openMSP430

an MSP430 clone....

Author: Olivier GIRARD

olgirard@gmail.com

Rev. 1.2

December 27, 2009

Revision History

Rev.	Date	Author	Description
1.0	August 4th, 2009	GIRARD	First version.
1.1	August 30th, 2009	GIRARD	Replaced “openMSP430.inc“ with “openMSP430_defines.v“
1.2	December 27th, 2009	GIRARD	- Update file and directory description for hte FPGA projects (in particular, add the Altera project). - Diverse minor updates.

Contents

1. Overview

Introduction

The openMSP430 is a synthesizable 16bit microcontroller core written in Verilog. It is compatible with Texas Instruments' MSP430 microcontroller family and can execute the code generated by an MSP430 toolchain in a cycle accurate way.

The core comes with some peripherals (GPIO, Timer A, generic templates) and a Serial Debug Interface for in-system software development.

Download

Click here to download the complete tar archive of the project (OpenCores account required).

Without account, you can also run the following SVN command from a console (or GUI):

```
svn export http://opencores.org/ocsvn/openmsp430/openmsp430/trunk/ openmsp430
```

Features & Limitations

Features

- **Core:**
  - Full instruction set support.
  - All addressing modes are supported.
  - IRQ and NMI support.
  - Power saving modes functionality is supported.
  - Configurable ROM and RAM size.
  - Serial Debug Interface (Nexus class 3).
  - FPGA friendly (single clock domain, no clock gate).
  - Small size (uses ~43% of a XC3S200 Xilinx Spartan-3).

- **Peripherals:**
    - Basic Clock Module.
    - Watchdog.
    - Timer A.
    - GPIO (port 1 to 6).

Limitations

- **Core:**
    - Instructions can't be executed from the data memory.

- **Peripherals:**
    - Basic clock module doesn't offer the full functionality of a real MSP430.

Links

Development has been performed using the following freely available (excellent) tools:

- Icarus Verilog : Verilog simulator.
- GTKWave Analyzer : Waveform viewer.
- MSPGCC : GCC toolchain for the Texas Instruments MSP430 MCUs.
- ISE WebPACK : Xilinx's FPGA synthesis tool.

A few MSP430 links:

- Wikipedia: MSP430
- TI: MSP430x1xx Family User's Guide

Legal information

2. Core

Table of content

1. Introduction

The openMSP430 is a 16-bit microcontroller core compatible with TI's MSP430 family (note that the extended version of the architecture, the MSP430X, isn't supported by this IP). It is based on a Von Neumann architecture, with a single address space for instructions and data.

This design has been implemented to be FPGA friendly. Therefore, the core doesn't contain any clock gate and has only a single clock domain. As a consequence, the clock management block has a few limitations.

This IP doesn't contain the program and data memory blocks internally (these are technology dependent hard macros which are connected to the IP during chip

integration). However the core is fully configurable in regard to the supported RAM and ROM size.

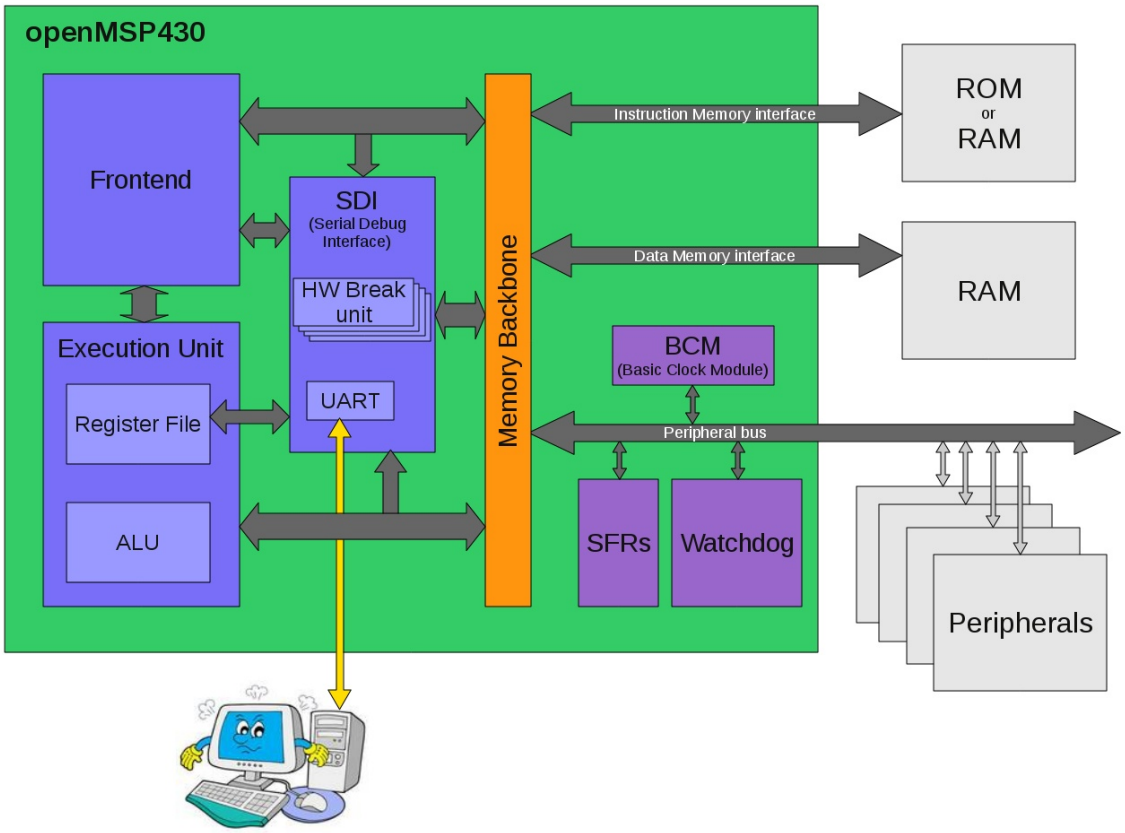
In addition to the CPU core itself, several peripherals are also provided and can be easily connected to the core during integration.

2. Design

2.1 Core

2.1.1 Design structure

The following diagram shows the openMSP430 design structure:



- **Frontend**: This module performs the instruction Fetch and Decode tasks. It also contains the execution state machine.
- **Execution unit**: Containing the ALU and the register file, this module executes the current decoded instruction according to the execution state.
- **Serial Debug Interface**: Contains all the required logic for a Nexus class 3 debugging unit (without trace). Communication with the host is done with a standard 8N1 serial interface.

- **Memory backbone**: This block performs a simple arbitration between the frontend and execution-unit for instruction and data memory access.
- **Basic Clock Module**: Generates the ACLK and SMCLK enable signals.
- **SFRs**: The **S**pecial **F**unction **R**egister**s** block contains diverse configuration registers (NMI, Watchdog, ...).
- **Watchdog**: Although it is a peripheral, the watchdog is permanently included in the core because of its tight links with the NMI interrupts and the PUC reset generation.

2.1.2 Limitations

The known core limitations are the following:

- Instructions can't be executed from the data memory.
- SCG0 is not implemented (turns off DCO).
- MCLK can't be divided and can only have DCO_CLK as source (see Basic Clock Module section).

2.1.3 Configuration

It is possible to configure the openMSP430 core through the "*openMSP430_defines.v*" file located in the *rtl* directory (see file and directory description).

Two parameters can be adjusted by the user in order to define the ROM and RAM sizes:

```
// ROM Size
//              9 -> 1kB
//             10 -> 2kB
//             11 -> 4kB
//             12 -> 8kB
//             13 -> 16kB
`define ROM_AWIDTH 10

// RAM Size
//              6 -> 128 B
//              7 -> 256 B
//              8 -> 512 B
//              9 -> 1 kB
//             10 -> 2 kB
`define RAM_AWIDTH 6
```

The following parameters define if the debug interface should be included or not and how many hardware breakpoint units should be included:

```
//-----------------------------------------------------------------------------
// REMOTE DEBUGGING INTERFACE CONFIGURATION
//-----------------------------------------------------------------------------

// Include Debug interface
`define DBG_EN
```

```
// Debug interface selection
// `define DBG_UART -> Enable UART (8N1) debug interface
// `define DBG_JTAG -> DON'T UNCOMMENT, NOT SUPPORTED YET
//
`define DBG_UART
//`define DBG_JTAG

// Number of hardware breakpoints (each unit contains 2 hw address breakpoints)
// `define DBG_HWBRK_0 -> Include hardware breakpoints unit 0
// `define DBG_HWBRK_1 -> Include hardware breakpoints unit 1
// `define DBG_HWBRK_2 -> Include hardware breakpoints unit 2
// `define DBG_HWBRK_3 -> Include hardware breakpoints unit 3
//
`define DBG_HWBRK_0
`define DBG_HWBRK_1
`define DBG_HWBRK_2
`define DBG_HWBRK_3
```

All remaining defines located in this file are system constants and should not be edited.

2.1.4 Pinout

The full pinout of the openMSP430 core is provided in the following table:

Port Name	Direction	Width	Description
Clocks			
dco_clk	Input	1	Fast oscillator (fast clock), CPU clock
lfxt_clk	Input	1	Low frequency oscillator (typ. 32kHz)
mclk	Output	1	Main system clock
aclk_en	Output	1	ACLK enable
smclk_en	Output	1	SMCLK enable
Resets			
puc	Output	1	Main system reset
reset_n	Input	1	Reset Pin (low active)
Interrupts			
irq	Input	14	Maskable interrupts (one-hot signal)
nmi	Input	1	Non-maskable interrupt (asynchronous)
irq_acc	Output	14	Interrupt request accepted (one-hot signal)
External Peripherals interface			
per_addr	Output	8	Peripheral address
per_din	Output	16	Peripheral data input
per_dout	Input	16	Peripheral data output

per_en	Output	1	Peripheral enable (high active)
per_wen	Output	2	Peripheral write enable (high active)
RAM interface			
ram_addr	Output	`RAM_AWIDTH[1]	RAM address
ram_cen	Output	1	RAM chip enable (low active)
ram_din	Output	16	RAM data input
ram_dout	Input	16	RAM data output
ram_wen	Output	2	RAM write enable (low active)
ROM interface			
rom_addr	Output	`ROM_AWIDTH[1]	ROM address
rom_cen	Output	1	ROM chip enable (low active)
rom_din_dbg	Output	16	ROM data input --FOR SERIAL DEBUG INTERFACE--
rom_dout	Input	16	ROM data output
rom_wen_dbg	Output	2	ROM write enable (low active) --FOR SERIAL DEBUG INTERFACE--
Serial Debug interface			
dbg_freeze	Output	1	Freeze peripherals
dbg_uart_txd	Output	1	Debug interface: UART TXD
dbg_uart_rxd	Input	1	Debug interface: UART RXD

[1]: This parameter is declared in the "openMSP430_defines.v" file and defines the RAM/ROM size.

2.1.5 Instruction Cycles and Lengths

The number of CPU clock cycles required for an instruction depends on the instruction format and the addressing modes used, not the instruction itself.

In the following tables, the number of cycles refers to the main clock (*MCLK*). Differences with the original MSP430 are highlighted in green (the original value being red).

- **Interrupt and Reset Cycles**

Action	No. of Cycles	Length of Instruction
Return from interrupt (RETI)	5	1
Interrupt accepted	6	-
WDT reset	4	-
Reset (!RST/NMI)	4	-

- **Format-II (Single Operand) Instruction Cycles and Lengths**

Addressing Mode	No. of Cycles			Length of Instruction
	RRA, RRC, SWPB, SXT	PUSH	CALL	
Rn	1	3	3 (4)	1
@Rn	3	4	4	1
@Rn+	3	4 (5)	4 (5)	1
#N	N/A	4	5	2
X(Rn)	4	5	5	2
EDE	4	5	5	2
&EDE	4	5	5	2

- **Format-III (Jump) Instruction Cycles and Lengths**

All jump instructions require one code word, and take two CPU cycles to execute, regardless of whether the jump is taken or not.

- **Format-I (Double Operand) Instruction Cycles and Lengths**

Addressing Mode		No. of Cycles	Length of Instruction
Src	Dst		
Rn	Rm	1	1
	PC	2	1
	x(Rm)	4	2
	EDE	4	2
	&EDE	4	2
@Rn	Rm	2	1
	PC	3 (2)	1

	x(Rm)	5	2
	EDE	5	2
	&EDE	5	2
@Rn+	Rm	2	1
	PC	3	1
	x(Rm)	5	2
	EDE	5	2
	&EDE	5	2
#N	Rm	2	2
	PC	3	2
	x(Rm)	5	3
	EDE	5	3
	&EDE	5	3
x(Rn)	Rm	3	2
	PC	**3 (4)**	2
	x(Rm)	6	3
	EDE	6	3
	&EDE	6	3
EDE	Rm	3	2
	PC	**3 (4)**	2
	x(Rm)	6	3
	EDE	6	3
	&EDE	6	3
&EDE	Rm	3	2
	PC	3	2
	x(Rm)	6	3
	EDE	6	3
	&EDE	6	3

2.1.6 Serial Debug Interface

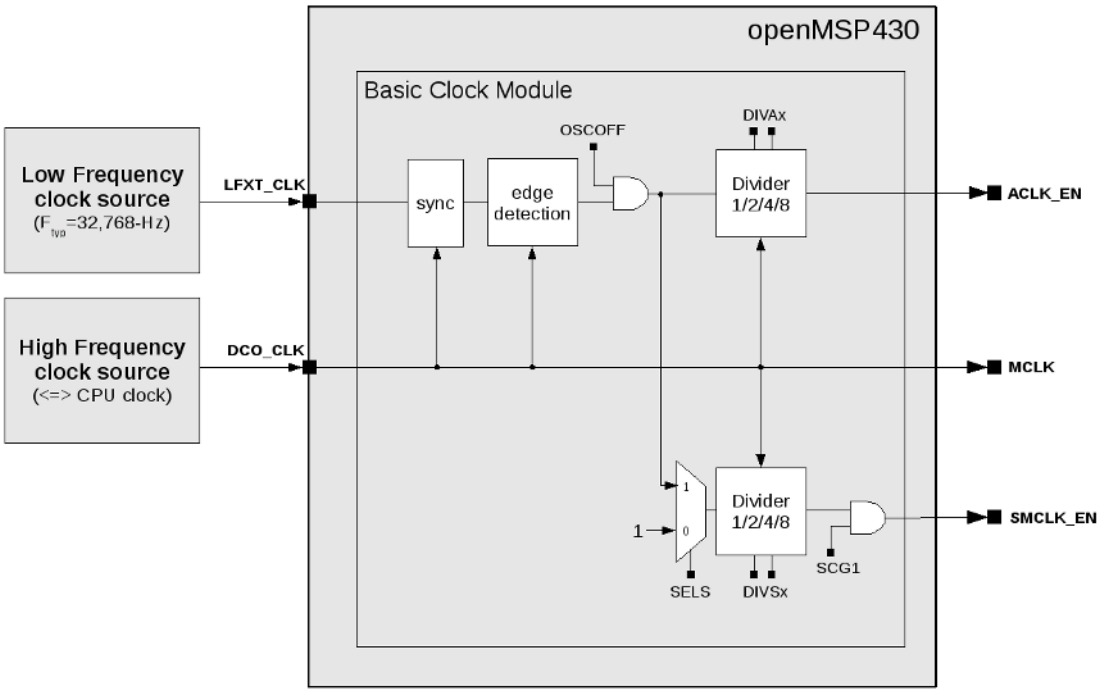
All the details about the Serial Debug Interface are located here.

2.2 Peripherals

In addition to the CPU core itself, several peripherals are also provided and can be easily connected to the core during integration.

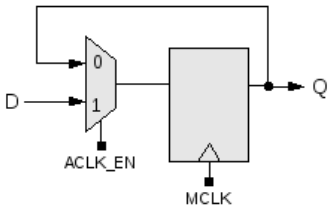
2.2.1 Basic Clock Module

In order to make an FPGA implementation as simple as possible (ideally, a non-designer should be able to do it), clock gates are not used in the design and neither are clock muxes.
With these constrains, the Basic Clock Module is implemented as following:



Note: CPUOFF doesn't switch MCLK off and will instead bring the CPU state machines in an IDLE state while MCLK will still be running.

In order to '*clock*' a register with ACLK or SMCLK, the following structure needs to be implemented:



The following Verilog code would implement a counter clocked with SMCLK:

```
reg [7:0] test_cnt;

always @ (posedge mclk or posedge puc)
if (puc)             test_cnt <= 8'h00;
else if (smclk_en)   test_cnt <= test_cnt + 8'h01;
```

Register Description

- DCOCTL: Not implemented
- BCSCTL1:
    - BCSCTL1[7:6]: Unused
    - BCSCTL1[5:4]: DIVAx
    - BCSCTL1[4:0]: Unused
- BCSCTL2:
    - BCSCTL2[7:4]: Unused
    - BCSCTL2[3] : SELS
    - BCSCTL2[2:1]: DIVSx
    - BCSCTL2[0] : Unused

2.2.2 Watchdog Timer

100% of the features advertised in the MSP430x1xx Family User's Guide (Chapter 10) have been implemented.

2.2.3 Digital I/O

100% of the features advertised in the MSP430x1xx Family User's Guide (Chapter 9) have been implemented.

The following Verilog parameters will enable or disable the corresponding ports in order to save area (i.e. FPGA utilization):

```
parameter P1_EN = 1'b1; // Enable Port 1
parameter P2_EN = 1'b1; // Enable Port 2
parameter P3_EN = 1'b0; // Enable Port 3
parameter P4_EN = 1'b0; // Enable Port 4
parameter P5_EN = 1'b0; // Enable Port 5
parameter P6_EN = 1'b0; // Enable Port 6
```

They can be updated as following during the module instantiation (here port 1, 2 and 3 are enabled):

```
gpio #(.P1_EN(1),
       .P2_EN(1),
       .P3_EN(1),
       .P4_EN(0),
       .P5_EN(0),
       .P6_EN(0)) gpio_0 (
```

The full pinout of the GPIO module is provided in the following table:

Port Name	Direction	Width	Description
Clocks & Resets			
mclk	Input	1	Main system clock
puc	Input	1	Main system reset
Interrupts			
irq_port1	Output	1	Port 1 interrupt
irq_port2	Output	1	Port 2 interrupt
External Peripherals interface			
per_addr	Input	8	Peripheral address
per_din	Input	16	Peripheral data input
per_dout	Output	16	Peripheral data output
per_en	Input	1	Peripheral enable (high active)
per_wen	Input	2	Peripheral write enable (high active)
Port 1			
p1_din	Input	8	Port 1 data input
p1_dout	Output	8	Port 1 data output
p1_dout_en	Output	8	Port 1 data output enable
p1_sel	Output	8	Port 1 function select
Port 2			
p2_din	Input	8	Port 2 data input
p2_dout	Output	8	Port 2 data output
p2_dout_en	Output	8	Port 2 data output enable
p2_sel	Output	8	Port 2 function select
Port 3			
p3_din	Input	8	Port 3 data input

p3_dout	Output	8	Port 3 data output
p3_dout_en	Output	8	Port 3 data output enable
p3_sel	Output	8	Port 3 function select
Port 4			
p4_din	Input	8	Port 4 data input
p4_dout	Output	8	Port 4 data output
p4_dout_en	Output	8	Port 4 data output enable
p4_sel	Output	8	Port 4 function select
Port 5			
p5_din	Input	8	Port 5 data input
p5_dout	Output	8	Port 5 data output
p5_dout_en	Output	8	Port 5 data output enable
p5_sel	Output	8	Port 5 function select
Port 6			
p6_din	Input	8	Port 6 data input
p6_dout	Output	8	Port 6 data output
p6_dout_en	Output	8	Port 6 data output enable
p6_sel	Output	8	Port 6 function select

2.2.4 Timer A

100% of the features advertised in the MSP430x1xx Family User's Guide (Chapter 11) have been implemented.

The full pinout of the Timer A module is provided in the following table:

Port Name	Direction	Width	Description
Clocks, Resets & Debug			
mclk	Input	1	Main system clock
aclk_en	Input	1	ACLK enable (from CPU)
smclk_en	Input	1	SMCLK enable (from CPU)
inclk	Input	1	INCLK external timer clock (SLOW)
taclk	Input	1	TACLK external timer clock (SLOW)
puc	Input	1	Main system reset
dbg_freeze	Input	1	Freeze Timer A counter

Interrupts			
irq_ta0	Output	1	Timer A interrupt: TACCR0
irq_ta1	Output	1	Timer A interrupt: TAIV, TACCR1, TACCR2
irq_ta0_acc	Input	1	Interrupt request TACCR0 accepted
External Peripherals interface			
per_addr	Input	8	Peripheral address
per_din	Input	16	Peripheral data input
per_dout	Output	16	Peripheral data output
per_en	Input	1	Peripheral enable (high active)
per_wen	Input	2	Peripheral write enable (high active)
Capture/Compare Unit 0			
ta_cci0a	Input	1	Timer A capture 0 input A
ta_cci0b	Input	1	Timer A capture 0 input B
ta_out0	Output	1	Timer A output 0
ta_out0_en	Output	1	Timer A output 0 enable
Capture/Compare Unit 1			
ta_cci1a	Input	1	Timer A capture 1 input A
ta_cci1b	Input	1	Timer A capture 1 input B
ta_out1	Output	1	Timer A output 1
ta_out1_en	Output	1	Timer A output 1 enable
Capture/Compare Unit 2			
ta_cci2a	Input	1	Timer A capture 2 input A
ta_cci2b	Input	1	Timer A capture 2 input B
ta_out2	Output	1	Timer A output 2
ta_out2_en	Output	1	Timer A output 2 enable

Note: for the same reason as with the Basic Clock Module, the two additional clock inputs (TACLK and INCLK) are internally synchronized with the MCLK domain. As a consequence, TACLK and INCLK should be at least 2 times slowlier than MCLK, and if these clock are used toghether with the Timer A output unit, some jitter might be observed on the generated output. If this jitter is critical for the application, ACLK and INCLK should idealy be derivated from DCO_CLK.

3.

Serial Debug Interface

Table of content

1. Introduction

The original MSP430 from TI provides a serial debug interface to give a simple path to software development. In that case, the communication with the host computer is typically build on a JTAG or Spy-Bi-Wire serial protocol. However, the global debug architecture from the MSP430 is unfortunately poorly documented on the web (and is also probably tightly linked with the internal core architecture).

A custom module has therefore been implemented for the openMSP430. The communication with the host is done with a simple RS232 cable (8N1 serial protocol) and the debug unit provides all the required features for Nexus Class 3 debugging (beside trace), namely:

- CPU control (run, stop, step, reset).
- Software & hardware breakpoint support.
- Memory read/write on-the-fly (no need to halt execution).
- CPU registers read/write on-the-fly (no need to halt execution).

2. Debug Unit

2.1 Register Mapping

The following table summarize the complete debug register set accessible through the debug communication interface:

Register Name	Address	Bit Field															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CPU_ID_LO	0x00	CPU_ID[7:0]								ROM_AWIDTH				RAM_AWIDTH			
CPU_ID_HI	0x01	CPU_ID[23:8]															
CPU_CTL	0x02	Reserved									CPU_RST	RST_BRK_EN	FRZ_BRK_EN	SW_BRK_EN	ISTEP	RUN	HALT
CPU_STAT	0x03	Reserved								HWBRK3_PND	HWBRK2_PND	HWBRK1_PND	HWBRK0_PND	SWBRK_PND	PUC_PND	Res.	HALT_RUN
MEM_CTL	0x04	Reserved												B/W	MEM/REG	RD/WR	START
MEM_ADDR	0x05	MEM_ADDR[15:0]															
MEM_DATA	0x06	MEM_DATA[15:0]															
MEM_CNT	0x07	MEM_CNT[15:0]															
BRK0_CTL	0x08	Reserved											RANGE_MODE	INST_EN	BREAK_EN	ACCESS_MODE	
BRK0_STAT	0x09	Reserved										RANGE_WR	RANGE_RD	ADDR1_WR	ADDR1_RD	ADDR0_WR	ADDR0_RD
BRK0_ADDR0	0x0A	BRK_ADDR0[15:0]															
BRK0_ADDR1	0x0B	BRK_ADDR1[15:0]															
BRK1_CTL	0x0C	Reserved											RANGE_MODE	INST_EN	BREAK_EN	ACCESS_MODE	
BRK1_STAT	0x0D	Reserved										RANGE_WR	RANGE_RD	ADDR1_WR	ADDR1_RD	ADDR0_WR	ADDR0_RD
BRK1_ADDR0	0x0E	BRK_ADDR0[15:0]															
BRK1_ADDR1	0x0F	BRK_ADDR1[15:0]															
BRK2_CTL	0x10	Reserved											RANGE_MODE	INST_EN	BREAK_EN	ACCESS_MODE	

BRK2_STAT	0x11	Reserved	RANGE_WR	RANGE_RD	ADDR1_WR	ADDR1_RD	ADDR0_WR	ADDR0_RD
BRK2_ADDR0	0x12	BRK_ADDR0[15:0]						
BRK2_ADDR1	0x13	BRK_ADDR1[15:0]						
BRK3_CTL	0x14	Reserved		RANGE_MODE	INST_EN	BREAK_EN	ACCESS_MODE	
BRK3_STAT	0x15	Reserved	RANGE_WR	RANGE_RD	ADDR1_WR	ADDR1_RD	ADDR0_WR	ADDR0_RD
BRK3_ADDR0	0x16	BRK_ADDR0[15:0]						
BRK3_ADDR1	0x17	BRK_ADDR1[15:0]						

2.2 CPU Control/Status Registers

2.2.1 CPU_ID

This 32 bit read-only register holds the ID of the implemented openMSP430 as well as the RAM and ROM size information.

Register Name	Address	Bit Field															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CPU_ID_LO	0x00	CPU_ID[7:0]								ROM_AWIDTH				RAM_AWIDTH			
CPU_ID_HI	0x01	CPU_ID[23:7]															

- **CPU_ID** : Set by default to 0x4D5350 (ascii code for "*MSP*")
- **ROM_AWIDTH** : Program memory address width for the current implementation. The ROM size is then equal to 2^{ROM_AWIDTH}
- **RAM_AWIDTH** : Data memory address width for the current implementation. The RAM size is then equal to 2^{RAM_AWIDTH}

2.2.2 CPU_CTL

This 8 bit read-write register is used to control the CPU and to configure some basic debug features. After a POR, this register is set to 0x00.

Register Name	Address	Bit Field							
		7	6	5	4	3	2	1	0
CPU_CTL	0x02	Res.	CPU_RST	RST_BRK_EN	FRZ_BRK_EN	SW_BRK_EN	ISTEP	RUN	HALT

- **CPU_RST** : Setting this bit to 1 will activate the PUC reset. Setting it back to 0 will release it.
- **RST_BRK_EN** : If set to 1, the CPU will automatically break after a PUC occurrence.
- **FRZ_BRK_EN** : If set to 1, the timers and watchdog are frozen when the CPU is

halted.

- **SW_BRK_EN** : Enables the software breakpoint detection.
- **ISTEP**[1] : Writing 1 to this bit will perform a single instruction step if the CPU is halted.
- **RUN**[1] : Writing 1 to this bit will get the CPU out of halt state.
- **HALT**[1] : Writing 1 to this bit will put the CPU in halt state.

[1]:this field is write-only and always reads back 0.

2.2.3 CPU_STAT

This 8 bit read-write register gives the global status of the debug interface. After a POR, this register is set to 0x00.

Register Name	Address	Bit Field							
		7	6	5	4	3	2	1	0
CPU_STAT	0x03	HWBRK3_PND	HWBRK2_PND	HWBRK1_PND	HWBRK0_PND	SWBRK_PND	PUC_PND	Res.	HALT_RUN

- **HWBRK3_PND** : This bit reflects if one of the Hardware Breakpoint Unit 3 status bit is set (i.e. BRK3_STAT≠0).
- **HWBRK2_PND** : This bit reflects if one of the Hardware Breakpoint Unit 2 status bit is set (i.e. BRK2_STAT≠0).
- **HWBRK1_PND** : This bit reflects if one of the Hardware Breakpoint Unit 1 status bit is set (i.e. BRK1_STAT≠0).
- **HWBRK0_PND** : This bit reflects if one of the Hardware Breakpoint Unit 0 status bit is set (i.e. BRK0_STAT≠0).
- **SWBRK_PND** : This bit is set to 1 when a software breakpoint occurred. It can be cleared by writing 1 to it.
- **PUC_PND** : This bit is set to 1 when a PUC reset occurred. It can be cleared by writing 1 to it.
- **HALT_RUN** : This read-only bit gives the current status of the CPU:

  0 - CPU is running.
  1 - CPU is stopped.

2.3 Memory Access Registers

The following four registers enable single and burst read/write access to both CPU-Registers and full memory address range.

In order to perform an access, the following sequences are typically done:

- single read access (MEM_CNT=0):
    1. set MEM_ADDR with the memory address (or register number) to be read
    2. set MEM_CTL (in particular RD/WR=0 and START=1)
    3. read MEM_DATA
- single write access (MEM_CNT=0):
    1. set MEM_ADDR with the memory address (or register number) to be written
    2. set MEM_DATA with the data to be written
    3. set MEM_CTL (in particular RD/WR=1 and START=1)
- burst read/write access (MEM_CNT≠0):
    - burst access are optimized for the communication interface used (i.e. for the UART). The burst sequence are therefore described in the corresponding section (3.4 Read/Write burst implementation for the CPU Memory access)

2.3.1 MEM_CTL

This 8 bit read-write register is used to control the Memory and CPU-Register read/write access. After a POR, this register is set to 0x00.

Register Name	Address	Bit Field							
		7	6	5	4	3	2	1	0
MEM_CTL	0x04	Reserved				B/W	MEM/REG	RD/WR	START

- **B/W** : **0** - 16 bit access.

  1 - 8 bit access (not valid for CPU-Registers).
- **MEM/REG** : **0** - Memory access.

  1 - CPU-Register access.
- **RD/WR** : **0** - Read access.

  1 - Write access.
- **START** : **0**- Do nothing

  1 - Initiate memory transfer.

2.3.2 MEM_ADDR

This 16 bit read-write register specifies the Memory or CPU-Register address to be used for the next read/write transfer. After a POR, this register is set to 0x0000.

Note: in case of burst (i.e. MEM_CNT≠0), this register specifies the first address of the burst transfer and will be incremented automatically as the burst goes (by 1 for 8-bit access and by 2 for 16-bit access).

Register Name	Address	Bit Field															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEM_ADDR	0x05	MEM_ADDR[15:0]															

- **MEM_ADDR** : Memory or CPU-Register address to be used for the next read/write transfer.

2.3.3 MEM_DATA

This 16 bit read-write register specifies (wr) or receive (rd) the Memory or CPU-Register data for the the next transfer. After a POR, this register is set to 0x0000.

Register Name	Address	Bit Field															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEM_DATA	0x06	MEM_DATA[15:0]															

- **MEM_DATA** : if MEM_CTL.WR - data to be written during the next write transfer.

  if MEM_CTL.RD - updated with the data from the read transfer

2.3.4 MEM_CNT

This 16 bit read-write register controls the burst access to the Memory or CPU-Registers. If set to 0, a single access will occur, otherwise, a burst will be performed. The burst being optimized for the communication interface, more details are given there. After a POR, this register is set to 0x0000.

Register Name	Address	Bit Field															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEM_CNT	0x07	MEM_CNT[15:0]															

- **MEM_CNT** : **=0** - a single access will be performed with the next transfer.

  ≠0 - specifies the burst size for the next transfer (i.e number of data access). This field will be automatically decremented as the burst goes.

2.4 Hardware Breakpoint Unit Registers

Depending on the defines located in the "*openMSP430_defines.v*" file, up to four hardware breakpoint units can be included in the design. These units can be individually controlled with the following registers.

2.4.1 BRKx_CTL

This 8 bit read-write register controls the hardware breakpoint unit x. After a POR, this register is set to 0x00.

Register Name	Address	Bit Field							
		7	6	5	4	3	2	1	0
BRKx_CTL	0x08, 0x0C, 0x10, 0x14	Reserved			RANGE_MODE	INST_EN	BREAK_EN	ACCESS_MODE	

- **RANGE_MODE** : **0** - Address match on BRK_ADDR0 or BRK_ADDR1 (normal mode)

  1 - Address match on BRK_ADDR0→BRK_ADDR1 range (range mode)
- **INST_EN** : **0** - Checks are done on the execution unit (data flow).

  1 - Checks are done on the frontend (instruction flow).
- **BREAK_EN** : **0** - Watchpoint mode enable (don't stop on address match).

  1 - Breakpoint mode enable (stop on address match).
- **ACCESS_MODE** : **00** - Disabled

  01 - Detect read access.

  10 - Detect write access.

  11 - Detect read/write access

  Note: **'10'** & **'11'** modes are not supported on the instruction flow

2.4.2 BRKx_STAT

This 8 bit read-write register gives the status of the hardware breakpoint unit x. Each status bit can be cleared by writing 1 to it. After a POR, this register is set to 0x00.

Register Name	Address	Bit Field							
		7	6	5	4	3	2	1	0
BRKx_STAT	0x09, 0x0D, 0x11, 0x15	Reserved		RANGE_WR	RANGE_RD	ADDR1_WR	ADDR1_RD	ADDR0_WR	ADDR0_RD

- **RANGE_WR** : This bit is set whenever the CPU performs a write access within the BRKx_ADDR0→BRKx_ADDR1 range (valid if RANGE_MODE=1 and ACCESS_MODE[1]=1).
- **RANGE_RD** : This bit is set whenever the CPU performs a read access within the BRKx_ADDR0→BRKx_ADDR1 range (valid if RANGE_MODE=1 and ACCESS_MODE[0]=1).
- **ADDR1_WR** : This bit is set whenever the CPU performs a write access at the BRKx_ADDR1 address (valid if RANGE_MODE=0 and ACCESS_MODE[1]=1).
- **ADDR1_RD** : This bit is set whenever the CPU performs a read access at the BRKx_ADDR1 address (valid if RANGE_MODE=0 and ACCESS_MODE[0]=1).
- **ADDR0_WR** : This bit is set whenever the CPU performs a write access at the BRKx_ADDR0 address (valid if RANGE_MODE=0 and ACCESS_MODE[1]=1).
- **ADDR0_RD** : This bit is set whenever the CPU performs a read access at the BRKx_ADDR0 address (valid if RANGE_MODE=0 and ACCESS_MODE[0]=1).

2.4.3 BRKx_ADDR0

This 16 bit read-write register holds the value which is compared against the address value currently present on the program or data address bus. After a POR, this register is set to 0x0000.

Register Name	Address	Bit Field															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BRKx_ADDR0	0x0A, 0x0E, 0x12, 0x16	BRK_ADDR0[15:0]															

- **BRK_ADDR0** : Value compared against the address value currently present on the program or data address bus.

2.4.4 BRKx_ADDR1

This 16 bit read-write register holds the value which is compared against the address value currently present on the program or data address bus. After a POR, this register is set to 0x0000.

Register Name	Addresses	Bit Field															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BRKx_ADDR1	0x0B, 0x0F, 0x13, 0x17	BRK_ADDR1[15:0]															

- **BRK_ADDR1** : Value compared against the address value currently present on the program or data address bus.

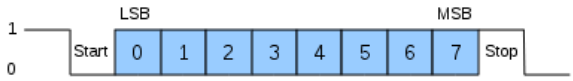
3. Debug Communication Interface: UART

With its UART interface, the openMSP430 debug unit can communicate with the host computer using a simple RS232 cable (connected to the dbg_uart_txd and dbg_uart_rxd ports of the IP).

Using an standard USB to RS232 adaptor, the interface provides a reliable communication link up to 1,5Mbps.

3.1 Serial communication protocol: 8N1

There are plenty tutorials on Internet regarding RS232 based protocols. However, here is quick recap about 8N1 (1 Start bit, 8 Data bits, No Parity, 1 Stop bit):

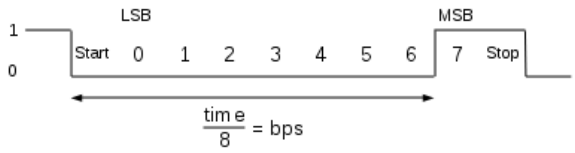


As you can see in the above diagram, data transmission starts with a Start bit, followed by the data bits (LSB sent first and MSB sent last), and ends with a "Stop" bit.

3.2 Synchronization frame

After a POR, the Serial Debug Interface expects a synchronization frame from the host computer in order to determine the communication speed (i.e. the baud rate).

The synchronization frame looks as following:



As you can see, the host simply sends the 0x80 value. The openMSP430 will then measure the time between the falling and rising edge, divide it by 8 and automatically deduce the baud rate it should use to properly communicate with the host.

Important note: if you want to change the communication speed between two debugging sessions, the openMSP430 needs to go over a POR cycle and a new synchronization frame needs to be send.

3.3 Read/Write access to the debug registers

In order to perform a read / write access to a debug register, the host needs to send a command frame to the openMSP430.

In case of write access, this command frame will be followed by 1 or 2 data frames and in case of read access, the openMSP430 will send 1 or 2 data frames after receiving the command.

3.3.1 Command Frame

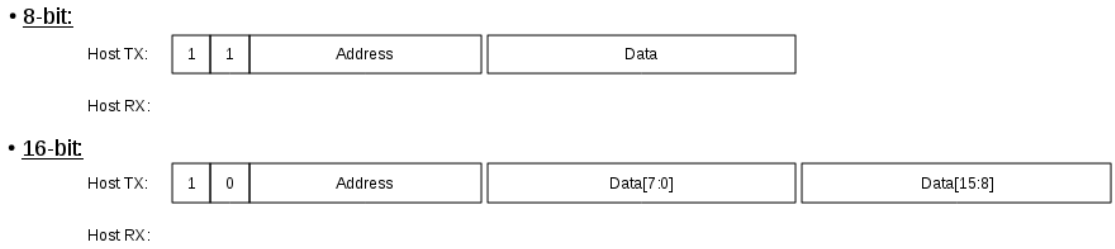
The command frame looks as following:

7	6	5	4	3	2	1	0
WR	B/W	Address					

- **WR** : Perform a Write access when set. Read otherwise.
- **B/W** : Perform a 8-bit data access when set (one data frame). 16-bit otherwise (two data frame).
- **Address** : Debug register address.

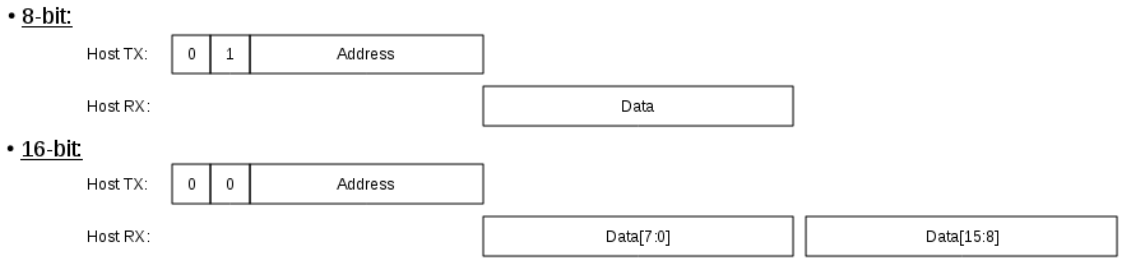
3.3.2 Write access

A write access transaction looks like this:



3.3.3 Read access

A read access transaction looks like this:



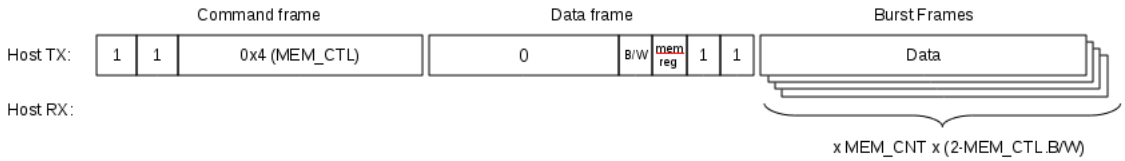
3.4 Read/Write burst implementation for the CPU Memory access

In order to optimize the data burst transactions for the UART, read/write access are not done by reading or writing the MEM_DATA register.

Instead, the data transfer starts immediately after the MEM_CTL.START bit has been set.

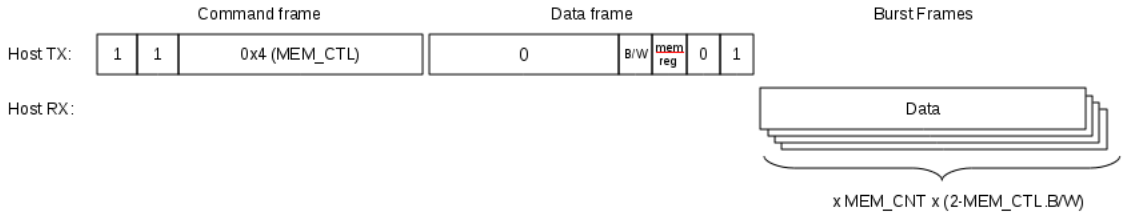
3.4.1 Write Burst access

A write burst transaction looks like this:



3.4.2 Read Burst access

A read burst transaction looks like this:



4.

Software Development Tools

Table of content

1. Introduction

Building on the serial debug interface capabilities provided by the openMSP430, three small utility programs are provided:

- **openmsp430-loader:** a simple command line boot loader.
- **openmsp430-minidebug:** a minimalistic debugger with simple GUI.
- **openmsp430-gdbproxy:** GDB Proxy server to be used together with MSP430-GDB and the Eclipse, DDD, or Insight graphical front-ends.

All these software development tools have been developed in TCL/TK and were successfully tested on both Linux and Windows XP.

Note: in order to be able to directly execute the scripts, TCL/TK needs to be installed on your system. Optionally for Windows users, the scripts have been turned into single-file binary executable programs using freeWrap.

2. openmsp430-loader

This simple program allows the user to load the openMSP430 program memory with an executable file (ELF format) provided as argument.

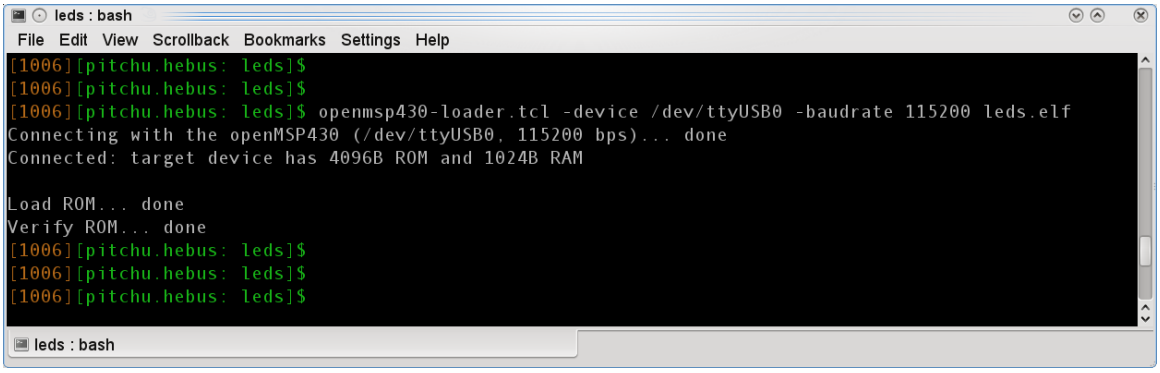
It is typically used in conjunction with '***make***' in order to automatically load the program after the compile step (see '***Makefile***' from software examples provided with the project's FPGA implementation).

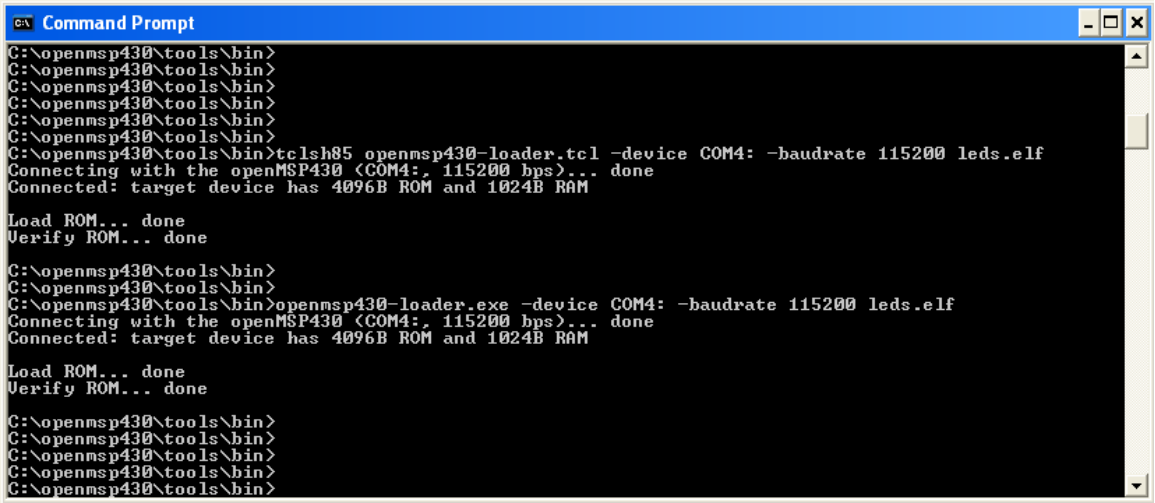
The program can be called with the following syntax:

```
openmsp430-loader.tcl [-device <communication device>] [-baudrate <communication speed>] <elf-file>

Examples:          openmsp430-loader.tcl -device /dev/ttyUSB0    -baudrate 9600    leds.elf
                   openmsp430-loader.tcl -device COM2:           -baudrate 38400   ta_uart.elf
```

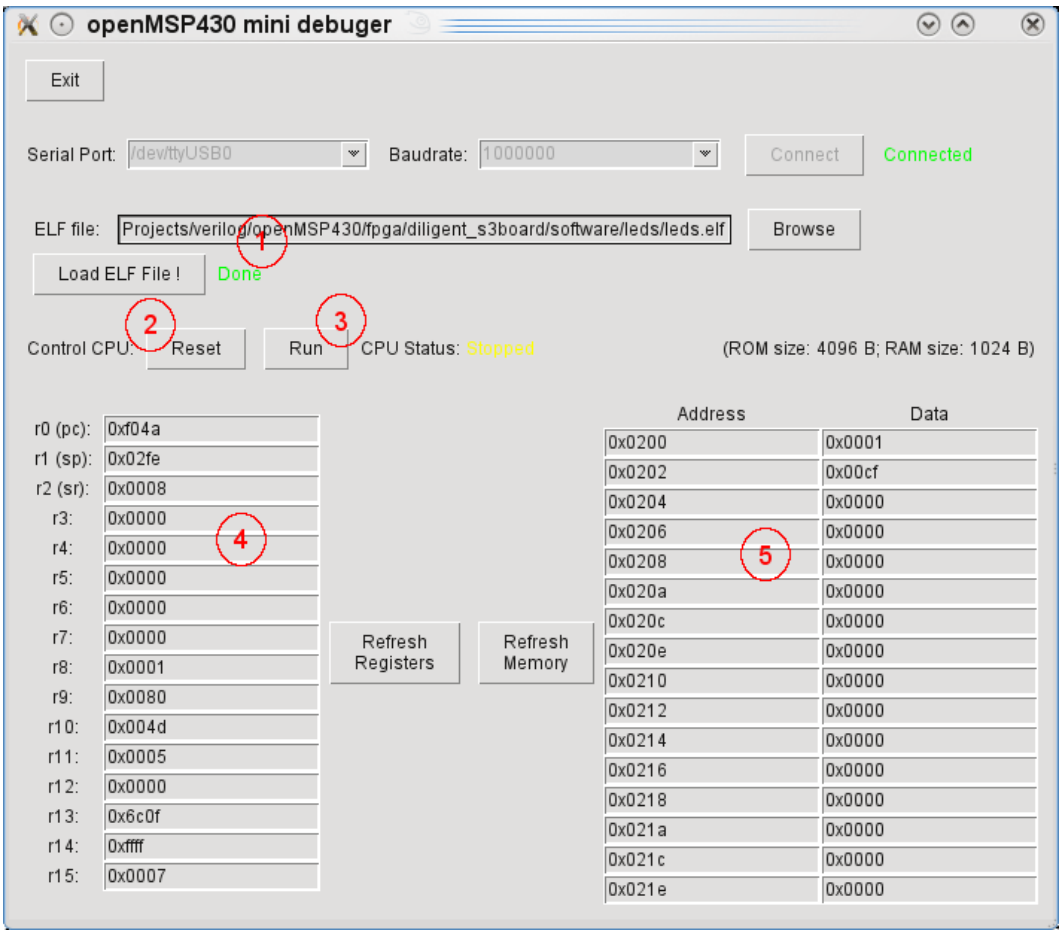
These screenshots show the script in action under Linux and Windows:





3. openmsp430-minidebug

This small program provides a minimalistic graphical interface enabling simple interaction with the openMSP430:



As you can see from the screenshot, it allows the following actions:

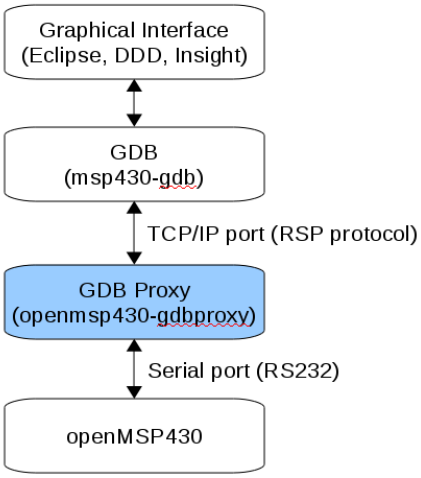
- **(1)** Load the program memory with an ELF file
- **(2)** Reset the CPU
- **(3)** Stop/Start the program execution
- **(4)** Read/Write access of the CPU registers
- **(5)** Read/Write access of the whole memory range (program, data, peripherals)

4. openmsp430-gdbproxy

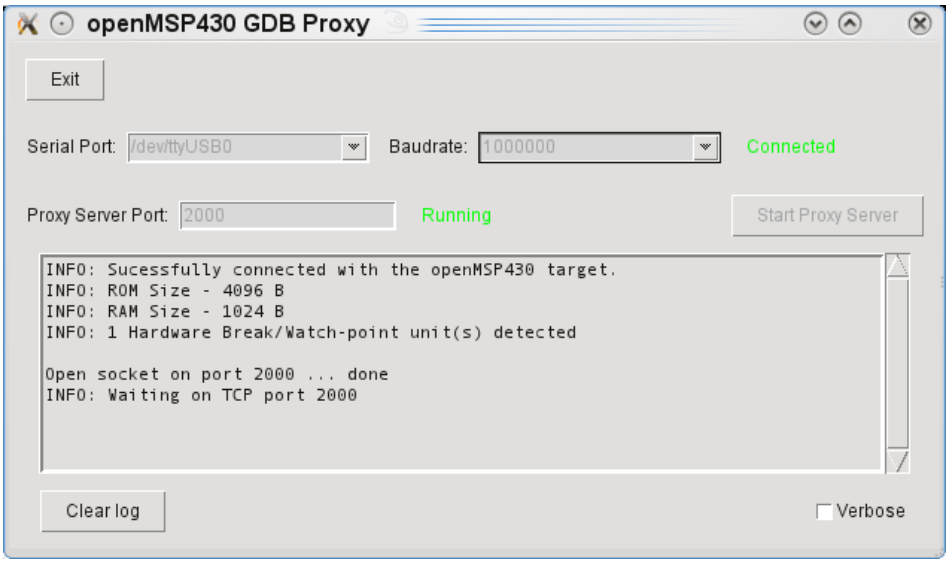
The purpose of this program is to replace the '***msp430-gdbproxy***' utility provided by the mspgcc toolchain.

Typically, a GDB proxy creates a local port for gdb to connect to, and handles the communication with the target hardware. In our case, it is basically a bridge between the RSP communication protocol from GDB and the serial debug interface from the openMSP430.

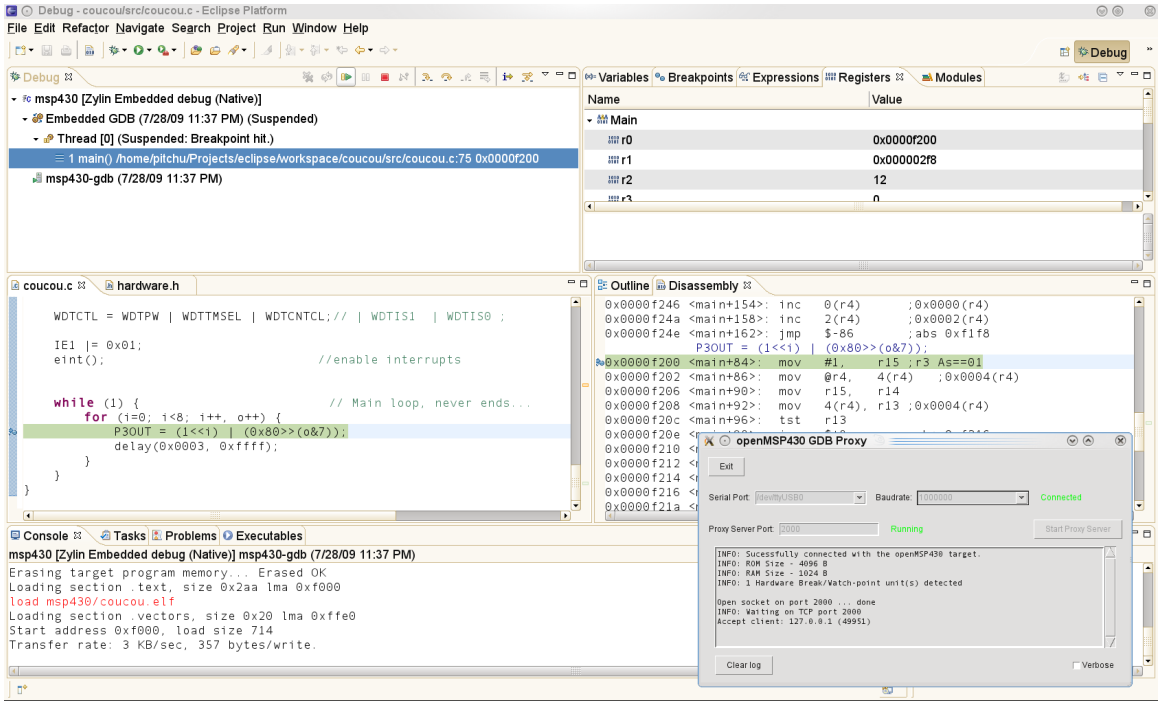
Schematically the communication flow looks as following:

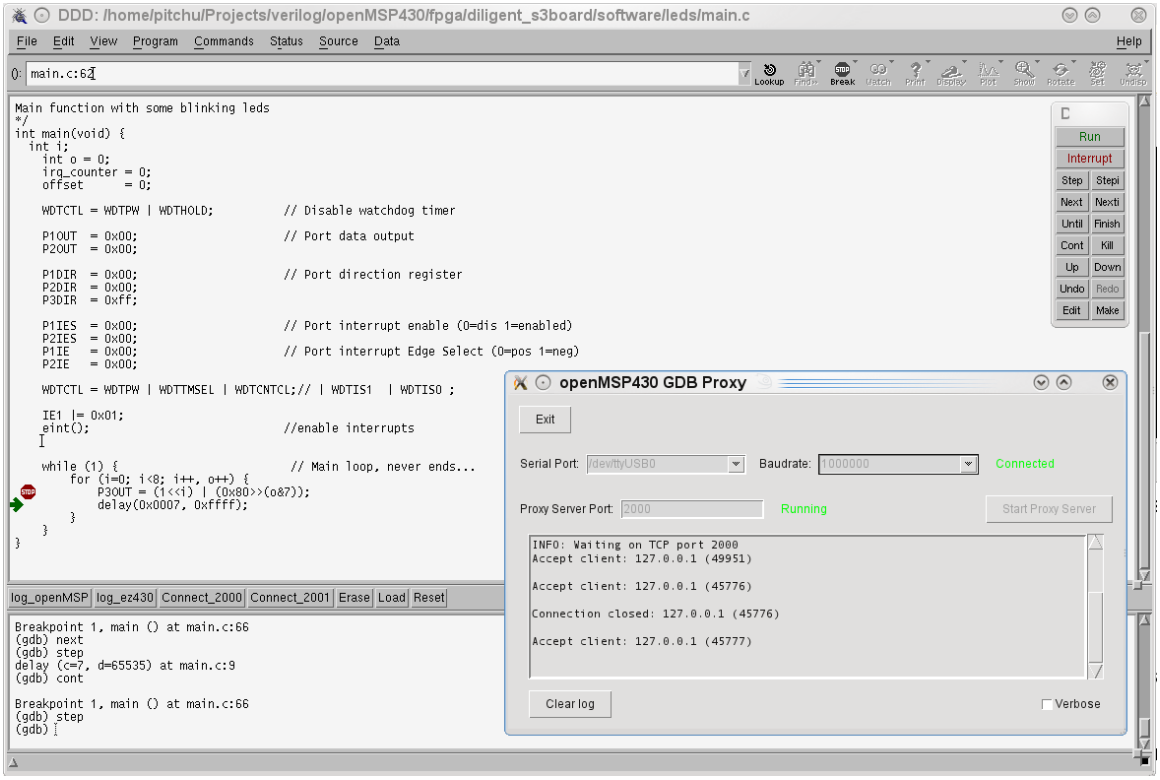


Like the original '***msp430-gdbproxy***' program, '***openmsp430-gdbproxy***' can be controlled from the command line. However, it also provides a small graphical interface:



These two additional screenshots show the script in action together with the Eclipse and DDD graphical frontends:





Tip: There are several tutorials on Internet explaining how to configure Eclipse for the MSP430. As an Eclipse newbie, I found the followings quite helpful:

- Use Eclipse and mspgcc - The easy way (English)
- MSP430 - Entwicklungumgebung (German)

5. MSPGCC Toolchain

5.1 Some notes regarding msp430-gdb

As of today (July 2009), the GDB port for the MSP430 has some problems (here).

The stepping over function is not available and the backtrace and finish commands don't work properly.

There is fortunately a patch existing, and until it is included into GDB, I can only recommend to recompile GDB with it (I didn't try it for Windows but it is quite straight forward to do for Linux).

5.2 CPU selection for msp430-gcc

The following table aims to help selecting the proper **-mmcu** option for the **msp430-gcc** call.

Note that only the ROM size should imperatively match the openMSP430 configuration.

ROM Size: 1 kB		
msp430x110	1 kB	128 B
msp430x1101	1 kB	128 B
msp430x2001	1 kB	128 B
msp430x2002	1 kB	128 B
msp430x2003	1 kB	128 B
msp430x2101	1 kB	128 B
ROM Size: 2 kB		
msp430x1111	2 kB	128 B
msp430x2011	2 kB	128 B
msp430x2012	2 kB	128 B
msp430x2013	2 kB	128 B

msp430x2111	2 kB	128 B
msp430x2112	2 kB	128 B
msp430x311	2 kB	128 B
ROM Size: 4 kB		
msp430x112	4 kB	256 B
msp430x1121	4 kB	256 B
msp430x1122	4 kB	256 B
msp430x122	4 kB	256 B
msp430x1222	4 kB	256 B
msp430x2122	4 kB	256 B
msp430x2121	4 kB	256 B
msp430x312	4 kB	256 B
msp430x412	4 kB	256 B
ROM Size: 8 kB		
msp430x123	8 kB	256 B
msp430x133	8 kB	256 B
msp430x313	8 kB	256 B
msp430x323	8 kB	256 B
msp430x413	8 kB	256 B
msp430x423	8 kB	256 B
msp430xE423	8 kB	256 B
msp430xE4232	8 kB	256 B
msp430xW423	8 kB	256 B
msp430x1132	8 kB	256 B
msp430x1232	8 kB	256 B
msp430x1331	8 kB	256 B
msp430x2131	8 kB	256 B
msp430x2132	8 kB	256 B
msp430x2232	8 kB	512 B
msp430x2234	8 kB	512 B
msp430x233	8 kB	1024 B
msp430x2330	8 kB	1024 B
ROM Size: 16 kB		

msp430x4250	16 kB	256 B
msp430xG4250	16 kB	256 B
msp430x135	16 kB	512 B
msp430x1351	16 kB	512 B
msp430x155	16 kB	512 B
msp430x2252	16 kB	512 B
msp430x2254	16 kB	512 B
msp430x315	16 kB	512 B
msp430x325	16 kB	512 B
msp430x415	16 kB	512 B
msp430x425	16 kB	512 B
msp430xE425	16 kB	512 B
msp430xW425	16 kB	512 B
msp430xE4252	16 kB	512 B
msp430x435	16 kB	512 B
msp430x4351	16 kB	512 B
msp430x235	16 kB	2048 B
msp430x2350	16 kB	2048 B
ROM Size: 32 kB		
msp430x4270	32 kB	256 B
msp430xG4270	32 kB	256 B
msp430x147	32 kB	1024 B
msp430x1471	32 kB	1024 B
msp430x157	32 kB	1024 B
msp430x167	32 kB	1024 B
msp430x2272	32 kB	1024 B
msp430x2274	32 kB	1024 B
msp430x337	32 kB	1024 B
msp430x417	32 kB	1024 B
msp430x427	32 kB	1024 B
msp430xE427	32 kB	1024 B
msp430xE4272	32 kB	1024 B
msp430xW427	32 kB	1024 B

msp430x437	32 kB	1024 B
msp430xG437	32 kB	1024 B
msp430x4371	32 kB	1024 B
msp430x447	32 kB	1024 B
msp430x2370	32 kB	2048 B
msp430x247	32 kB	4096 B
msp430x2471	32 kB	4096 B

5.

File and Directory Description

Table of content

1. Introduction

To simplify the integration of this IP, the directory structure is based on the OpenCores recommendations.

2. Directory structure: openMSP430 core

core				***openMSP430 Core top level directory***
	bench			***Top level testbench directory***
		verilog		
			tb_openMSP430.v	*Testbench top level module*
			ram.v	*RAM verilog model*

			registers.v	*Connections to Core internals for easy debugging*
			dbg_uart_tasks.v	*UART tasks for the serial debug interface*
			msp_debug.v	*Testbench instruction decoder and ASCII chain generator for easy debugging*
	doc			***Diverse documentation***
		slau049f.pdf		*MSP430x1xx Family User's Guide*
	rtl			***RTL sources***
		verilog		
			openMSP430_defines.v	*openMSP430 core configuration file (ROM and RAM size definition, Debug Interface configuration)*
			openMSP430.v	*openMSP430 top level*
			frontend.v	*Instruction fetch and decode*
			execution_unit.v	*Execution unit*
			alu.v	*ALU*
			register_file.v	*Register file*
			mem_backbone.v	*Memory backbone*
			clock_module.v	*Basic Clock Module*
			sfr.v	*Special function registers*
			watchdog.v	*Watchdog Timer*
			dbg.v	*Serial Debug Interface main block*
			dbg_hwbrk.v	*Serial Debug Interface hardware breakpoint unit*
			dbg_uart.v	*Serial Debug Interface UART communication block*
			periph	***Peripherals directory***
				gpio.v	*Digital I/O (Port 1 to 6)*
				timerA.v	*Timer A*
				template_periph_16b.v	*Verilog template for 16 bit peripherals*
				template_periph_8b.v	*Verilog template for 8 bit peripherals*
	sim			***Top level simulations directory***
		rtl_sim		***RTL simulations***
			bin	***RTL simulation scripts***

				msp430sim	*Main simulation script*
				asm2ihex.sh	*Assembly file compilation (Intel HEX file generation)*
				ihex2mem.tcl	*Verilog ROM memory file generation*
				rtlsim.sh	*Verilog Icarus simulation script*
				template.def	*ASM linker definition file template*
			run		***For running RTL simulations***
				run	*Run single simulation of a given vector*
				run_all	*Run regression of all vectors*
				run_disassemble	*Disassemble ROM content of the latest simulation*
				load_waveform.sav	*SAV file for gtkWave*
			src		***RTL simulation vectors sources***
				submit.f	*Verilog simulator command file*
				sing-op_*.s43	*Single-operand assembler vector files*
				sing-op_*.v	*Single-operand verilog stimulus vector files*
				two-op_*.s43	*Two-operand assembler vector files*
				two-op_*.v	*Two-operand verilog stimulus vector files*
				c-jump_*.s43	*Jump assembler vector files*
				c-jump_*.v	*Jump verilog stimulus vector files*
				op_modes.s43	*CPU operating modes assembler vector files (CPUOFF, OSCOFF, SCG1)*
				op_modes.v	*CPU operating modes verilog stimulus vector files (CPUOFF, OSCOFF, SCG1)*
				clock_module.s43	*Basic Clock Module assembler vector files*
				clock_module.v	*Basic Clock Module verilog stimulus vector files*
				dbg_*.s43	*Serial Debug Interface assembler vector files*
				dbg_*.v	*Serial Debug Interface verilog stimulus vector files*
				gpio_*.s43	*Digital I/O assembler vector files*
				gpio_*.v	*Digital I/O verilog stimulus vector files*
				template_periph_*.s43	*Peripheral templates assembler vector files*

				template_periph_*.v	*Peripheral templates verilog stimulus vector files*
				wdt_*.s43	*Watchdog timer assembler vector files*
				wdt_*.v	*Watchdog timer verilog stimulus vector files*
				tA_*.s43	*Timer A assembler vector files*
				tA_*.v	*Timer A verilog stimulus vector files*
	synthesis				***Top level synthesis directory***
		synopsys			*Synopsys (Design Compiler) directory*
			run_syn		*Run synthesis*
			synthesis.tcl		*Main synthesis TCL script*
			library.tcl		*Load library, set operating conditions and wire load models*
			read.tcl		*Read RTL*
			constraints.tcl		*Set design constrains*
			results		*Results directory*

3. Directory structure: FGPA projects

3.1 Xilinx Spartan 3 example

fpga					***openMSP430 FPGA Projects top level directory***
	xilinx_diligent_s3board				***Xilinx FPGA Project based on the Diligent Spartan-3 board***
		bench			***Top level testbench directory***
			verilog		
				tb_openMSP430_fpga.v	*FPGA testbench top level module*
				registers.v	*Connections to Core internals for easy debugging*
				msp_debug.v	*Testbench instruction decoder and ASCII chain generator for easy debugging*
				glbl.v	*Xilinx "glbl.v" file*
		doc			***Diverse documentation***

			board_user_guide.pdf			*Spartan-3 FPGA Starter Kit Board User Guide*
			msp430f1121a.pdf			*msp430f1121a Specification*
			xapp462.pdf			*Xilinx Digital Clock Managers (DCMs) user guide*
		rtl				***RTL sources***
			verilog			
				openMSP430_fpga.v		*FPGA top level file*
				driver_7segment.v		*Four-Digit, Seven-Segment LED Display driver*
				io_mux.v		*I/O mux for port function selection.*
				openmsp430		***Local copy of the openMSP430 core.*** *The *define.v file has been adjusted to the requirements of the project.*
				coregen		*Xilinx's coregen directory*
					ram_8x512_hi.*	*512 Byte RAM (upper byte)*
					ram_8x512_lo.*	*512 Byte RAM (lower byte)*
					rom_8x2k_hi.*	*2 kByte ROM (upper byte)*
					rom_8x2k_lo.*	*2 kByte ROM (lower byte)*
		sim				***Top level simulations directory***
			rtl_sim			***RTL simulations***
				bin		***RTL simulation scripts***
					msp430sim	*Main simulation script*
					ihex2mem.tcl	*Verilog ROM memory file generation*
					rtlsim.sh	*Verilog Icarus simulation script*
				run		***For running RTL simulations***
					run	*Run simulation of a given software project*
					run_disassemble	*Disassemble ROM content of the latest simulation*
				src		***RTL simulation verilog stimulus***
					submit.f	*Verilog simulator command file*
					*.v	*Stimulus vector for the corresponding software project*
		software				***Software C programs to be loaded in***

					ROM
			leds		*LEDs blinking application (from the CDK4MSP project)*
				makefile	
				hardware.h	
				main.c	
				7seg.h	
				7seg.c	
			ta_uart		*Software UART with Timer_A (from the CDK4MSP project)*
		synthesis			***Top level synthesis directory***
			xilinx		
				create_bitstream.sh	*Run Xilinx ISE synthesis in a Linux environment*
				create_bitstream.bat	*Run Xilinx ISE synthesis in a Windows environment*
				openMSP430_fpga.ucf	*UCF file*
				openMSP430_fpga.prj	*RTL file list to be synthesized*
				xst_verilog.opt	*Verilog Option File for XST. Among other things, the search path to the include files is specified here.*
				load_rom.sh	*Update bitstream's ROM with a given software ELF file in a Linux environment*
				load_rom.bat	*Update bitstream's ROM with a given software ELF file in a Windows environment*
				memory.bmm	*FPGA memory description for bitstream's ROM update*

3.2 Altera Cyclone II example

fpga					***openMSP430 FPGA Projects top level directory***
	altera_de1_board				***Altera FPGA Project based on Cyclone II Starter Development Board***
		README			*README file*
		bench			***Top level testbench directory***
			verilog		
				tb_openMSP430_fpga.v	*FPGA testbench top level module*
				registers.v	*Connections to Core internals for easy debugging*
				msp_debug.v	*Testbench instruction decoder and ASCII chain generator for easy debugging*
				altsyncram.v	*Altera verilog model of the altsyncram module..*
		doc			***Diverse documentation***
			DE1_Board_Schematic.pdf		*Cyclone II FPGA Starter Development Board Schematics*
			DE1_Reference_Manual.pdf		*Cyclone II FPGA Starter Development Board Reference Manual*
			DE1_User_Guide.pdf		*Cyclone II FPGA Starter Development Board User Guide*
		rtl			***RTL sources***
			verilog		
				OpenMSP430_fpga.v	*FPGA top level file*
				driver_7segment.v	*Four-Digit, Seven-Segment LED Display driver*
				io_mux.v	*I/O mux for port function selection.*
				ext_de1_sram.v	*Interface with altera DE1's external async SRAM (256kwords x 16bits)*
				ram16x512.v	*Single port RAM generated with the megafunction wizard*

				rom16x2048.v	*Single port ROM generated with the megafunction wizard*
				openmsp430	***Local copy of the openMSP430 core.*** *The *define.v file has been adjusted to the requirements of the project.*
		sim			***Top level simulations directory***
			rtl_sim		***RTL simulations***
				bin	***RTL simulation scripts***
				msp430sim	*Main simulation script*
				ihex2mem.tcl	*Verilog ROM memory file generation*
				rtlsim.sh	*Verilog Icarus simulation script*
				run	***For running RTL simulations***
				run	*Run simulation of a given software project*
				run_disassemble	*Disassemble ROM content of the latest simulation*
				src	***RTL simulation verilog stimulus***
				submit.f	*Verilog simulator command file*
				*.v	*Stimulus vector for the corresponding software project*
		software			***Software C programs to be loaded in ROM***
			bin		*Specific binaries required for software development.*
				mifwrite.cpp	*This prog is taken from http://www.johnloomis.org/ece595c/notes/isa/mifwrite.html and slightly changed to satisfy quartus6.1 *.mif eating engine.*
				mifwrite.exe	*Windows executable.*
				mifwrite	*Linux executable.*
			memledtest		*LEDs blinking application (from the CDK4MSP project)*
		synthesis			***Top level synthesis directory***
			altera		
				main.qsf	*Global Assignments file*

				main.sof	*SOF file*
				OpenMSP430_fpga.qpf	*Quartus II project file*
				openMSP430_fpga_top.v	*RTL file list to be synthesized*

4. Directory structure: Software Development Tools

tools				***openMSP430 Software Development Tools top level directory***
	bin			***Contains the executable files***
		openmsp430-loader.tcl		*Simple command line boot loader: TCL Script*
		openmsp430-loader.exe		*Simple command line boot loader: Windows executable*
		openmsp430-minidebug.tcl		*Minimalistic debugger with simple GUI: TCL Script*
		openmsp430-minidebug.exe		*Minimalistic debugger with simple GUI: Windows executable*
		openmsp430-gdbproxy.tcl		*GDB Proxy server to be used together with MSP430-GDB and the Eclipse, DDD, or Insight graphical front-ends: TCL Script*
		openmsp430-gdbproxy.exe		*GDB Proxy server to be used together with MSP430-GDB and the Eclipse, DDD, or Insight graphical front-ends: Windows executable*
	lib			***Common library***
		tcl-lib		***Common TCL library***
			dbg_uart.tcl	*Low level UART communication functions*
			dbg_functions.tcl	*Main utility functions for the openMSP430 serial debug interface*
			combobox.tcl	*A combobox listbox widget written in pure tcl (from Bryan Oakley)*
	openmsp430-gdbproxy			***GDB Proxy server main project directory***
		openmsp430-gdbproxy.tcl		*GDB Proxy server main TCL Script (symbolic link with the script in the **bin***

				directory)
		server.tcl		*TCP/IP Server utility functions. Send/Receive RSP packets from GDB.*
		commands.tcl		*RSP command execution functions.*
		doc		***Some documentation regarding GDB and the RSP protocol.***
			ew_GDB_RSP.pdf	*Document from Bill Gatliff: Embedding with GNU: the gdb Remote Serial Protocol*
			Howto-GDB_Remote_Serial_Protocol.pdf	*Document from Jeremy Bennett (Embecosm): Howto: GDB Remote Serial Protocol - Writing a RSP Server*
	freewrap642			***The freeWrap program turns TCL/TK scripts into single-file binary executable programs for Windows.***
		freewrap.exe		*freeWrap executable to run on TCL/TK scripts (i.e. with GUI)*
		freewrapTCLSH.exe		*freeWrap executable to run on pure TCL scripts (i.e. command line)*
		tclpip85s.dll		*freeWrap mandatory DLL*
		generate_exec.bat		*Simple Batch file for auto generation of the tools' windows executables*