetivos

- **Al completar este módulo el alumno será capaz de:**
  - Describir la arquitectura de los drivers
  - Distinguir entre drivers Nivel-1 y Nivel-2
  - Listar los tipos de timers de procesador
  - Entender la API timer de CPU
  - Describir la funcionalidad del depurador GNU (GDB)
  - Describir la funcionalidad del XMD (Xilinx Microprocessor Debugger)
  - Describir el framework Target Communications del Eclipse

Temario

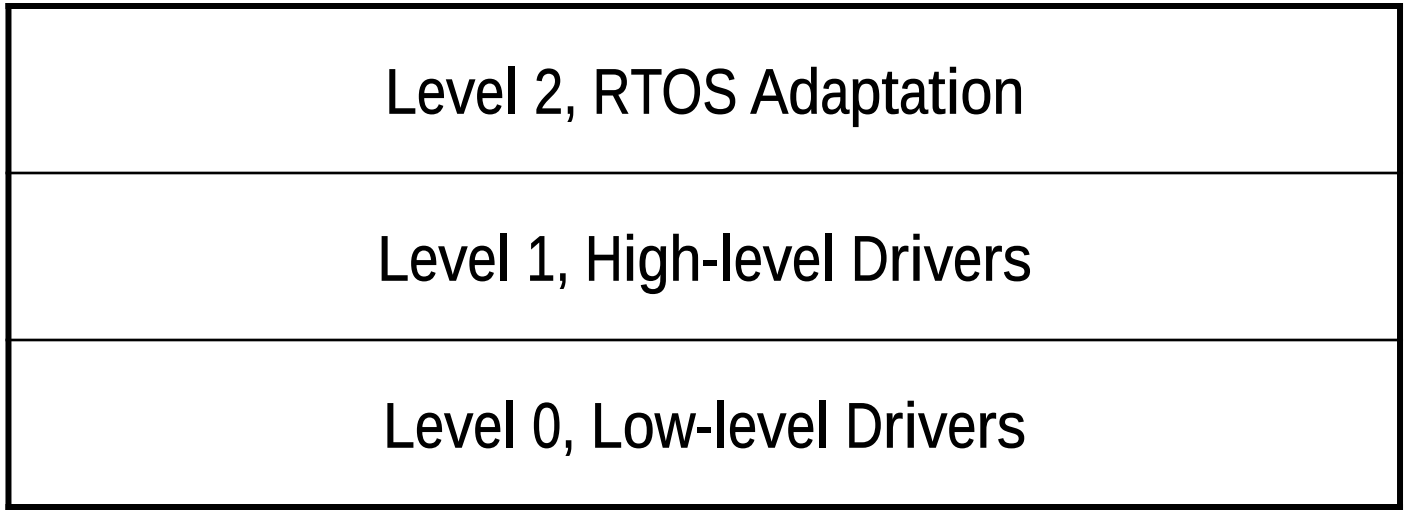
- ***Arquitectura de los Drivers***
- **Timers y API**
- **Herramientas de Depuración**
  - Herramientas de Hardware
  - Herramientas de Software
- **Depuración en SDK**
- **Resumen**

Drivers

- **Los drivers de Xilinx son diseñados para alcanzar los siguientes objetivos:**
  - Proveer máxima portabilidad
    - Los drivers son provistos como código fuente en ANSI C
  - Soportar configuración de FPGA
    - Soporta múltiples instancias del dispositivo sin duplicación de código para cada instancia, mientras que al mismo tiempo permite manejar características únicas por instancia
  - Soporta casos de uso simples y complejos
    - Una arquitectura de driver en capa provee:
      - Un driver simple con uso de memoria mínimo
      - Drivers con características full con uso de memoria más grande
  - Fácil de usar y de mantener
    - Xilinx usa codificación estándard y provee códigos fuente bien documentados para los desarrolladores

Drivers: Level 0/Level 1

- **La arquitectura en capas proporciona una integración perfecta con …**
  - (Level 2) La capa de aplicación RTOS
  - (Level 1) Drivers de Alto-nivel que tienen todas las características y son portables a través de sistemas operativos y procesadores
  - (Level 0) Drivers de Bajo-nivel para caso de uso simples



Drivers: Level 0

- **Consiste de drivers de bajo-nivel**
- **Implementados como macros y funciones que son destinados para que el desarrollador pueda crear un sistema pequeño**
- **Características:**
  - Uso de memoria pequeño
  - Se realiza muy poco o ningún chequeo de error
  - No soporta parámetros de configuración de dispositivo
  - Soporta múltiples instancias de un dispositivo con dirección base de entrada a la API
  - Sólo E/S con polling
  - Llamadas a función bloqueantes

Drivers: Level 1

- **Consiste de drivers de alto-nivel**
- **Implementados como macros y funciones y destinados para que el desarrollador pueda utilizar todas las características de un dispositivo**
- **Características:**
  - API abstracta que aisla la API de los cambios del hardware del dispositivo
  - Soporta parámetros de configuración de dispositivo
  - Soporta múltiples instancias de un dispositivo
  - E/S por polling e interrupción
  - Llamadas a función no-bloqueantes para asistir aplicaciones complejas
  - Puede tener un uso de memoria grande

Ejemplo de comparación

UARTPS Nivel 1

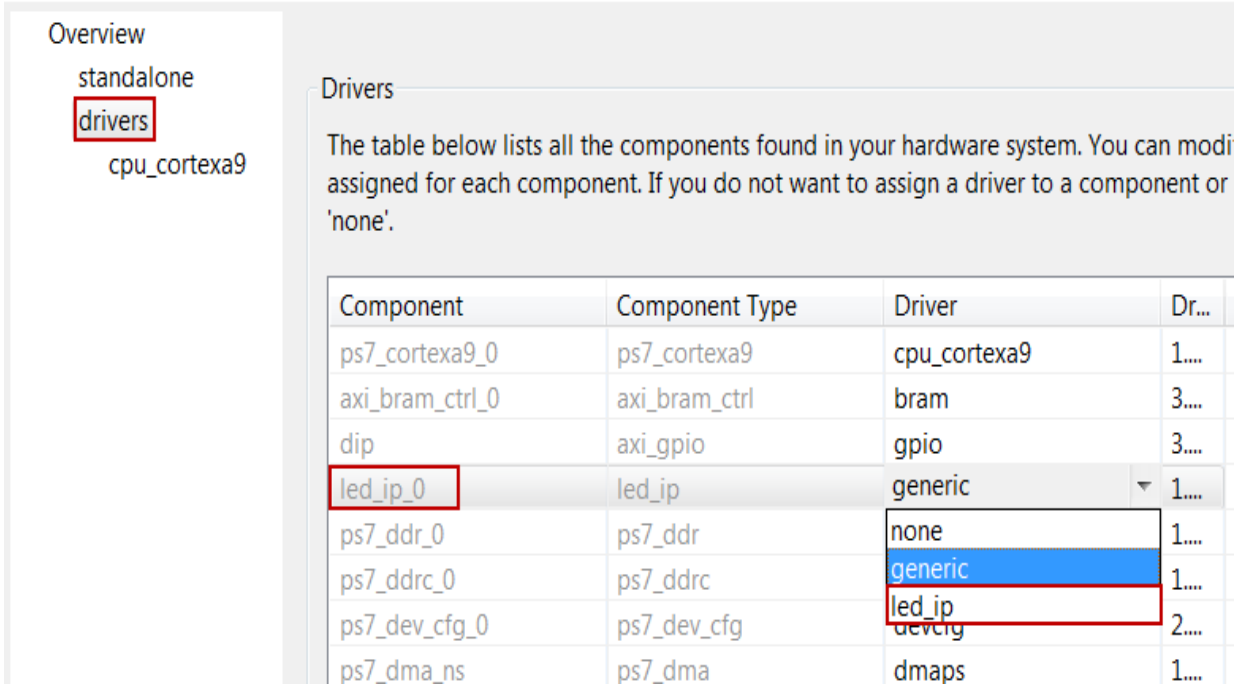
- XUartPs_CfgInitialize() - Inicializa una instancia específica de XUartPs de tal manera de quedar lista para ser usada
- XUartPs_Send() - Envía el buffer especificado usando el dispositivo en modo polling o por interrrupción.
- XUartPs_Recv() - Recibe un número especificado de bytes de datos del dispositivo y los almacena dentro del buffer especificado.
- XUartPs_SetBaudRate() - Establece el baud rate para un dispositivo.

UARTPS Nivel 0

- XUartPs_SendByte() - Envía un byte usando el dispositivo.
- XUartPs_RecvByte() - Recibe un byte del dispositivo.

Configuración de los drivers

- **Selección del panel de Drivers**
- **Por defecto, el panel de drivers muestra qué driver es usado para cada instancia de hardware en el diseño**
- **Habilita la selección de drivers personalizados y versiones para cada dispositivo en el diseño**



Temario

- Arquitectura de los Drivers
- ***Timers y API***
- Herramientas de Depuración
  - Herramientas de Hardware
  - Herramientas de Software
- Depuración en SDK
- Resumen

Timers: Procesador Cortex-A9

- Los Timers son una parte importante de un sistema embebido
- CPU Timer y Watchdog Timer
- Global timer (GTC)
- Dos 16-bit triple timer counter (TTC)
- System watchdog timer (SWDT)

Timer/Counter

- El timer es un contador descendente de 32-bit
- Archivos de encabezado xscutimer.h, xscutimer_hw.h
- XScuTimer_LookupConfig() - Busqueda de la configuración del dispositivo basado en el ID único del dispositivo
- XScuTimer_CfgInitialize() - Inicializa una instancia específica XTtcPs de tal manera que el driver está listo para usarse
- XScuTimer_Start() – Inicia el timer
- XScuTimer_Stop() – Detiene el timer
- XScuTimer_GetPrescaler() – Toma el valor del pre-escalador
- XscuTimer_SetPrescaler() – Fija el valor del pre-escalador entre 1 y 16

Timer/Counter

- XScuTimer_EnableAutoReload() – Carga el contador con el valor inicial cuando ocurre un time out
- XScuTimer_IsExpired() – Verifica si el timer ha alcanzado el valor final
- XScuTimer_RestartTimer() – Lee el valor del contador y lo escribe de vuelta
- XScuTimer_LoadTimer() – Carga el timer con el valor provisto
- XScuTimer_GetCounterValue() – Toma el valor actual del contador; útil para determinar períodos de tiempo
- XScuTimer_EnableInterrupt() – Habilita el mecanismo de interrupción
- XScuTimer_GetInterruptStatus() – Toma el estado de la interrupción
- XScuTimer_ClearInterruptStatus() – Limpia el flag de interrupción

Timers: Triple Timer Counter API

- XTtcPs_LookupConfig() - Buscar la configuración de un dispositivo basado en su ID
- XTtcPs_CfgInitialize() - Inicializar una instancia específica de XTtcPs de tal manera que el driver está listo para ser usado
- XTtcPs_SetMatchValue() - Fijar el valor de los registros que matchean
- XTtcPs_SetOptions() - Fijar las opciones para el dispositivo TTC
- XTtcPs_SetPreScalar() - Fijar el bit de habilitación del prescalador
- XTtcPs_GetMatchValue() - Tomar el valor de los registros matcheados
- XTtcPs_GetOptions() - Tomar la configuración para las opciones para el dispositivo TTC

AXI Timer

- XTmrCtr_Initialize() - Inicializar una instancia/driver específica de un timer/counter
- XTmrCtr_InterruptHandler() - Interrupt Service Routine (ISR) para el driver
- XTmrCtr_SetHandler() - Establecer la función callback del timer, la cual el driver llama cuando el timer especificado llega al final de la cuenta (times out)
- XTmrCtr_GetOptions() - Habilita las opciones específicas para el timer counter especificado
- XTmrCtr_Start() - Inicia el timer counter especificado del dispositivo de tal manera que comienza a correr
- XTmrCtr_Stop() - Para el timer counter deshabilitándolo
- XTmrCtr_GetCaptureValue() - Retorna el valor del timer counter que fue capturado la última vez que la entrada de captura externa fue puesta a 1

AXI Timer

- XTmrCtr_GetOptions() - Tomar las opciones para el timer counter especificado
- XTmrCtr_GetStats() - Tomar una copia de la estructura XtmrCtrStats, la cual contiene las estadísticas actuales para este driver
- XTmrCtr_Getvalue() - Tomar el valor actual para el timer counter
- XTmrCtr_Reset() - Resetear el timer counter especificado del dispositivo

Temario

- Arquitectura de los Drivers
- Timers y API
- ***Herramientas de Depuración***
  - *Herramientas de Hardware*
  - *Herramientas de Software*
- Depuración en SDK
- Resumen

Depuración

- **La depuración es una parte integral del desarrollo de sistemas embebidos**
- **El proceso de depuración está definido como testing, stabilizing, localizing, y correción de errores**
- **Dos métodos de depuración:**
  - Depuración de hardware via una punta lógica, analizador lógico, o un emulador in-circuit
  - Depuración de software a través de un instrumento de depuración
    - Un instrumento de depuración de software es un código fuente que es agregado al programa con el propósito de depuración
- **Tipos de Depuración:**
  - Depuración funcional
  - Depuración de performance

Soporte para Depuración de Software

- **Vivado/SDK soporta depuración de software a través de:**
  - Herramientas GDB
    - Interfaz gráfica unificada para depuración y verificación de sistemas de procesamiento
  - Xilinx Microprocessor Debugger (XMD)
    - Corre todas las herramientas de depuración y se comunica con el hardware
    - Shell para la comunicación con el hardware
    - Sintáxis Tcl (Tool command language) y un intérprete de comandos
  - Las herramientas GNU se comunican con el hardware a través de XMD
  - Xilinx System Debugger, Eclipse Target Communications Framework (TCF)

Soporte para Depuración de Hardware

- **Vivado soporta depuración de hardware via las siguientes herramientas**
  - El software Logic Analyzer
    - Soft-core base logic analyzer
    - Opera a través de un cable de descarga Xilinx
- **Zynq™ AP SoC virtual platform**
  - Simulación funcional del hardware físico con el propósito de desarrollo de software, integración, y test
  - Corre en el desktop
  - Facilita el desarrollo temprano de software y la prueba
- **Zynq AP SoC open-source QEMU model**
  - Emulador de máquina Open-source y virtualizador para ambiente Linux

Depurador XMD

- **La utilidad Xilinx Microprocessor Debug (XMD) provee una variedad de servicios de depuración de usuario**
  - Conexión física entre su workstation y el diseño de software
  - Conexión a un controlador BSCAN interno
  - Descarga del programa
  - Identificación y control de procesador
  - Comandos de depuración de bajo nivel
  - Interfaz hacia el depurador GNU
  - Interaz Tcl e intérprete de comandos

Depurador XMD

- Para depuración standalone o aplicaciones bare-metal, funciona como gdbserver para gdb y SDK
- Para aplicaciones Linux, el SDK interacúa con un gdbserver corriendo en el target
- XMD es iniciado via el comando en SDK Xilinx > Program FPGA

Funcionalidad XMD

- **Motor XMD**
  - Programa que facilita una interfaz GDB unificada
  - Interfaz Tcl e intérprete de comandos
- **XMD soporta depuración de aplicaciones en diferentes targets**
- **GDB/TCF pueden conectarse al XMD en la misma computadora o en una remota sobre internet**

Comandos XMD

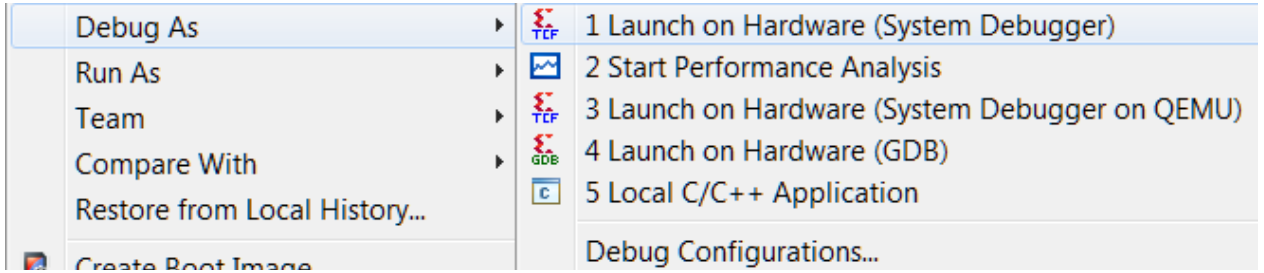
- **Existen muchos comandos XMD**
- **Comandos populares para bootear y controlar el programa**
  - connect – conectar al procesador
  - dow – descargar el archivo ejecutable ELF
  - elf_verify – verificar el archivo ELF con la imagen en memoria
  - run – iniciar la ejecución del programa desde el reset
  - con – continuar la ejecución del programa desde el program counter actual
  - stop – detener el procesador target
  - exit – cerrar la ventana XMD
- **XMD buscará un procesador cuando sea iniciado y lanzado desde el SDK Run > Debug menu**
- **El comando connect se ejecutará automáticamente**
  - connect arm 64

XMD: Interfaz Tcl

- **xhelp: Lista todos los comandos Tcl**
- **xrmem target addr [num]: Lee el número de bytes o 1 byte de la dirección *addr* de la memoria**
- **xwmem target addr value: Escribe un byte en la dirección *addr* de memoria**
- **xrreg target [reg]: Lee todos los registros o sólo el registro número *reg***
- **xwreg target reg value: Escribe un valor de 32-bit en el registro número *reg***
- **xdownload target [-data] filename [addr]: Descarga el ELF dado o el archivo de datos (with -data option) sobre la memoria del target actual**
- **xcontinue target [addr]: Continúa la ejecución desde la ubicación actual del PC o desde el argumento opcional *addr***

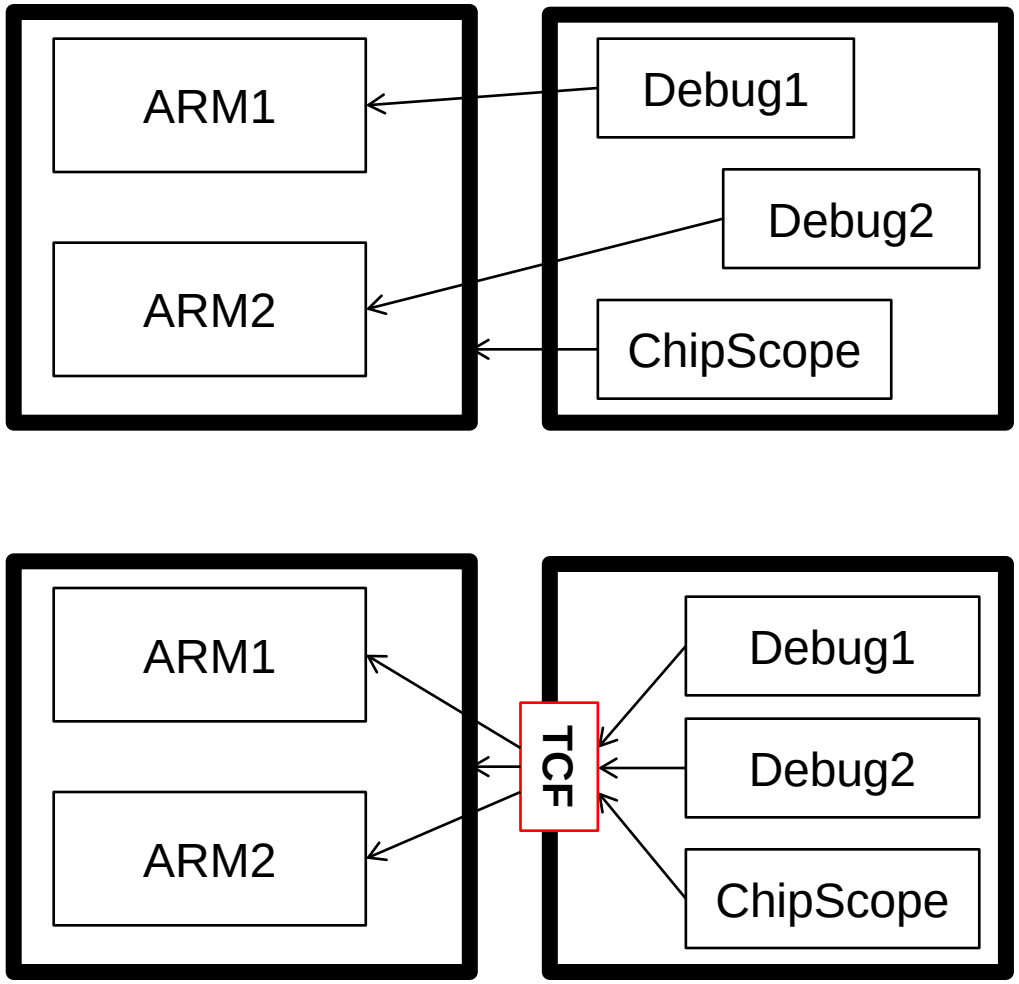
Depurador del Sistema

- **Eclipse Target Communication Framework**
  - Protocolo de red extensible para comunicarse con sistemas embebidos
- **Configuración simple por target**
  - (No por herramienta como el gdb)
- **Homogenous, and heterogeneous, SMP and AMP support**
- **Soporte de Neon**
- **Depuración multicore verdadera a través de JTAG**
- **Más rápido que GDB/XMD**



Eclipse Target Communication Framework

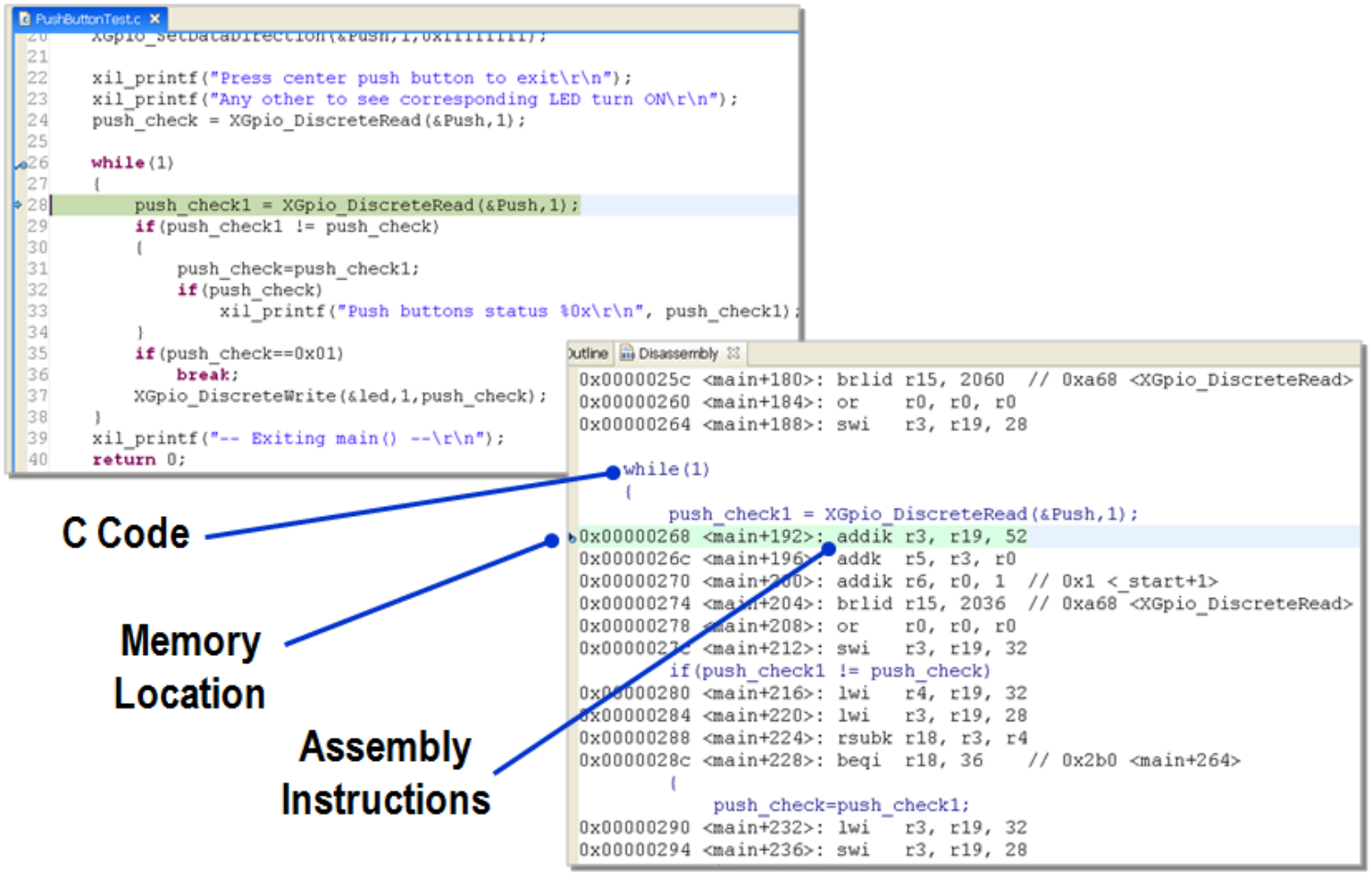
- **Protocolo de red extensible y abierto**
- **Permite servicios para conectarse de manera transparente**
- **Todos los links de comunicación pueden compartir el mismo protocolo**
- **Abstracción del canal de transporte**
  - (Capa de transporte no específica. Por ejemplo, TCP/IP, Serial Line, SSH tunnel)



Funcionalidad de Depuración

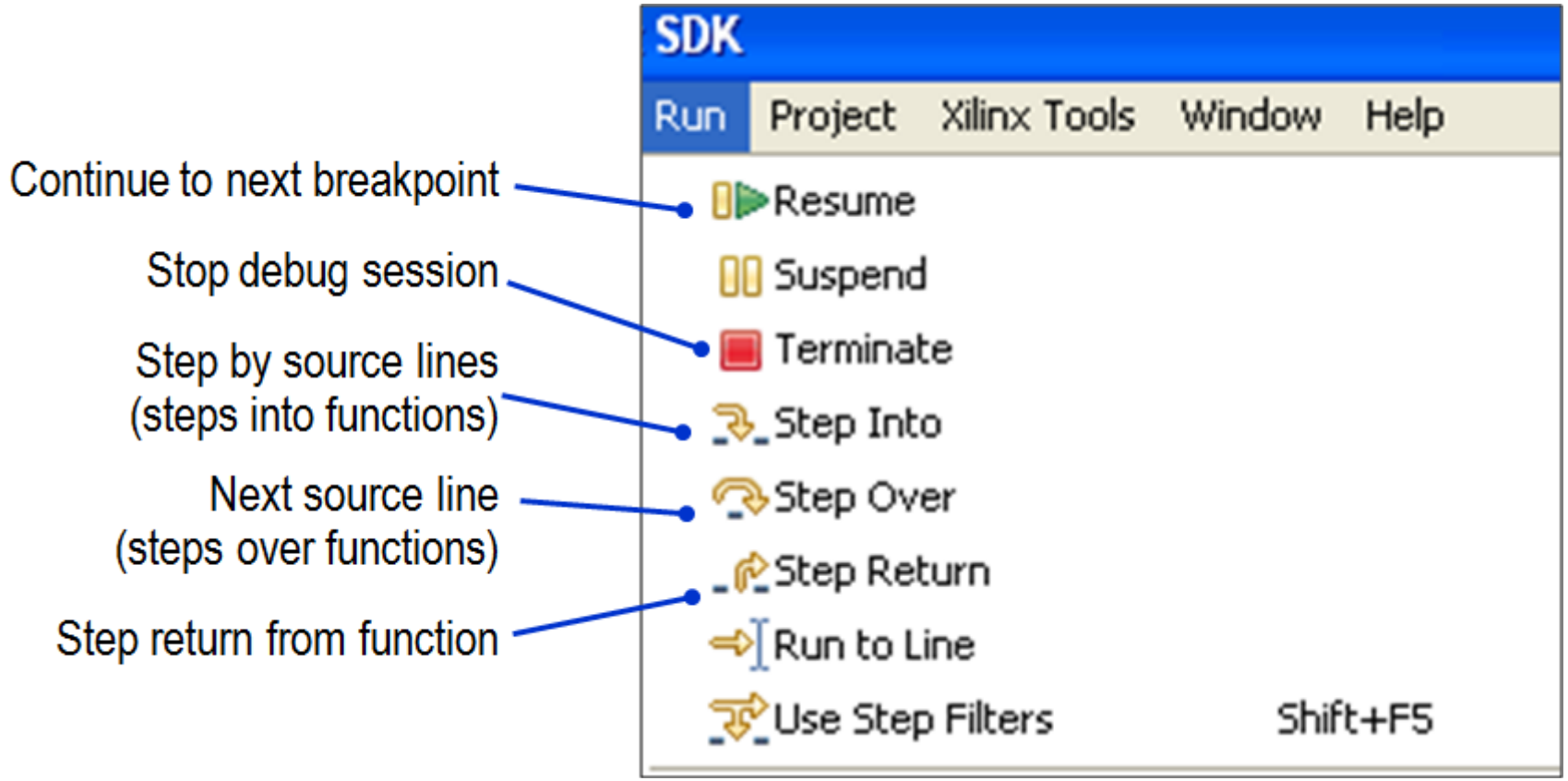
- **Source-level debugger (GDB/System Debugger)**
- **Iniciar su programa**
  - Establecer breakpoints (hacer que el programa se detenga bajo condiciones específicas)
  - Examinar qué ha pasado cuando su programa encuentra un breakpoint
    - Registros
    - Memoria
    - Stack
    - Variables
    - Expresiones
  - Cambiar algo en su programa de tal manera que pueda experimentar los efectos del bug corregido y continuar en busca de otro
- **Puede depurar programas escritos en C y C++**

Depuración


PushButtonTest.c
21
22 xil_printf("Press center push button to exit\r\n");
23 xil_printf("Any other to see corresponding LED turn ON\r\n");
24 push_check = XGpio_DiscreteRead(&Push,1);
25
26 while(1)
27 {
28 push_check1 = XGpio_DiscreteRead(&Push,1);
29 if(push_check1 != push_check)
30 {
31 push_check=push_check1;
32 if(push_check)
33 xil_printf("Push buttons status %0x\r\n", push_check1);
34 }
35 if(push_check==0x01)
36 break;
37 XGpio_DiscreteWrite(&led,1,push_check);
38 }
39 xil_printf("-- Exiting main() --\r\n");
40 return 0;
Outline
Disassembly
0x0000025c <main+180>: brlid r15, 2060 // 0xa68 <XGpio_DiscreteRead>
0x00000260 <main+184>: or r0, r0, r0
0x00000264 <main+188>: swi r3, r19, 28
while(1)
{
push_check1 = XGpio_DiscreteRead(&Push,1);
0x00000268 <main+192>: addik r3, r19, 52
0x0000026c <main+196>: addk r5, r3, r0
0x00000270 <main+200>: addik r6, r0, 1 // 0x1 <_start+1>
0x00000274 <main+204>: brlid r15, 2036 // 0xa68 <XGpio_DiscreteRead>
0x00000278 <main+208>: or r0, r0, r0
0x0000027c <main+212>: swi r3, r19, 32
if(push_check1 != push_check)
0x00000280 <main+216>: lwi r4, r19, 32
0x00000284 <main+220>: lwi r3, r19, 28
0x00000288 <main+224>: rsubk r18, r3, r4
0x0000028c <main+228>: beqi r18, 36 // 0x2b0 <main+264>
{
push_check=push_check1;
0x00000290 <main+232>: lwi r3, r19, 32
0x00000294 <main+236>: swi r3, r19, 28
C Code
Memory Location
Assembly Instructions

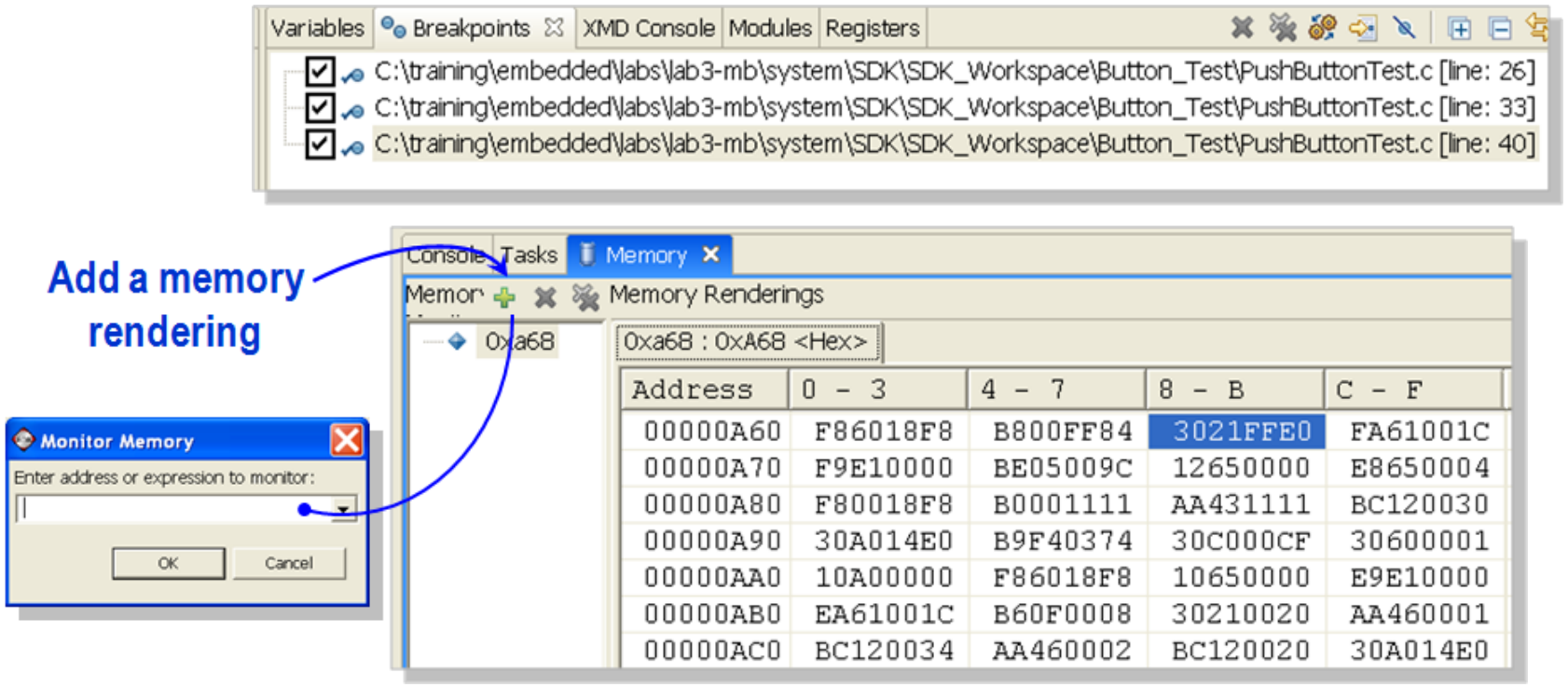
GUI de Depuración

- **Control en tiempo de ejecución (Run-time)**



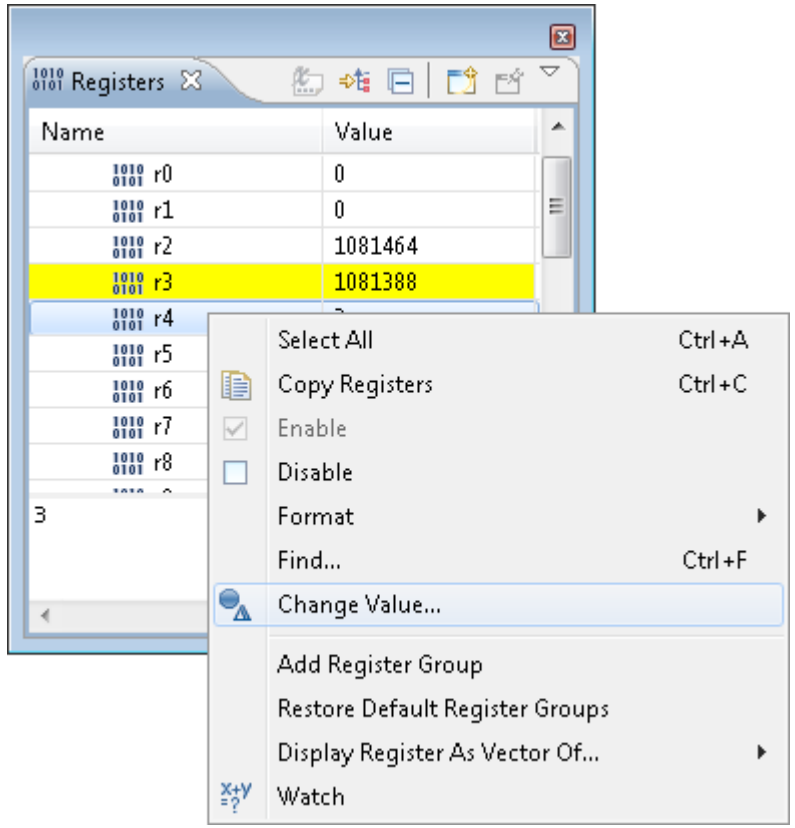
Funcionalidad de Depuración

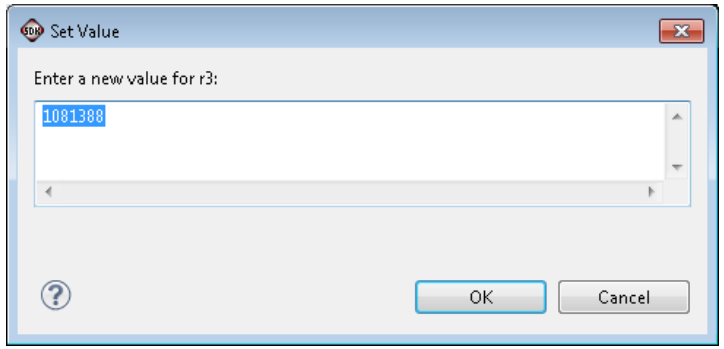
- Los breakpoints pueden ser habilitados o inhabilitados
- Para cambiar una posición de memoria cualquiera, hacer click en un campo de memoria

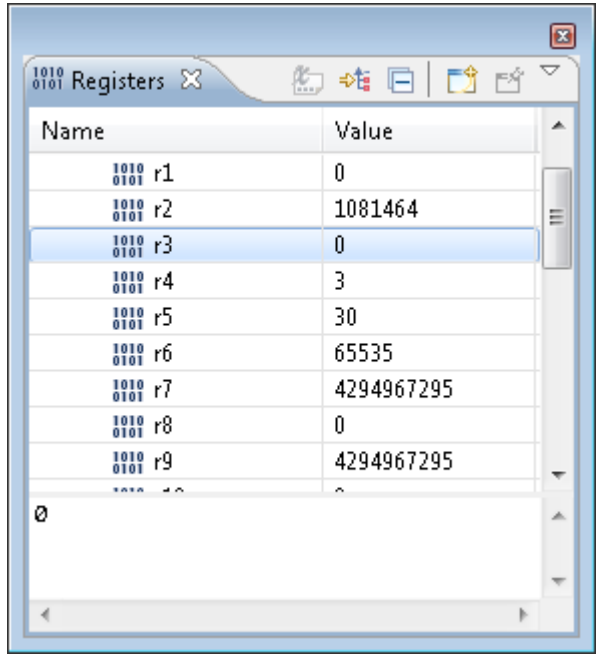


Funcionalidad de Depuración

- El color amarillo en el registro indica que ha cambiado su valor (útil cuando se está haciendo seguimiento de código ensamblado)
- Para cambiar cualquier valor, hacer click para editar, o click-derecho en el campo



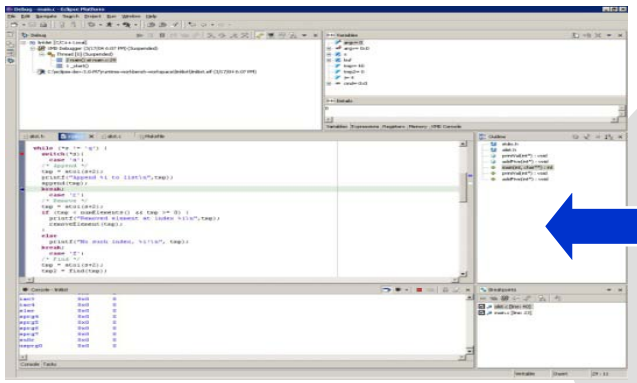




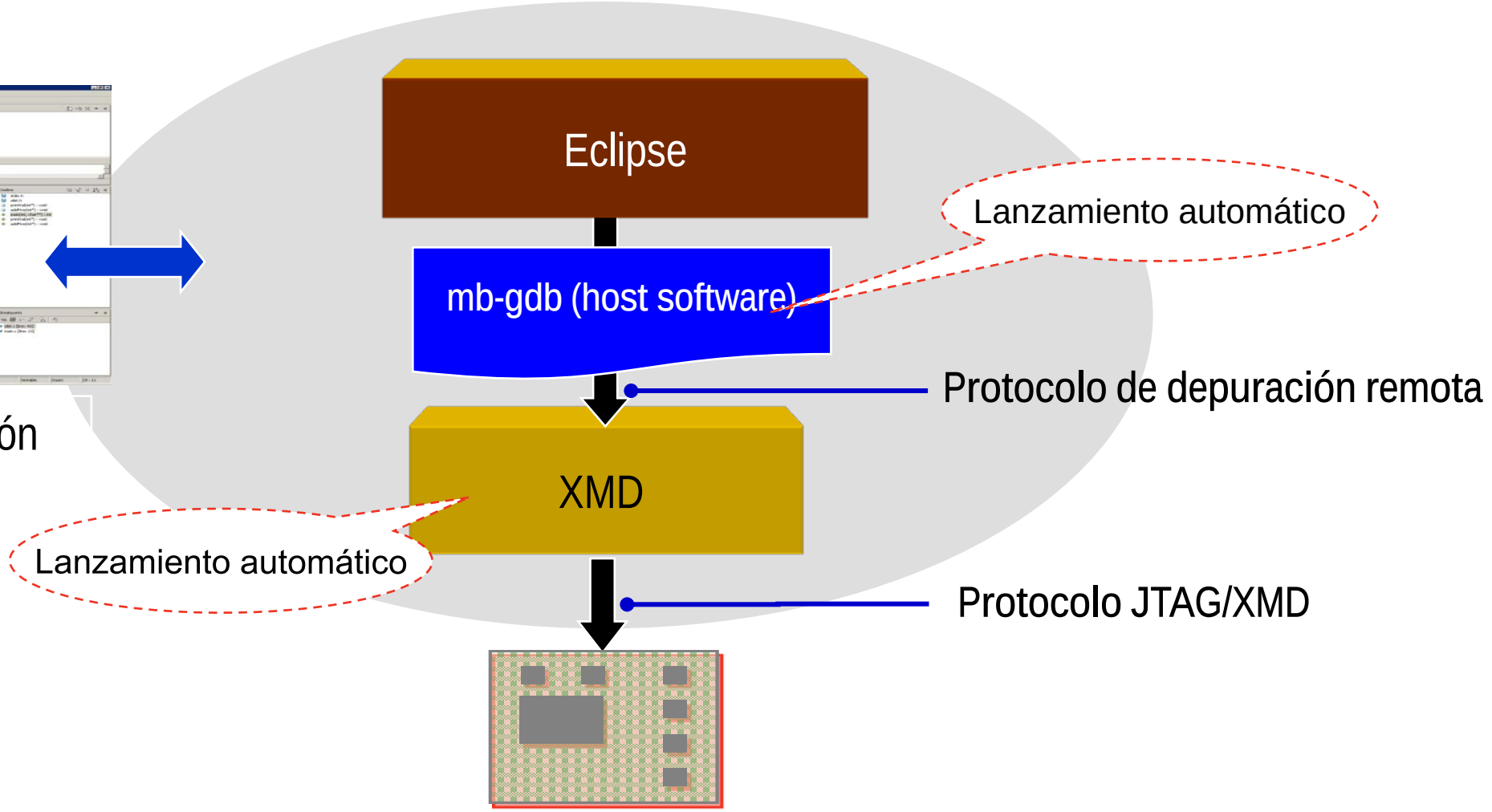
Temario

- Arquitectura de los Drivers
- Timers y API
- Herramientas de Depuración
  - Herramientas de Hardware
  - Herramientas de Software
- ***Depuración en SDK***
- Resumen

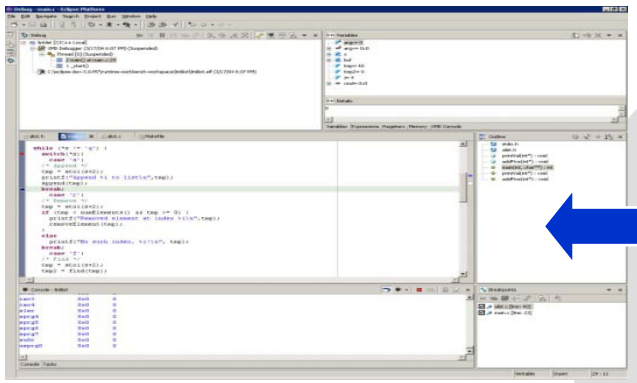
Depuración usando SDK (XMD)



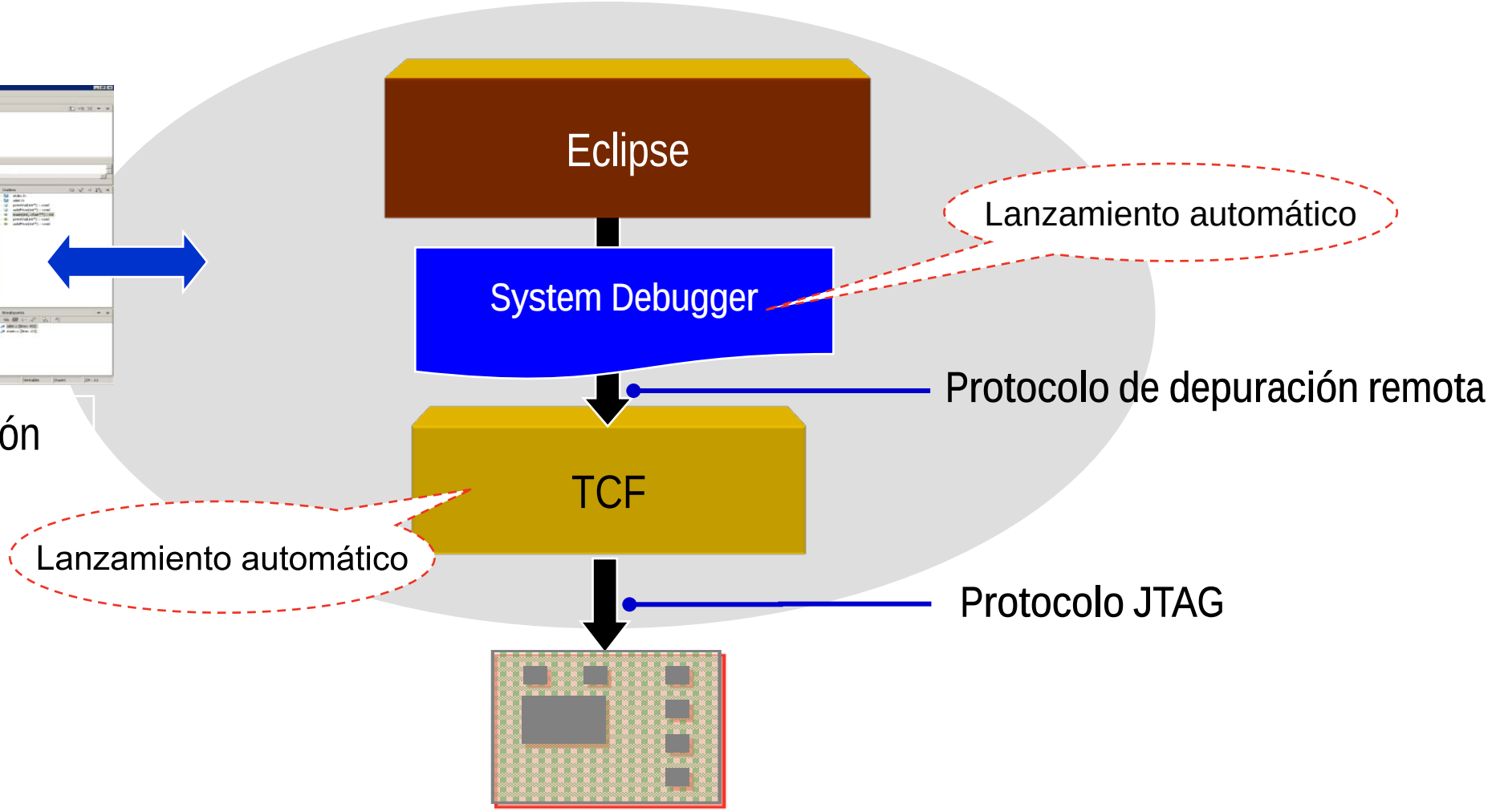
Interfaz de depuración gráfica de Xilinx



Depuración usando SDK (TCF)

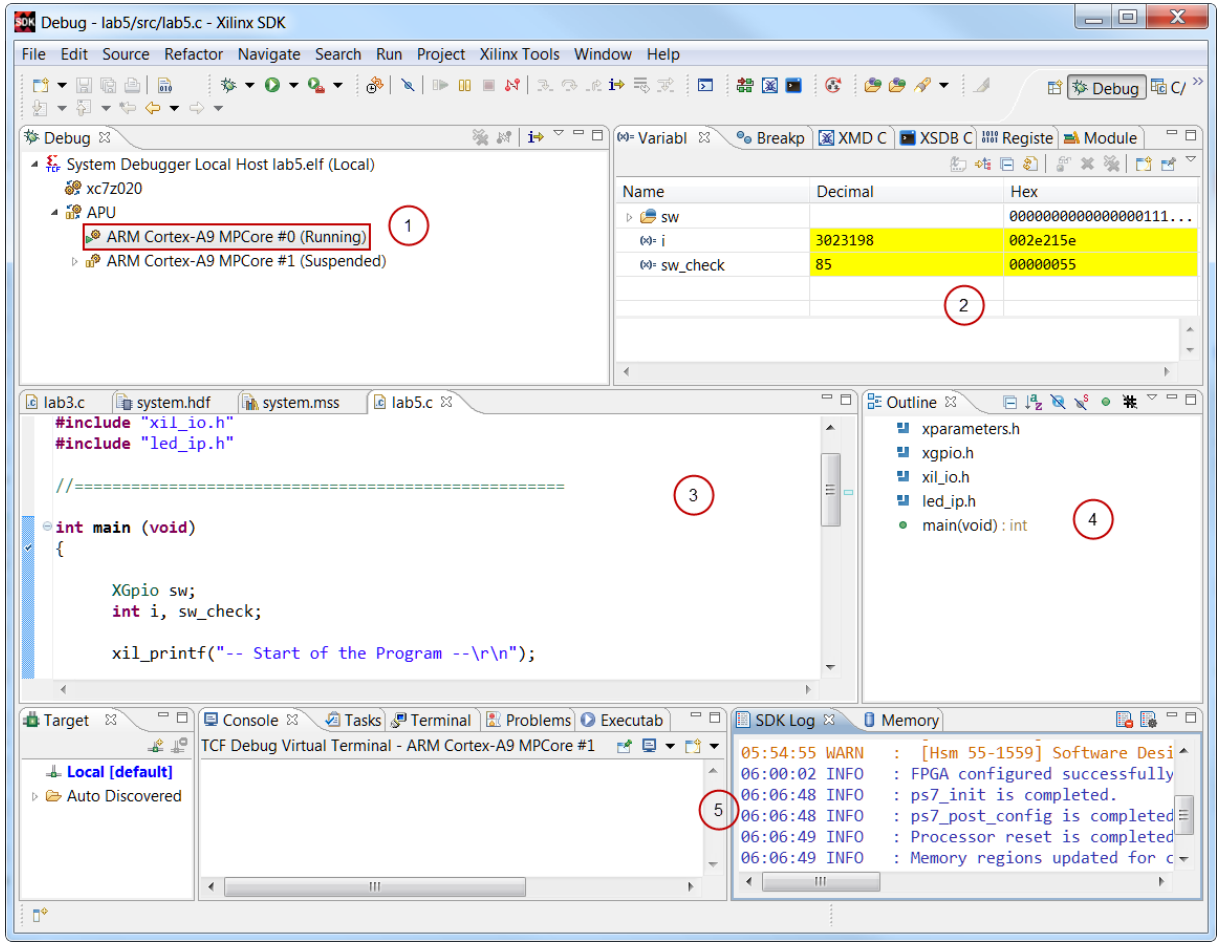


Interfaz de depuración gráfica de Xilinx



Perspectiva Debug del SDK

- **Vistas de variables, breakpoints, y registros**
- **Editor C/C++**
- **Estructura del código (outline)**
  - Puede agregarse una vista de desensamblado usando Window > Show View > Disassembly
- **Vistas de consola, log, y memoria**



Temario

- Arquitectura de los Drivers
- Timers y API
- Herramientas de Depuración
  - Herramientas de Hardware
  - Herramientas de Software
- Depuración en SDK
- ***Resumen***

Resumen

- **La depuración es una parte integral del desarrollo de sistemas embebidos**
- **Vivado provee herramientas para facilitar la depuración de hardware y software**
  - La depuración de hardware se realiza usando Vivado Logic Analyzer
  - La depuración de software se realiza usando xmd y el depurador GNU
- **El SDK provee un entorno, una perspectiva, y herramientas subyacentes para habilitar de una manera transparente la depuración de software**
- **Depuración XMD/GDB**
  - Dispositivo simple sobre una conexión dedicada
- **System Debugger/TCF**
  - Depuración Multicore, conexión compartida