NIKTECH INC

Developers' Guide

Developers' Guide

190 Shooting Star Isle
Foster City, CA - 94404

Table of Contents

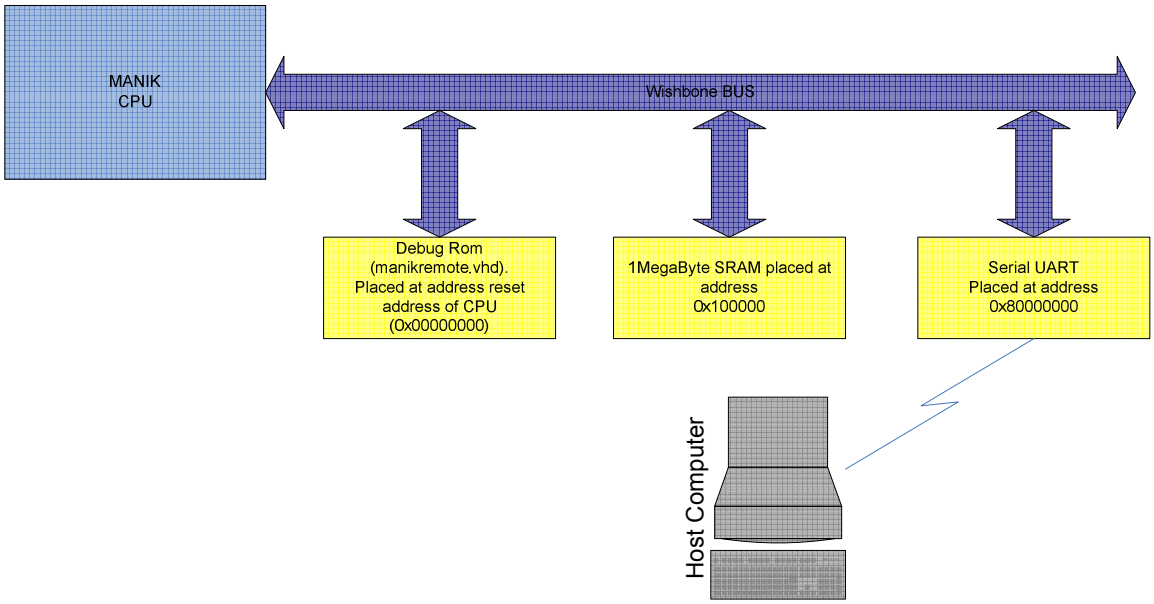
Introduction

This document describes hardware and software development using MANIK – 32bit RISC CPU. The hardware portion will describe the process to build SoCs and changing the Configuration parameters. The software development will deal with creating and compiling programs, loading the program into the memory of the target system and debugging the program.

A Typical SoC

This chapter describes a typical SoC. The system includes the CPU, the Debug Monitor and an external ASRAM memory .

Block Diagram



CPU Startup

On Power-up or Reset the CPU will start executing code from the location pointed to by the Reset Vector. The Reset Vector is located at address 0x000000000 by default, but can be changed at system generation by the CPU Configuration parameter INTR_VECBASE.

In a typical case the debug ROM is placed at the system reset location and starts executing. The code in the debug ROM (***$(MANIK_BASE)/manikremote/manikremote.c***) will wait for commands from the Host debugger (via the Serial Port). The debug ROM code is relocatable (i.e. if the reset address is changed the same debug ROM without modifications).

Debug ROM Monitor

The addition to interacting with the host based debugger , the ROM Monitor also provides some basic interrupt handling services to the application program. The Monitor program can also send the output to the user console on the host using the debugger.

The Debug ROM Monitor uses a special protocol to communicate with the host based debugger, via a serial port. The system is **required** to have a **serial** port. The Debug ROM Monitor will need to be recompiled if the base address of the serial port is modified, the code in the debug ROM is relocatable (i.e. if the reset address is changed the same debug ROM without modifications).The default address for the serial port is 0x80000000.

Interrupt service

When the Debug ROM Monitor is included the interrupts are generally handled by the debug ROM Monitor program. The application can register a interrupt service routine function, this function will be called by the debug ROM Monitor when the corresponding interrupt is detected.

```
void register_isr(int type, void(*isr)(int *), int xnum);
```

type – can be either **TIMER_IRQ** (to register to for the Timer Interrupt) or **EXTRN_IRQ** (to register for one of the external interrupts).

isr – pointer to the interrupt service routine.

xnum – The number of the external interrupt 0-5 (Required only when registering for an external interrupt type = **EXTRN_IRQ**).

To **un-register** for an interrupt call the same function with a NULL pointer for the **ISR** parameter.

The routine registers a function to be called by the Interrupt service routine, it **does not** enable the interrupt. The registered interrupt service routine is called with a pointer to the register save area, the application program can manipulate the value of the registers. The register order is enumerated in **manik_system.h**. The changes made to the register will take effect when the ISR finishes executing.

The ISR is called when the processor is in interrupt processing state (IP flag in the PSW is 1), the routine can generate **software interrupts** ,either by executing the **swint** instruction or by calling system functions such **printf**. .

Compiling programs

The MANIK tool chain uses the GNU C Compiler (GCC) for compilation. Both C & C++ languages are supported.

Compiling for Instruction set simulator

The default configuration of the compiler is to compile for the instruction set simulator.

manik-elf-gcc –o banner.elf.iss banner.c

Will compile a program called ***banner.c*** and create an executable ***banner.elf.iss***. To execute the program using the ISS use the following command.

manik-elf-run banner.elf.iss

Note hardware breakpoints and watch points are not supported by the ISS.

Compiling with Debug ROM Monitor

When a program is compiled for the Debug ROM Monitor, some of the functions are performed by the Debug ROM Monitor, these include system startup, register save/restore for interrupt handling, communication with the debugger and output to stdout (via the debugger).

```
manik-elf-gcc -custom-crt banner.c  -T
$(MANIK_BASE)/c_system_lib/$(BOARD)/board_linker_script -o
banner.elf
```

The –T option specifies the linker script to be used, the linker script specifies the memory map for the SoC being used and is board specific. The option ***–custom-crt*** is used to tell the compiler to use an alternate startup file **cstmcrt0.s**.

Compiling a Standalone application

The Reset vector of the CPU must contains some valid code, typically the Debug ROM Monitor is placed at this address; a Flash Memory or initialized FPGA Block RAM can also be placed at this address. In such a case the User application is responsible of system startup and interrupts handling.

A typical example of such an application is the Debug ROM Monitor itself. The **Makefile** in the directory ***$(MANIK_BASE)/manikremote*** can be used as an example. The Debug ROM Monitor is assumed to be located at address 0.

Note that if the application used I/O functions such as **printf**; the application should contain a function ***write*** which will send the characters to the device designated as ***stdout***.

Using the Built-in Timer

The MANIK CPU core has a built-in timer module, the TIMER can be used in **count down mode,** or in **interrupt mode.**

In the count down mode, the timer is preloaded with a value and started, the counter will be decrement every TIMER_CLKDIV cycles , (default every clock cycle) and stop at zero, the TU flag (PSW bit 12) will be set if an underflow occurs.

In the interrupt mode the timer is loaded and is decremented like the count down mode, an interrupt is generated when the timer reaches zero.

Some software support is provided to facilitate the interrupt based timer service. The software supports uses the stand C library to implement the interface to the timer. To use the service the application program must first register a **signal handler** for **SIGALRM,** this is done by calling the standard C library function **signal.**

```
#include <signal.h>

typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);
```

The signal() system call installs a new signal handler for the signal with number ***signum***. The signal handler is set to ***sighandler*** which may be a user specified function, or either SIG_IGN or SIG_DFL.

Upon arrival of a signal with number ***signum*** the following happens. If the corresponding handler is set to SIG_IGN, then the signal is ignored. If the handler is set to SIG_DFL, then the default action associated to the signal occurs. Finally, if the handler is set to a function ***sighandler*** then first either the handler is reset to SIG_DFL or an implementation-dependent blocking of the signal is performed and next ***sighandler*** is called with argument ***signum***.

The application must then arm the timer , this is done by calling the standard C library function **setitimer.**

```
      #include <sys/time.h>

int setitimer(int  which,  const  struct  itimerval  *value,
     struct itimerval *ovalue);
```

The setitimer() function call sets the value of the timer specified by ***which*** to the value specified in the structure pointed to by ***value***, and if ***ovalue*** is not ***NULL***, stores the previous value of the timer in the structure pointed to by ***ovalue***.

The **which** parameter can have the value ITIMER_REAL, this decrements in real time. A SIGALRM signal is delivered when this timer expires.

The function **signal** is implemented as part of the standard C library, the function **setitimer** is dependent on the system frequency and is built as part of the system specific library found in the file ***$(MANIK_BASE)/c_system_lib/timer.c***.

Using the debugger

The MANIK tool chain includes the GNU Debugger (GDB) ported for MANIK.

Download & Execute

This section provides a brief overview of using GDB (manik-elf-gdb) from the command line, to connect to the target Debug ROM Monitor , download and execute/debug programs.

- Use vendor specific tool to configure the FPGA using the generated **bitstream**.
- Start MANIK debugger

  manik-elf-gdb $(MANIK_BASE)/examples/banner

- Connect to the target board via a serial port

  (gdb) target manikrem /dev/com4 115200

  /dev/com4 is the serial port on the PC to which FPGA serial port is connected.

115200 is the baud rate for the serial port (defined in **manikconfig.vhd(CONFG_BAUD_RATE))**

The response from the debugger should be

Remote manikrem connected to /dev/com4

- Load the program into the target memory. The load location is defined in the linker script.

```
(gdb) load
```

The response should be

Loading section ram_section, size 0xc4b4 lma 0x4000000

Start address 0x4000000, load size 50356

Transfer rate: 80569 bits/sec, 508 bytes/write.

- To execute the program

```
(gdb) continue
```

Setting Break Points

Software breakpoints can be set with the **break** command in gdb

```
(gdb) break banner

Breakpoint 1 at 0x40000d8: file banner.c, line 19.
```

To set a **hardware** break point use the command **hbreak ,** with the same parameters. Note the CPU must be configured with the **HW_BPENB** option. The hardware breakpoint interrupt is generated **before** the instruction at the address specified executes.**Two** hardware breakpoints can be set simultaneously. Hardware break points set in an **interrupt service routine** cannot be cleared without a cpu reset.

To set a **hardware watch point**, use the command **watch <expression>**. The expression generally contains a (global) variable name , or an address expression; e.g. ***watch some_array[10]*** this watch point will stop execution whenever the value of ***some_array[10]*** changes.