www.opencores.org

openMSP430

an MSP430 clone....

Author: Olivier GIRARD
olgirard@gmail.com

Rev. 1.7
August 18, 2010

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.
1.3	December 29th, 2009	GIRARD	- Renamed the "rom_*" ports to "pmem_*". - Renamed the "ram_*" ports to "dmem_*". - Renamed the "ROM_AWIDTH" Verilog define to "PMEM_AWIDTH". - Renamed the "RAM_AWIDTH" Verilog define to "DMEM_AWIDTH". - Prefixed all the verilog sub-modules of the openMSP430 core with "omsp_". - Diverse minor updates
1.4	January 12th, 2010	GIRARD	- Added the "Integration and Connectivity"section.
1.5	March 7th, 2010	GIRARD	- Add Hardware multiplier info. - Added the "Area and Speed Analysis" section.
1.6	August 1st, 2010	GIRARD	- Update core configuration section. - Expand the CPU selection table for msp430-gcc.
1.7	August 18th, 2010	GIRARD	- Update CPU_ID description in the serial debug interface chapter..

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 (16x16 Hardware Multiplier, 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).

The following SVN command can be run from a console (or GUI):

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

To keep yourself informed about project updates, you can subscribe to the following RSS feed.

Features & Limitations

Features

- **Core:**
  - Full instruction set support.
  - All addressing modes are supported.
  - IRQ and NMI support.
  - Power saving modes functionality is supported.
  - Configurable memory size for both program and data.

    - Serial Debug Interface (Nexus class 3).
    - FPGA friendly (single clock domain, no clock gate).
    - Small size (Xilinx: 1650 LUTs / Altera: 1550 LEs / ASIC: 8k gates).

- **Peripherals:**
    - 16x16 Hardware Multiplier.
    - Basic Clock Module.
    - Watchdog.
    - Timer A.
    - GPIO (port 1 to 6).
    - Templates for 8 and 16 bit peripherals (under BSD license).

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
- TI: a list of available MSP430 Open Source projects out there on the web today.

Legal information

MSP430 is a trademark of Texas Instruments, Inc. This project is not affiliated in any way with Texas Instruments. All other product names are trademarks or registered trademarks of their respective owners.

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 instruction 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/or ROM sizes.

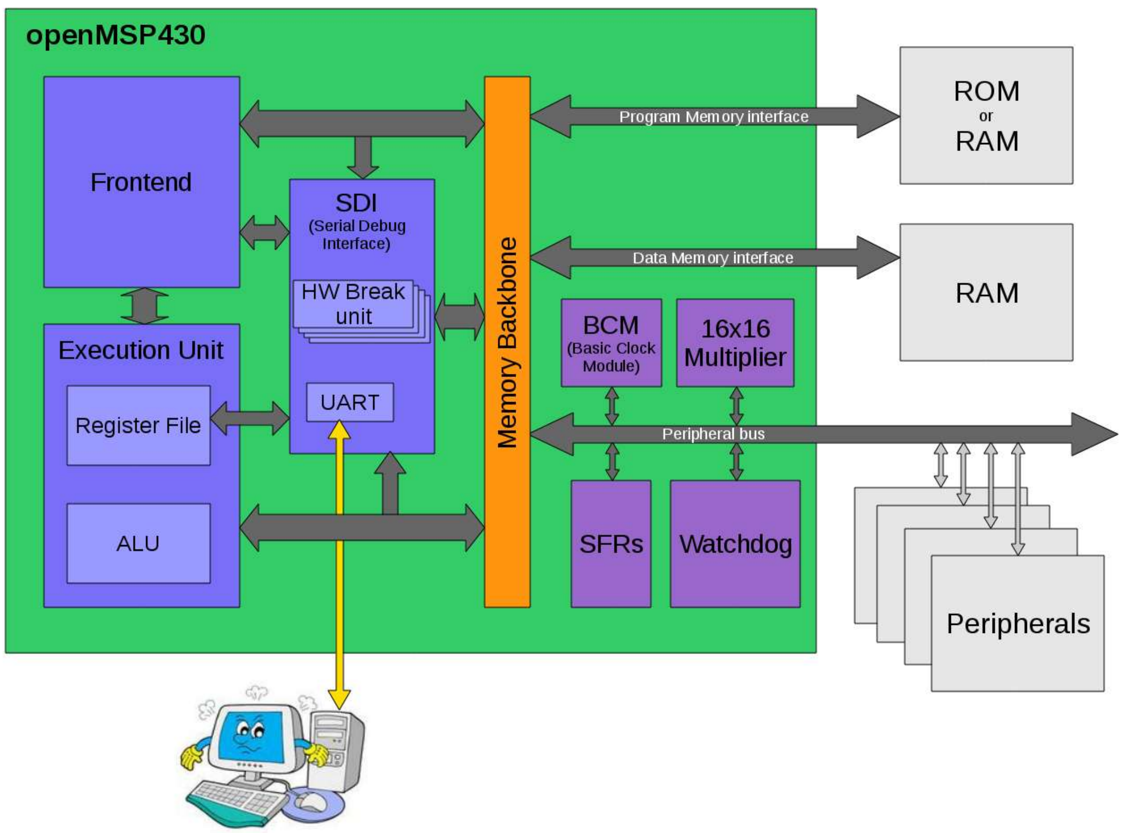
In addition to the CPU core itself, several peripherals are also provided and can be easily connected to the core during integration.

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 program, data and peripheral 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 sets of parameters can be adjusted by the user in order to define the program and data memory sizes:

```
// Program Memory Size:
//----------------------------
//`define PMEM_SIZE_59_KB
//`define PMEM_SIZE_55_KB
//`define PMEM_SIZE_54_KB
//`define PMEM_SIZE_51_KB
//`define PMEM_SIZE_48_KB
//`define PMEM_SIZE_41_KB
//`define PMEM_SIZE_32_KB
//`define PMEM_SIZE_24_KB
//`define PMEM_SIZE_16_KB
//`define PMEM_SIZE_12_KB
//`define PMEM_SIZE_8_KB
//`define PMEM_SIZE_4_KB
`define PMEM_SIZE_2_KB
//`define PMEM_SIZE_1_KB

// Data Memory Size:
//---------------------------
//`define DMEM_SIZE_32_KB
//`define DMEM_SIZE_24_KB
//`define DMEM_SIZE_16_KB
//`define DMEM_SIZE_10_KB
```

```
//`define DMEM_SIZE_8_KB
//`define DMEM_SIZE_5_KB
//`define DMEM_SIZE_4_KB
//`define DMEM_SIZE_2p5_KB
//`define DMEM_SIZE_2_KB
//`define DMEM_SIZE_1_KB
//`define DMEM_SIZE_512_B
//`define DMEM_SIZE_256_B
`define DMEM_SIZE_128_B
```

Note: The sum of both program and data memories **SHOULD NOT** exceed 63.5 kB.

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
//
`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
```

Note: Since the hardware breakpoint units are relatively big, it is recommended to include as many as you plan to use. These units are particularly useful if your instruction memory is a ROM (i.e. when you can't use software breakpoints) or if you want to be able to stop the CPU whenever some particular data addresses are accessed.

At last, this parameter controls if the hardware multiplier is included or not.

```
// Include/Exclude Hardware Multiplier
`define MULTIPLIER
```

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)
Program Memory interface			
pmem_addr	Output	`PMEM_AWIDTH[1]	Program Memory address
pmem_cen	Output	1	Program Memory chip enable (low active)
pmem_din	Output	16	Program Memory data input (optional[2])
pmem_dout	Input	16	Program Memory data output
pmem_wen	Output	2	Program Memory write enable (low active) (optional[2])
Data Memory interface			
dmem_addr	Output	`DMEM_AWIDTH[1]	Data Memory address
dmem_cen	Output	1	Data Memory chip enable (low active)
dmem_din	Output	16	Data Memory data input
dmem_dout	Input	16	Data Memory data output
dmem_wen	Output	2	Data Memory write enable (low active)
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)
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]: These two optional ports can be connected whenever the program memory is a RAM. This will allow the user to load a program through the serial debug interface and to use software breakpoints.

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.

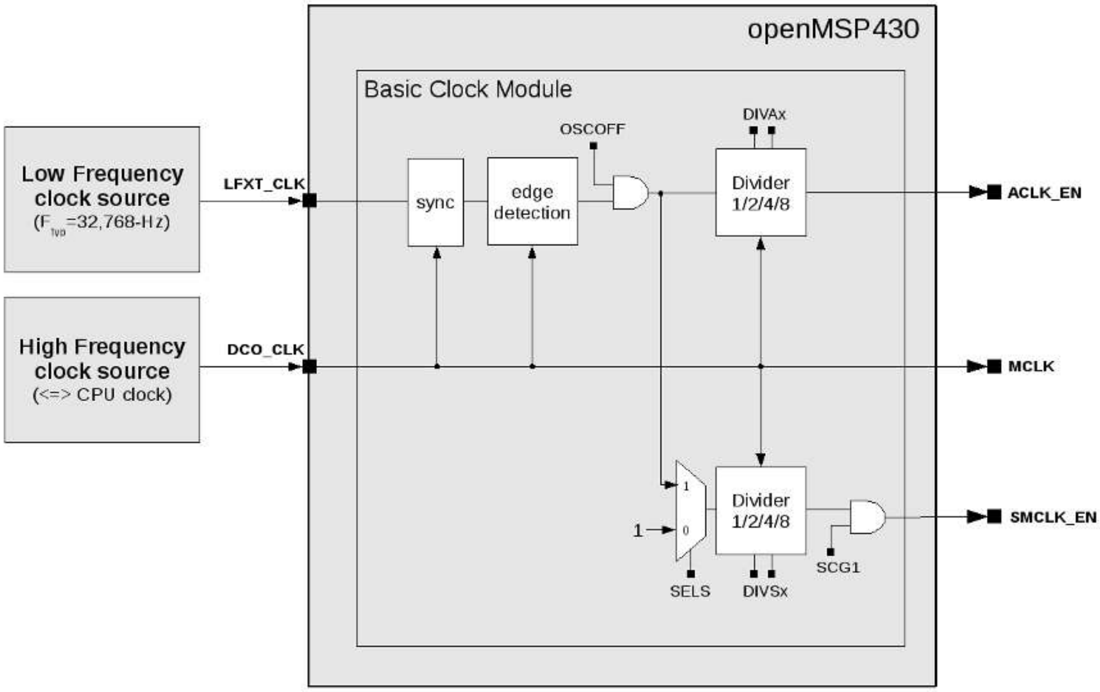
2.2 Peripherals

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

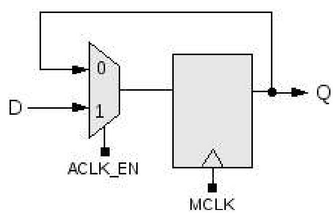
In order to make an FPGA implementation as simple as possible (ideally, a non-designer should be able to do it), clock gates are not used in the design and neither are clock muxes.

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.

2.2.5 16x16 Hardware Multiplier

100% of the features advertised in the MSP430x1xx Family User's Guide (Chapter 7) have been implemented.

The following parameter in the *openMSP430_defines.v* file controls if the hardware multiplier should be included or not.

```
// Include/Exclude Hardware Multiplier
`define MULTIPLIER
```

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	DMEM_AWIDTH															
CPU_ID_HI	0x01	PMEM_AWIDTH															
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 program and data memory size information of the implemented openMSP430. .

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	DMEM_AWIDTH															
CPU_ID_HI	0x01	PMEM_AWIDTH															

- **PMEM_AWIDTH** : Program memory size in byte for the current implementation.
- **DMEM_AWIDTH** : Data memory size in byte for the current implementation.

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)

  Note: range mode is not supported by the core unless the `HWBRK_RANGE define is set to 1'b1 in the *openMSP430_define.v* file.

- **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).

  Note: range mode is not supported by the core unless the `HWBRK_RANGE define is set to 1'b1 in the *openMSP430_define.v* file.
- **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.

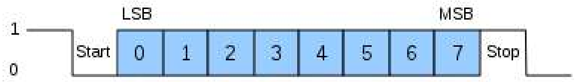
3. Debug Communication Interface: UART

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):

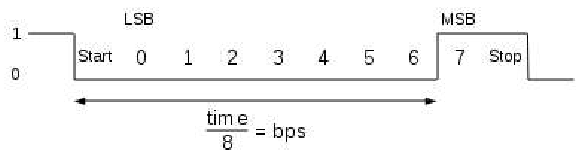


As you can see in the above diagram, data transmission starts with a Start bit, followed by the data bits (LSB sent first and MSB sent last), and ends with a "Stop" bit.

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.

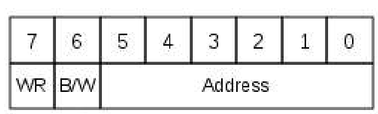
3.3 Read/Write access to the debug registers

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

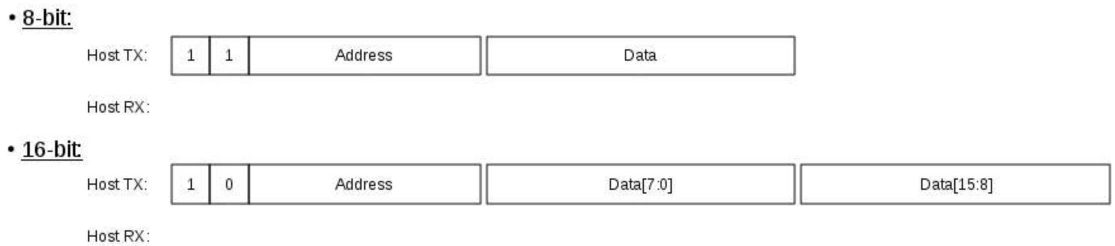
The command frame looks as following:



- **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.

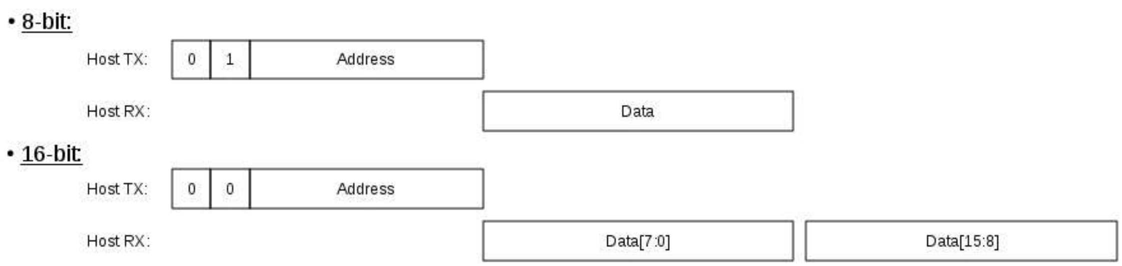
3.3.2 Write access

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.

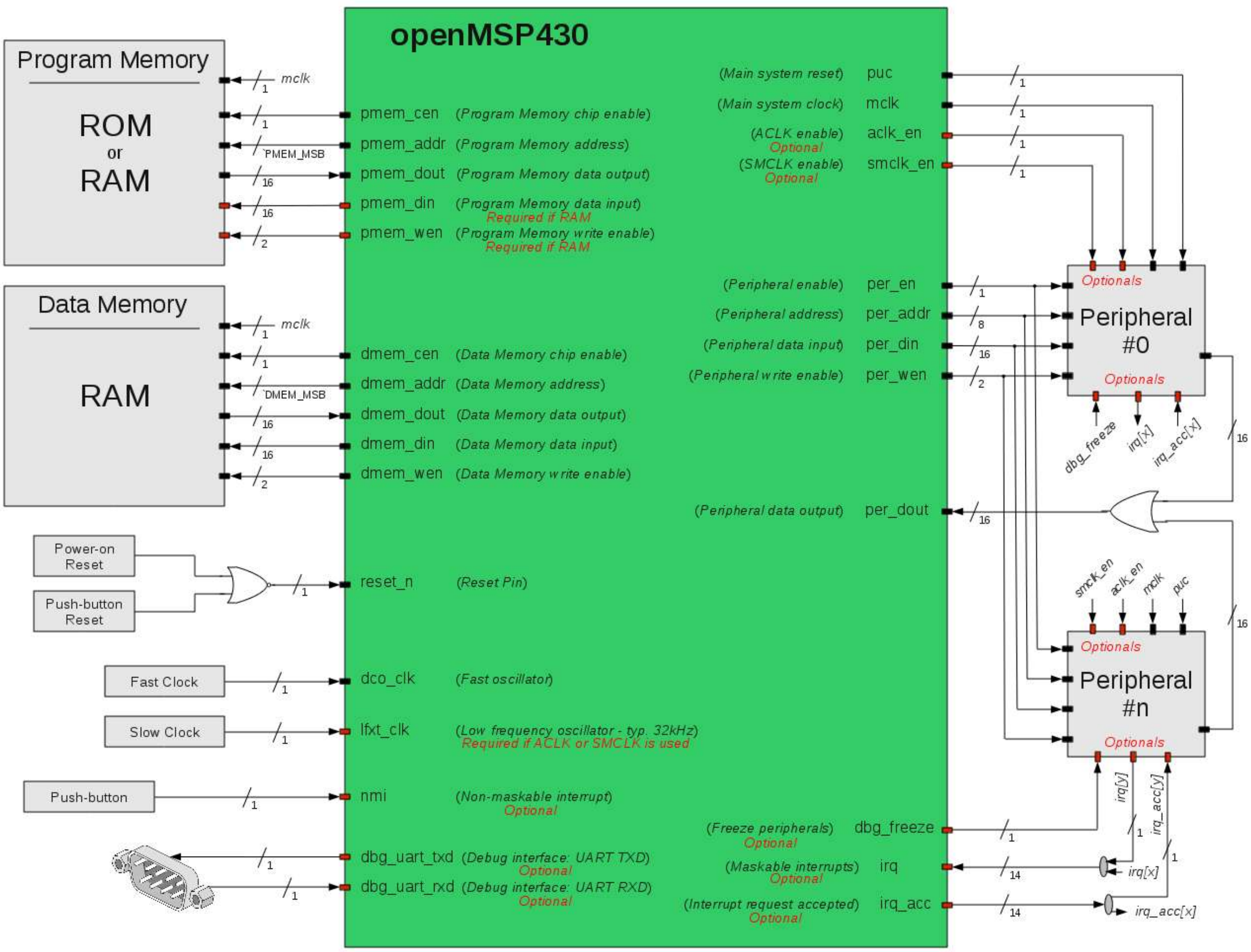
Integration and Connectivity

Table of content

1. Overview

This chapter aims to give a comprehensive description of all openMSP430 core interfaces in order to facilitates its integration within an ASIC or FPGA.

The following diagram shows an overview of the openMSP430 core connectivity:



The full pinout of the core is summarized in the following table.

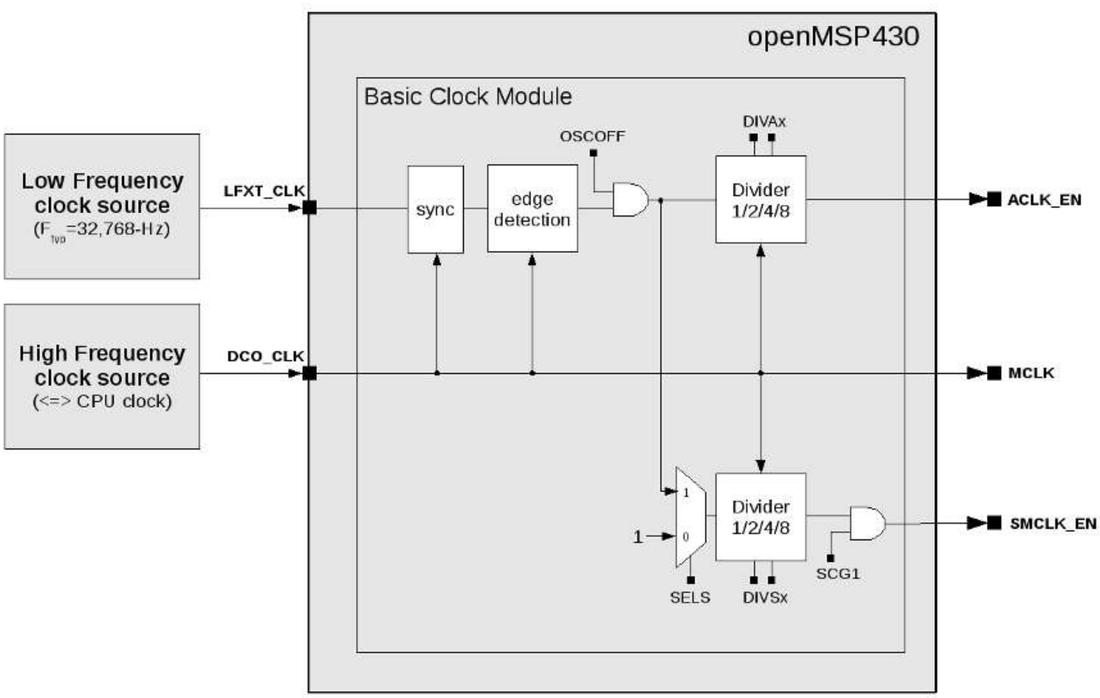
Port Name	**Direction**	**Width**	**Description**
Clocks			
dco_clk	Input	1	Fast oscillator (fast clock), CPU clock
lfxt_clk	Input	1	Low frequency oscillator (typ. 32kHz)
mclk	Output	1	Main system clock
aclk_en	Output	1	ACLK enable
smclk_en	Output	1	SMCLK enable
Resets			
puc	Output	1	Main system reset
reset_n	Input	1	Reset Pin (low active)
Program Memory interface			
pmem_addr	Output	`PMEM_AWIDTH[1]	Program Memory address
pmem_cen	Output	1	Program Memory chip enable (low
pmem_din	Output	16	Program Memory data input
pmem_dout	Input	16	Program Memory data output
pmem_wen	Output	2	Program Memory write enable (low
Data Memory interface			
dmem_addr	Output	`DMEM_AWIDTH[1]	Data Memory address
dmem_cen	Output	1	Data Memory chip enable (low active)
dmem_din	Output	16	Data Memory data input
dmem_dout	Input	16	Data Memory data output
dmem_wen	Output	2	Data Memory write enable (low active)
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)
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
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

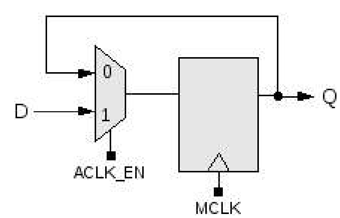
[1]: This parameter is declared in the "openMSP430_defines.v" file and defines the RAM/ROM size.

2. Clocks

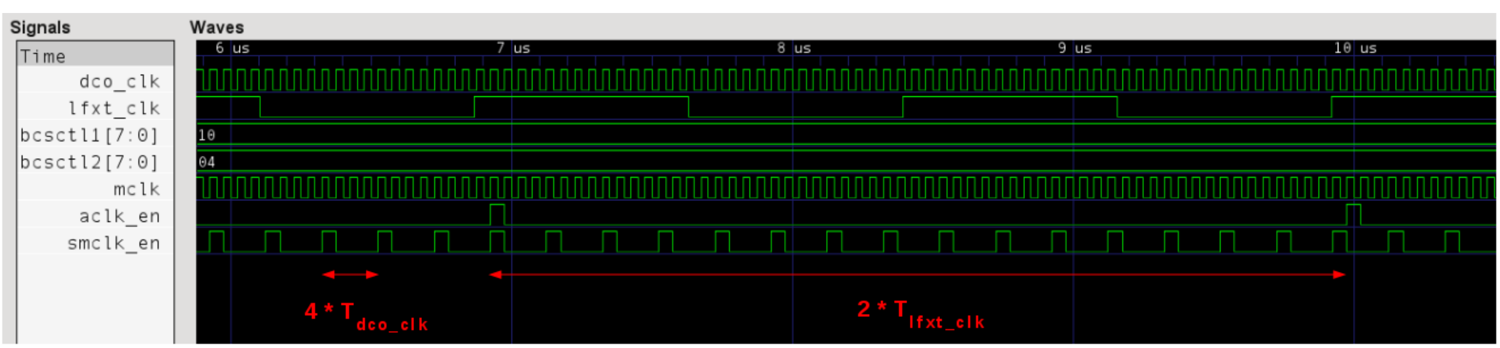
The different clocks in the design are managed by the Basic Clock Module:



- **DCO_CLK**: this input port is typically connected to a PLL, RC oscillator or any clock resource the target FPGA might provide.
  From a synthesis tool perspective (ISE, Quartus, Libero, Design Compiler...), this the only port where a clock needs to be declared.
- **LFXT_CLK**: if ACLK_EN or SMCLK_EN are going to be used in the project (for example through the Watchdog or TimerA peripherals), then this port needs to be connected to a clock running at least two time slower as DCO_CLK (typically 32kHz). It can be connected to 0 or 1 otherwise.
- **MCLK**: the main system clock drives the complete openMSP430 clock domain, including program/data memories and the peripheral interfaces.
- **ACLK_EN / SMCLK_EN**: these two clock enable signals can be used in order to emulate the original ACLK and SMCLK from the MSP430 specification.
  An example of this can be found in the Watchdog and TimerA modules, where it is implemented as following:



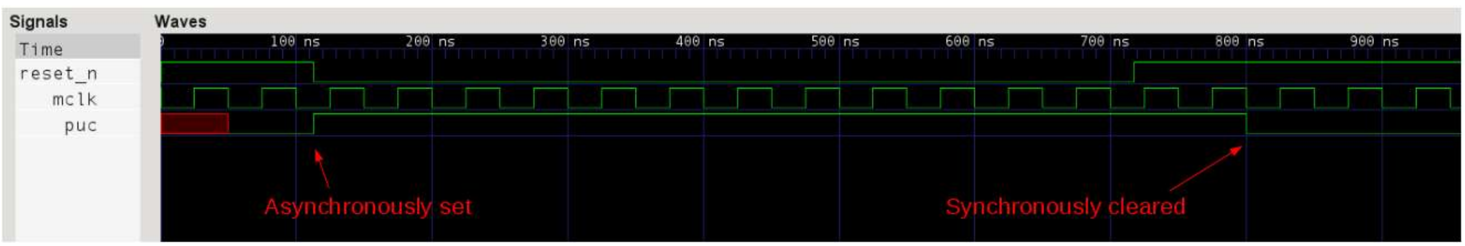
As an illustration, the following waveform shows the different clocks where the software running on the openMSP430 configures the BCSCTL1 and BCSCTL2 registers so that *ACLK_EN* and *SMCLK_EN* are respectively running at *LFXT_CLK/2* and *DCO_CLK/4*.



3. Resets

- **RESET_N**: this input port is typically connected to a board push button and is generally combined with the system power-on-reset.
- **PUC**: the Power-Up-Clear signal is asynchronously set with the reset pin (*RESET_N*), the watchdog reset or the serial debug interface reset. In order to get clean timings, it is synchronously cleared with MCLK's falling edge. As a general rule, this signal should be used as the reset of the *MCLK* clock domain.

The following waveform illustrates this:



4. Program Memory

Depending on the project needs, the program memory can be either implemented as a ROM or RAM.

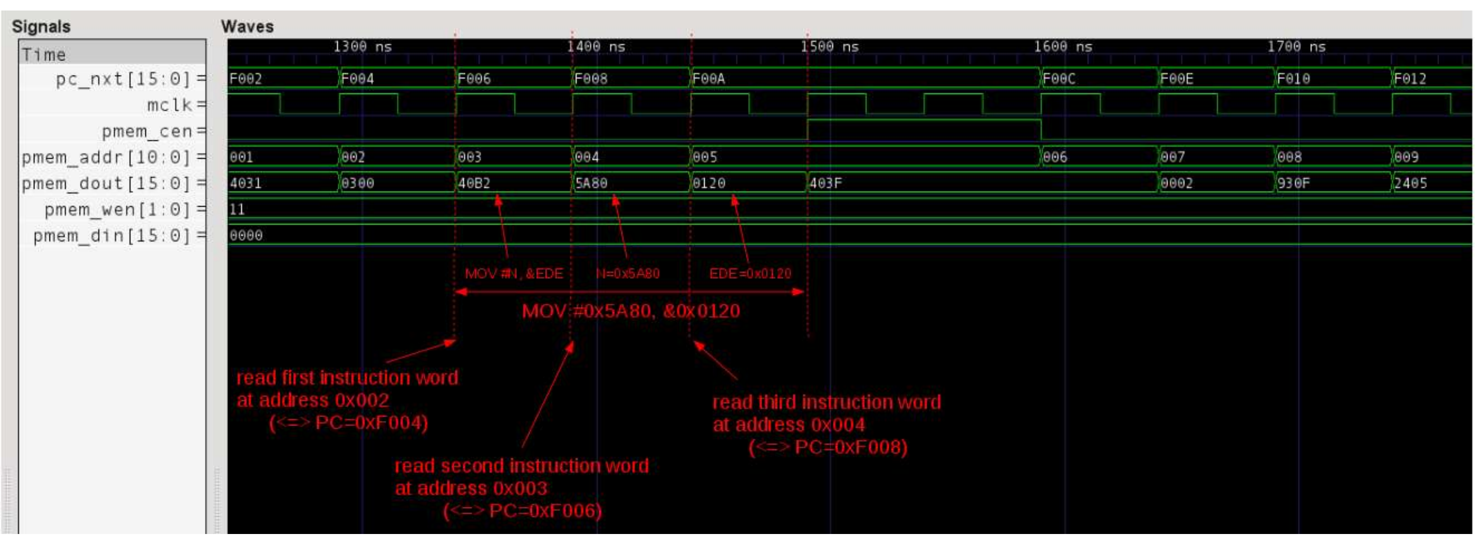
If a ROM is selected then the *PMEM_DIN* and *PMEM_WEN* ports won't be connected. In that case, the software debug capabilities are limited because the serial debug interface can only use hardware breakpoints in order to stop the program execution. In addition, updating the software will require a reprogramming of the FPGA.

If the program memory is a RAM, the developer gets full flexibility regarding software debugging. The serial debug interface can be used to update the program memory and software breakpoints can be used.

That said, the protocol between the openMSP430 and the program memory is quite standard. Signal description goes as following:

- **PMEM_CEN**: when this signal is active, the read/write access will be executed with the next *MCLK* rising edge. Note that this signal is LOW ACTIVE.
- **PMEM_ADDR**: Memory address of the 16 bit word which is going to be accessed.
  Note: in order to calculate the core logical address from the program memory physical address, the formula goes as following:
  *LOGICAL@=2*PHYSICAL@+0x10000-PMEM_SIZE*
- **PMEM_DOUT**: the memory output word will be updated with every valid read/write access (i.e. *PMEM_DOUT* is not updated if *PMEM_CEN*=1).
- **PMEM_WEN**: this signal selects which byte should be written during a valid access. PMEM_WEN[0] will activate a write on the lower byte, PMEM_WEN[1] a write on the upper byte. Note that these signals are LOW ACTIVE.
- **PMEM_DIN**: the memory input word will be written with the valid write access according to the *PMEM_WEN* value.

The following waveform illustrates some read accesses of the program memory (write access are illustrated in the data memory section):



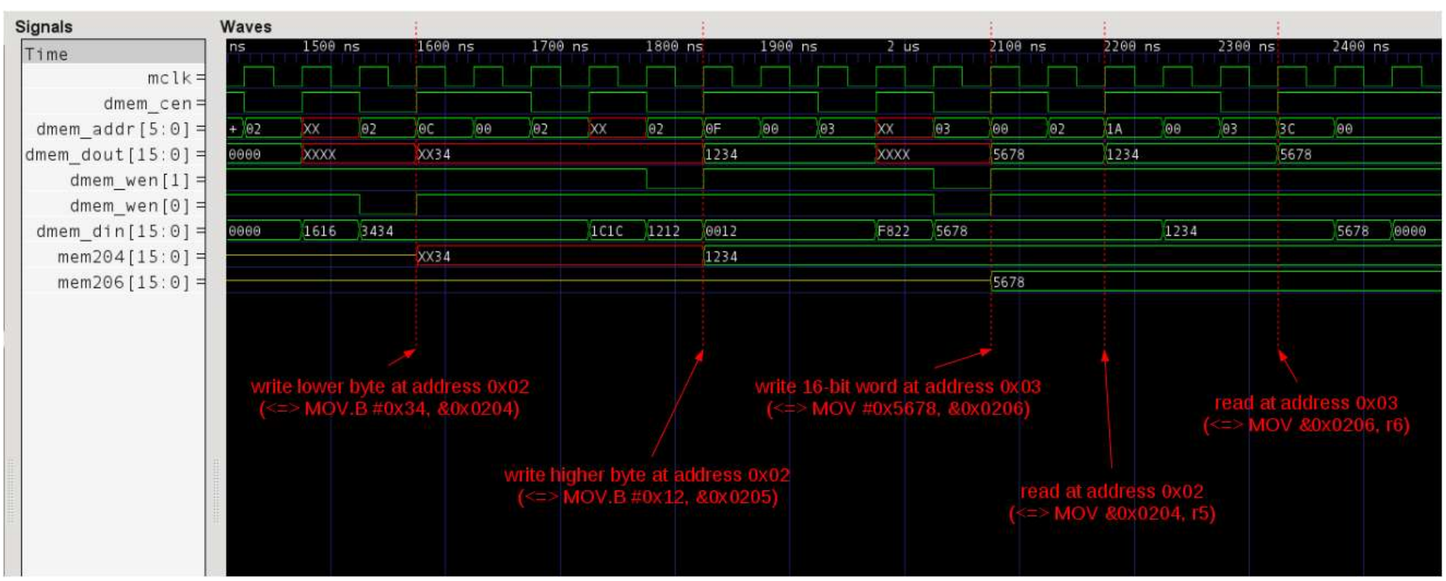
5. Data Memory

The data memory is always implemented as a RAM.

The protocol between the openMSP430 and the data memory is the same as the one of the program memory. Therefore, the signal description is the same:

- **DMEM_CEN**: when this signal is active, the read/write access will be executed with the next *MCLK* rising edge. Note that this signal is LOW ACTIVE.
- **DMEM_ADDR**: Memory address of the 16 bit word which is going to be accessed.
  Note: in order to calculate the core logical address from the data memory physical address, the formula goes as following: *LOGICAL@=2*PHYSICAL@+0x200*
- **DMEM_DOUT**: the memory output word will be updated with every valid read/write access (i.e. *DMEM_DOUT* is not updated if *DMEM_CEN*=1).
- **DMEM_WEN**: this signal selects which byte should be written during a valid access. DMEM_WEN[0] will activate a write on the lower byte, DMEM_WEN[1] a write on the upper byte. Note that these signals are LOW ACTIVE.
- **DMEM_DIN**: the memory input word will be written with the valid write access according to the *DMEM_WEN* value.

The following waveform illustrates some read/write access to the data memory:



6. Peripherals

The protocol between the openMSP430 core and its peripherals is the exactly same as the one with the data and program memories in regards to write access and differs slightly for read access.

On the connectivity side, the specificity is that the read data bus of all peripherals should be ORed together before being connected to the core, as showed in the diagram of the Overview section.

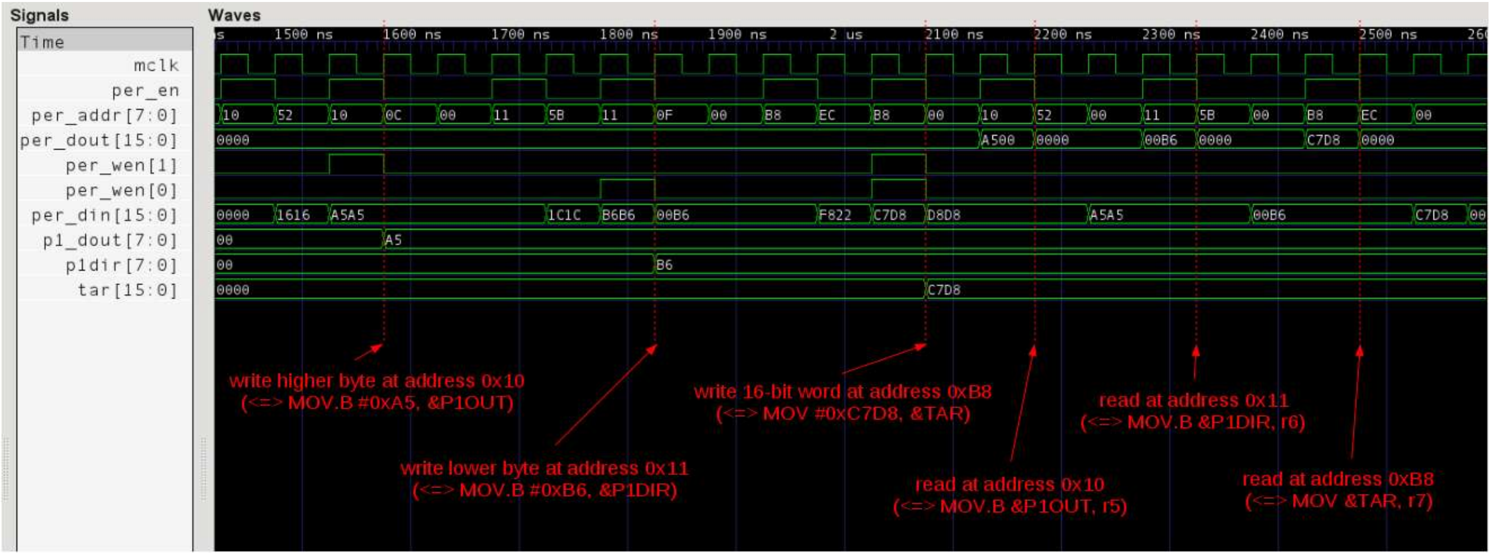
From the logical point of view, during a read access, each peripheral outputs the combinatorial value of its read mux and returns 0 if it doesn't contain the addressed register. On the waveforms, this translates by seeing the register value on *PER_DOUT* while *PER_EN* is valid and not one clock cycle afterwards as it is the case with the program and data memories.

In any case, it is recommended to use the templates provided with the core in order to develop your own custom peripherals.

The signal description therefore goes as following:

- **PER_EN**: when this signal is active, read access are executed during the current *MCLK* cycle while write access will be executed with the next *MCLK* rising edge. Note that this signal is HIGH ACTIVE.
- **PER_ADDR**: peripheral register address of the 16 bit word which is going to be accessed.
  Note: in order to calculate the core logical address from the peripheral register physical address, the formula goes as following: *LOGICAL@=2*PHYSICAL@*
- **PER_DOUT**: the peripheral output word will be updated with every valid read/write access, it will be set to 0 otherwise.
- **PER_WE**: this signal selects which byte should be written during a valid access. PER_WEN[0] will activate a write on the lower byte, PER_WEN[1] a write on the upper byte. Note that these signals are HIGH ACTIVE.
- **PER_DIN**: the peripheral input word will be written with the valid write access according to the *PER_WEN* value.

The following waveform illustrates some read/write access to the peripheral registers:



7. Interrupts

As with the original MSP430, the interrupt priorities of the openMSP430 are fixed in hardware accordingly to the connectivity of the *NMI* and *IRQ* ports. If two interrupts are pending simultaneously, the higher priority interrupt will be serviced first.
The following table summarize this:

Interrupt Port	Vector address	Priority
RESET_N	0xFFFE	15 (highest)
NMI	0xFFFC	14
IRQ[13]	0xFFFA	13
IRQ[12]	0xFFF8	12
IRQ[11]	0xFFF6	11
IRQ[10]	0xFFF4	10
IRQ[9]	0xFFF2	9
IRQ[8]	0xFFF0	8
IRQ[7]	0xFFEE	7

IRQ[6]	0xFFEC	6
IRQ[5]	0xFFEA	5
IRQ[4]	0xFFE8	4
IRQ[3]	0xFFE6	3
IRQ[2]	0xFFE4	2
IRQ[1]	0xFFE2	1
IRQ[0]	0xFFE0	0 (lowest)

The signal description goes as following:

- **NMI**: The **N**on-**M**askable **I**nterrupt has higher priority than other IRQs and is masked by the NMIIE bit instead of GIE.
  It is internally synchronized to the *MCLK* domain and can therefore be connected to any asynchronous signal of the chip (which could for example be a pin of the FPGA). If unused, this signal should be connected to 0.
- **IRQ**: The standard interrupts can be connected to any signal coming from the *MCLK* domain (typically a peripheral). Priorities can be chosen by selecting the proper bit of the *IRQ* bus as shown in the table above. Unused interrupts should be connected to 0.
  Note: *IRQ[10]* is internally connected to the Watchdog interrupt. If this bit is also used by an external peripheral, they will both share the same interrupt vector.
- **IRQ_ACC**: Whenever an interrupt request is serviced, some peripheral automatically clear their pending flag in hardware. In order to do so, the *IRQ_ACC* bus can be used by using the bit matching the corresponding *IRQ* bit. An example of this is shown in the implementation of the TACCR0 Timer A interrupt.

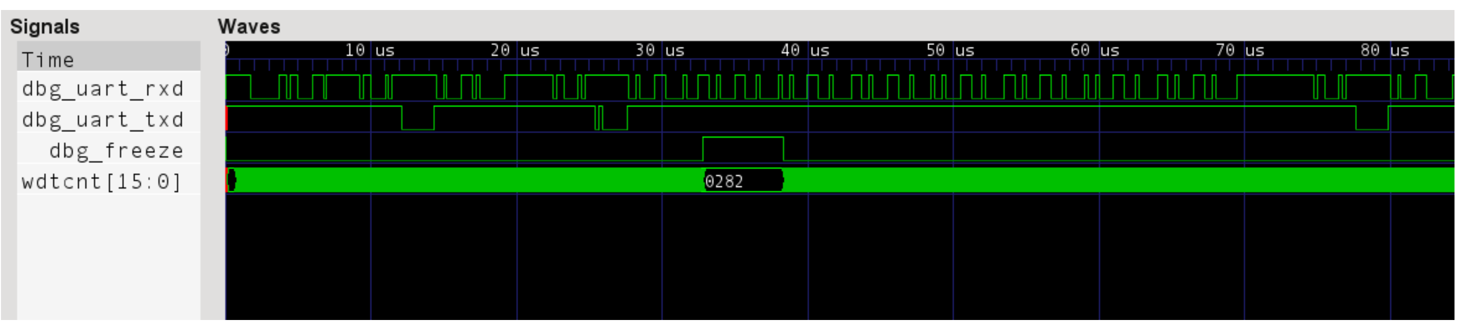
The following waveform illustrates a TAIV interrupt issued by the Timer-A, which is connected to *IRQ[8]*:

8. Serial Debug Interface

The serial debug interface module provides a two-wires communication bus for remote debugging and an additional freeze signal which might be useful for some peripherals.

- **DBG_FREEZE**: this signal will be set whenever the debug interface stops the CPU (and if the *FRZ_BRK_EN* field of the CPU_CTL debug register is set). As its name implies, the purpose of *DBG_FREEZE* is to freeze a peripheral whenever the CPU is stopped by the software debugger.
For example, it is used by the Watchdog timer in order to stop its free-running counter. This prevents the CPU from being reseted by the watchdog every times the user stops the CPU during a debugging session.
- **DBG_UART_TXD** / **DBG_UART_RXD**: these signals are typically connected to an RS-232 transceiver and will allow a PC to communicate with the openMSP430 core.

The following waveform shows some communication traffic on the serial bus :



5

Area and Speed Analysis

Table of content

Notice: the results presented here might vary depending on the tool versions, applied timing constraints and exact configuration of the openMSP430 core. The FPGA results were obtained using the free tool versions provided by the vendors (i.e ISE 11.1, QuartusII 9.1 & Libero 8.5). The ASIC synthesis was run with Synopsys Design Compiler 2007.12 (without dc_ultra or any special feature).

1. Overview

1.1 FPGAs

			Utilization			
Manu-facturer	Devices	Info	Basic Config. (Core + Watchdog)	Hardware Multiplier	With debug interface (Software breakpoints)	Additional Hardware breakpoint unit
Xilinx	Spartan 3 Spartan 3E Spartan 3A Spartan 3A DSP Virtex 4	4-inputs LUTs	1 620	+ 200	+ 520	+ 80
	Spartan 6 Virtex 5 Virtex 6	6-inputs LUTs	1 240	+ 150	+ 350	+ 70
Altera	Cyclone II Cyclone III Cyclone IV GX Stratix	LEs	1 550	+ 210	+ 480	+ 110
	Arria GX Arria II GX Stratix II Stratix III	ALUTs	1 030	+ 115	+ 380	+ 90
Actel	ProASIC3E ProASIC3L ProASIC3 Fusion IGLOOe	Tiles	3 550	+ 1 060	+ 1 200	+ 220
-	-	Registers	470	+ 75	+ 140	+ 45

Speed (in MHz, min and max values across all speed grades)			
Manufacturer	**Devices**	**Basic Configuration (Core + Watchdog + HW Multiplier)**	**With debug interface**
Xilinx	**Spartan 3 Spartan 3E Spartan 3A Spartan 3A DSP**	**30 - 40**	**25 - 35**
	Spartan 6	**40 - 65**	**35 - 60**
	Virtex 4	**50 - 70**	**45 - 60**
	Virtex 5	**75 - 100**	**65 - 85**
	Virtex 6	**90 - 115**	**75 - 100**
Altera	**Cyclone II**	**35 - 45**	**30 - 45**
	Cyclone III Cyclone IV GX	**40 - 55**	**35 - 50**
	Arria II GX	**65 - 85**	**60 - 80**
	Stratix II	**55 - 75**	**50 - 65**
	Stratix III	**75 - 95**	**70 - 90**
Actel	**ProASIC3E ProASIC3L ProASIC3 Fusion IGLOOe**	**15 - 25**	**15 - 25**

1.2 ASICs

			Area			
Process	Target Frequency	Info	Basic Config. (Core + Watchdog)	Hardware Multiplier	With debug interface (Software breakpoints)	Additional Hardware breakpoint unit
180 nm	50 MHz	kGates	8	+ 2.5	+ 2	+ 0.8
	100 MHz	kGates	10	+ 4.4	+ 2	+ 1.2

2. Detailed results

2.1 FPGAs

2.1.1 Xilinx

FPGA Device	Speed Grade	Info	openMSP430 Configuration						
			No Debug	No Debug with HW multiplier	With debug interface (no HW breakpoints)	With debug interface (# hardware breakpoint units)			
						1	2	3	4
Spartan 3	-4	4-LUTs	1 609	1 811	2 125	2 165	2 272	2 366	2 458
		Registers	458	533	594	637	679	721	763
		Speed (MHz)	30.06	30.05	28.29	22.26	25.59	25.19	23.50
	-5	4-LUTs	1 609	1 811	2 127	2 166	2 276	2 367	2 459
		Registers	458	533	594	637	679	721	763
		Speed (MHz)	32.69	34.11	32.63	27.09	28.90	27.68	27.63
Spartan 3E	-4	4-LUTs	1 615	1 816	2 131	2 185	2 298	2 383	2 474
		Registers	458	533	594	637	679	721	763
		Speed (MHz)	32.03	32.09	28.00	27.26	27.23	27.36	24.80
	-5	4-LUTs	1 615	1 816	2 131	2 184	2 295	2 383	2 474
		Registers	458	533	594	637	679	721	763
		Speed (MHz)	37.27	37.71	32.51	32.34	29.51	30.26	29.29
Spartan 3A	-4	4-LUTs	1 629	1 832	2 139	2 191	2 304	2 396	2 489
		Registers	459	534	595	638	680	722	764
		Speed (MHz)	31.23	31.05	29.08	26.45	26.39	24.87	24.47
	-5	4-LUTs	1 622	1 827	2 138	2 187	2 302	2 388	2 483
		Registers	459	534	595	638	680	722	764
		Speed (MHz)	36.03	36.14	33.50	30.65	30.79	29.70	27.70

Spartan 3A DSP	-4	4-LUTs	1 628	1 831	2 140	2 197	2 310	2 402	2 497
		Registers	459	534	595	638	680	722	764
		Speed (MHz)	31.18	31.26	29.51	26.61	26.48	24.30	24.56
	-5	4-LUTs	1 621	1 826	2 136	2 196	2 312	2 400	2 495
		Registers	459	534	595	638	680	722	764
		Speed (MHz)	39.31	37.59	33.59	32.61	33.00	28.86	28.49
Spartan 6	-2	6-LUTs	1 277	1 436	1 620	1 705	1 774	1 851	1 905
		Registers	459	533	595	638	680	722	764
		Speed (MHz)	41.06	41.03	39.09	34.88	34.40	26.89	33.86
	-3	6-LUTs	1 271	1 425	1 603	1 685	1 753	1 829	1 876
		Registers	459	533	595	638	680	722	764
		Speed (MHz)	58.19	58.21	50.01	46.45	45.84	41.76	43.33
	-4	6-LUTs	1 267	1 424	1 603	1 681	1 750	1 828	1 873
		Registers	459	533	595	638	680	722	764
		Speed (MHz)	64.96	67.62	57.38	51.11	50.07	42.97	43.61
Virtex 4	-10	4-LUTs	1 629	1 829	2 151	2 200	2 305	2 395	2 490
		Registers	459	534	595	638	680	722	764
		Speed (MHz)	50.12	51.17	45.61	43.60	41.67	42.34	39.18
	-11	4-LUTs	1 632	1 810	2 152	2 202	2 307	2 396	2 491
		Registers	459	534	595	638	680	722	764
		Speed (MHz)	57.27	56.14	53.79	48.82	48.59	48.70	47.39
	-12	4-LUTs	1 627	1 819	2 152	2 199	2 305	2 394	2 489
		Registers	459	534	595	638	680	722	764
		Speed (MHz)	66.56	64.59	57.46	59.27	50.78	54.40	53.87

Virtex 5	**-1**	**6-LUTs**	1 219	1 372	1 601	1 691	1 753	1 832	1 881
		Registers	458	532	594	637	679	721	763
		Speed (MHz)	74.39	74.69	70.25	63.09	59.89	53.00	53.53
	-2	**6-LUTs**	1 221	1 372	1 601	1 692	1 752	1 831	1 881
		Registers	458	532	594	637	679	721	763
		Speed (MHz)	82.16	82.12	77.42	69.06	64.99	58.40	70.06
	-3	**6-LUTs**	1 215	1 367	1 602	1 692	1 751	1 831	1 882
		Registers	458	532	594	637	679	721	763
		Speed (MHz)	97.85	97.25	85.67	72.33	76.99	71.20	69.05
Virtex 6	**-1**	**6-LUTs**	1 237	1 390	1 585	1 673	1 746	1 818	1 866
		Registers	458	532	594	637	679	721	763
		Speed (MHz)	89.32	92.59	87.64	68.59	77.03	73.56	66.47
	-2	**6-LUTs**	1 235	1 388	1 582	1 668	1 737	1 816	1 860
		Registers	458	532	594	637	679	721	763
		Speed (MHz)	102.40	97.51	98.05	86.74	79.95	77.19	74.94
	-3	**6-LUTs**	1 234	1 387	1 579	1 668	1 737	1 815	1 860
		Registers	458	532	594	637	679	721	763
		Speed (MHz)	111.74	115.71	102.04	88.93	89.90	84.47	90.80

2.1.2 Altera

FPGA Device	Speed Grade	Info	openMSP430 Configuration						
			No Debug	No Debug with HW multiplier	With debug interface (no HW breakpoints)	With debug interface (# hardware breakpoint units)			
						1	2	3	4
Cyclone II	-6	LEs	1 552	1 785	2 040	2 179	2 286	2 418	2 507
		Registers	467	537	610	653	695	737	779
		Speed (MHz)	45.10	47.32	42.79	43.81	41.57	42.10	40.71
	-7	LEs	1 556	1 781	2 049	2 191	2 298	2 414	2 508
		Registers	467	537	610	653	695	737	779
		Speed (MHz)	40.53	40.24	37.39	38.39	34.23	35.54	33.96
	-8	LEs	1 555	1 779	2 047	2 192	2 290	2 406	2 524
		Registers	467	537	610	653	695	737	779
		Speed (MHz)	33.07	32.97	32.00	30.62	29.78	29.63	26.38
Cyclone III	-6	LEs	1 539	1 752	2 021	2 148	2 251	2 357	2 450
		Registers	467	537	610	653	695	737	779
		Speed (MHz)	51.87	54.11	48.26	49.95	48.39	48.43	45.61
	-7	LEs	1 539	1 750	2 022	2 147	2 244	2 363	2 443
		Registers	467	537	610	653	695	737	779
		Speed (MHz)	46.25	43.88	44.28	41.64	39.18	40.59	40.86
	-8	LEs	1 542	1 752	2 020	2 158	2 243	2 380	2 448
		Registers	467	537	610	653	695	737	779
		Speed (MHz)	40.56	38.68	38.0	38.38	33.94	33.57	32.86
Cyclone IV GX	-6	LEs	1 541	1 750	2 024	2 148	2 246	2 364	2 459
		Registers	467	537	610	653	695	737	779

		Speed (MHz)	50.58	51.77	51.16	49.6	47.38	47.07	46.67
	-7	**LEs**	1 540	1 749	2 024	2 148	2 247	2 366	2 448
		Registers	467	537	610	653	695	737	779
		Speed (MHz)	47.09	44.19	44.43	42.63	42.49	41.6	39.03
	-8	**LEs**	1 544	1 747	2 020	2 147	2 244	2 363	2 444
		Registers	467	537	610	653	695	737	779
		Speed (MHz)	40.09	37.67	39.76	36.86	37.27	34.69	37.03
Arria GX	-6	**ALUTs**	1 044	1 160	1 414	1 525	1 588	1 675	1 765
		Registers	468	539	612	656	708	744	791
		Speed (MHz)	48.71	49.23	44.58	44.38	41.88	42.51	42.18
Arria II GX	-4	**ALUTs**	1 031	1 146	1 407	1 507	1 577	1 668	1 754
		Registers	469	540	611	654	706	749	793
		Speed (MHz)	84.37	83.22	78.81	75.19	75.75	76.3	79.81
	-5	**ALUTs**	1 025	1 148	1 404	1 503	1 600	1 670	1 742
		Registers	467	539	612	654	708	744	805
		Speed (MHz)	76.17	72.65	68.86	65.58	67.96	66.81	65.35
	-6	**ALUTs**	1 032	1 143	1 403	1 506	1 590	1 677	1 755
		Registers	469	539	611	659	704	753	793
		Speed (MHz)	62.63	61.59	59.66	57.2	55.76	59.04	57.41
Stratix	-5	**LEs**	1 525	1 730	1 989	2 081	2 185	2 279	2 378
		Registers	-	-	-	-	-	-	-
		Speed (MHz)	44.00	43.38	43.64	42.92	40.58	41.70	39.71
	-6	**LEs**	1 525	1 730	1 989	2 081	2 185	2 279	2 378
		Registers	-	-	-	-	-	-	-
		Speed (MHz)	39.88	40.74	39.82	37.18	37.42	36.97	36.81

	-7	**LEs**	1 525	1 730	1 989	2 081	2 185	2 279	2 378
		Registers	-	-	-	-	-	-	-
		Speed (MHz)	32.97	34.67	33.27	32.83	33.06	31.54	30.66
Stratix II	-3	**ALUTs**	1 040	1 145	1 422	1 523	1 590	1 665	1 753
		Registers	469	540	610	655	698	739	783
		Speed (MHz)	73.79	73.28	72.38	65.89	67.11	66.09	65.75
	-4	**ALUTs**	1 039	1 157	1 424	1 529	1 601	1 671	1 762
		Registers	469	540	613	658	699	741	781
		Speed (MHz)	63.75	63.29	60.31	58.10	56.84	59.57	59.26
	-5	**ALUTs**	1 039	1 155	1 419	1 527	1 592	1 678	1 763
		Registers	469	541	617	655	698	741	783
		Speed (MHz)	54.04	54.82	51.89	50.81	49.89	50.02	49.31
Stratix III	-2	**ALUTs**	1 029	1 147	1 408	1 511	1 597	1 666	1 748
		Registers	468	538	611	656	702	752	799
		Speed (MHz)	93.84	97.68	89.59	84.5	86.24	86.72	85.01
	-3	**ALUTs**	1 033	1 142	1 414	1 506	1 588	1 675	1 754
		Registers	469	539	610	656	699	753	807
		Speed (MHz)	83.68	80.16	75.77	71.9	76.64	73.49	75.35
	-4	**ALUTs**	1 030	1 147	1 411	1 505	1 587	1 670	1 760
		Registers	469	539	614	654	700	754	803
		Speed (MHz)	73.17	72.42	72.63	66.91	68.49	65.19	68.43

2.1.3 Actel

FPGA Device	Speed Grade	Info	openMSP430 Configuration						
			No Debug	No Debug with HW multiplier	With debug interface (no HW breakpoints)	With debug interface (# hardware breakpoint units)			
						1	2	3	4
ProASIC3E	Std	Tiles	3 585	4 734	4 884	5 014	5 263	5 571	5 747
		Registers	479	550	623	666	709	750	793
		Speed (MHz)	16.81	16.14	13.98	16.22	16.66	14.89	15.24
	-1	Tiles	3 635	4 585	4 742	5 004	5 246	5 345	5 713
		Registers	479	552	624	667	708	750	793
		Speed (MHz)	18.01	18.97	17.92	16.03	19.03	19.08	18.29
	-2	Tiles	3 556	4 573	4 811	5 002	5 210	5 446	5 625
		Registers	479	553	623	666	707	750	792
		Speed (MHz)	22.45	20.84	21.42	21.24	24.01	22.85	19.45
ProASIC3L	Std	Tiles	3 549	4 665	4 774	5 012	5 183	5 453	5 638
		Registers	480	552	623	667	709	750	792
		Speed (MHz)	14.31	14.27	15.14	14.42	14.74	14.15	14.05
	-1	Tiles	3 535	4 595	4 776	5 032	5 174	5 418	5 706
		Registers	479	551	623	666	708	750	793
		Speed (MHz)	18.13	17.31	15.90	18.34	17.14	17.69	16.27
ProASIC3	Std	Tiles	3 585	4 734	4 884	5 014	5 263	5 571	5 747
		Registers	479	550	623	666	709	750	793
		Speed (MHz)	16.47	15.62	15.03	16.55	16.00	14.63	15.38
	-1	Tiles	3 635	4 585	4 742	5 004	5 246	5 345	5 713
		Registers	479	552	624	667	708	750	793

		Speed (MHz)	18.03	19.21	18.39	18.40	18.95	17.13	18.59
	-2	**Tiles**	3 556	4 573	4 811	5 002	5 210	5 446	5 625
		Registers	479	553	623	666	707	750	792
		Speed (MHz)	22.80	21.97	21.67	21.24	22.57	23.27	20.75
IGLOOe	**Std**	**Tiles**	3 646	4 844	4 857	5 016	5 214	5 467	5 739
		Registers	479	552	623	666	709	751	791
		Speed (MHz)	14.01	14.51	13.61	13.85	14.29	14.44	14.10
Fusion	**Std**	**Tiles**	3 585	4 734	4 884	5 014	5 263	5 571	5 747
		Registers	479	550	623	666	709	750	793
		Speed (MHz)	16.65	15.84	14.25	15.60	15.62	15.20	15.50
	-1	**Tiles**	3 635	4 585	4 742	5 004	5 246	5 345	5 713
		Registers	479	552	624	667	708	750	793
		Speed (MHz)	17.90	18.46	17.79	17.86	17.81	18.69	17.98
	-2	**Tiles**	3 556	4 573	4 811	5 002	5 210	5 446	5 625
		Registers	479	553	623	666	707	750	792
		Speed (MHz)	22.30	21.34	20.58	20.27	21.48	21.39	20.59

2.2 ASICs

2.2.1 180nm

Target Frequency	Info	openMSP430 Configuration						
		No Debug	No Debug with HW multiplier	With debug interface (no HW breakpoints)	With debug interface (# hardware breakpoint units)			
					1	2	3	4
25 MHz	kgates	8 042	10 457	9 995	10 744	11 487	12 189	12 905
	µm²	80 256	104 352	99 742	107 223	114 637	121 643	128 784
	timing	clean	clean	clean	clean	clean	clean	clean
33 MHz	kgates	8 039	10 458	9 976	10 839	11 584	12 293	13 022
	µm²	80 226	104 365	99 552	108 164	115 602	122 677	129 956
	timing	clean	clean	clean	clean	clean	clean	clean
50 MHz	kgates	8 187	10 753	10 149	11 189	11 929	12 651	13 405
	µm²	81 703	107 305	101 285	111 660	119 048	126 253	133 778
	timing	clean	clean	clean	clean	clean	clean	clean
66 MHz	kgates	8 535	11 837	10 591	12 042	12 873	13 489	14 299
	µm²	85 172	118 130	105 693	120 176	128 465	134 606	142 692
	timing	clean	clean	clean	clean	clean	clean	clean
100 MHz	kgates	10 019	14 468	12 095	14 386	15 197	16 027	16 936
	µm²	99 988	144 382	120 698	143 560	151 660	159 936	169 014
	timing	clean	-0.98 ns	clean	clean	clean	clean	clean
125 MHz	kgates	11 851	16 142	13 838	16 502	17 209	17 660	18 718
	µm²	118 270	161 087	138 095	164 676	171 738	176 229	186 793
	timing	-0.75 ns	-2.85 ns	-0.62 ns	-1.46 ns	-1.66 ns	-1.81 ns	-1.81 ns

6.

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.

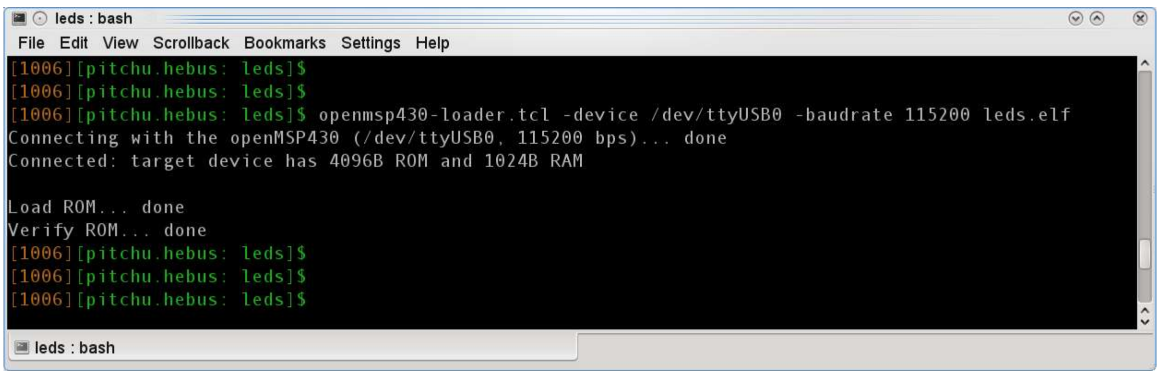
It is typically used in conjunction with '***make***' in order to automatically load the program after the compile step (see '***Makefile***' from software examples provided with the project's FPGA implementation).

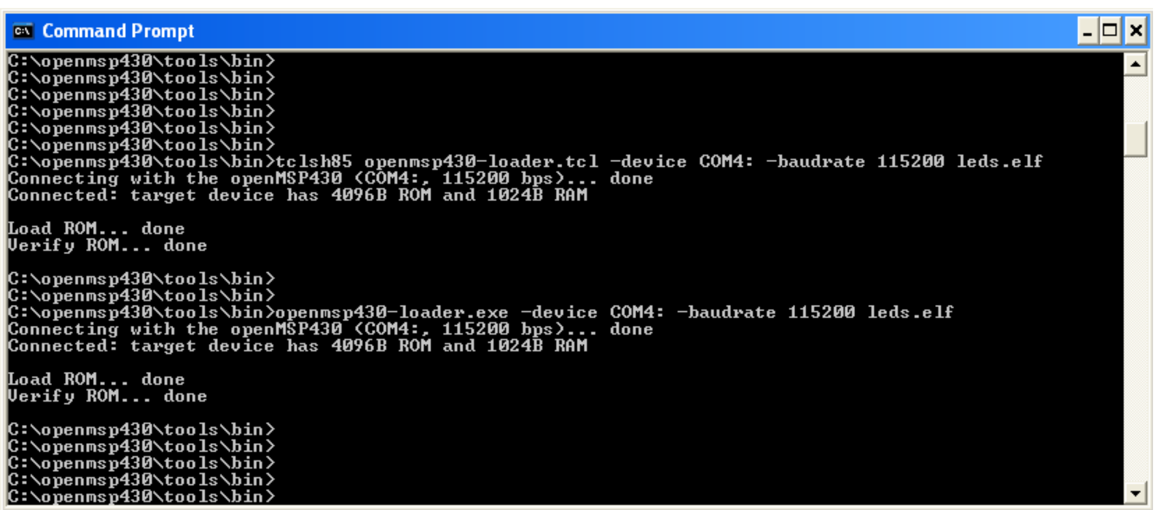
The program can be called with the following syntax:

```
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
```

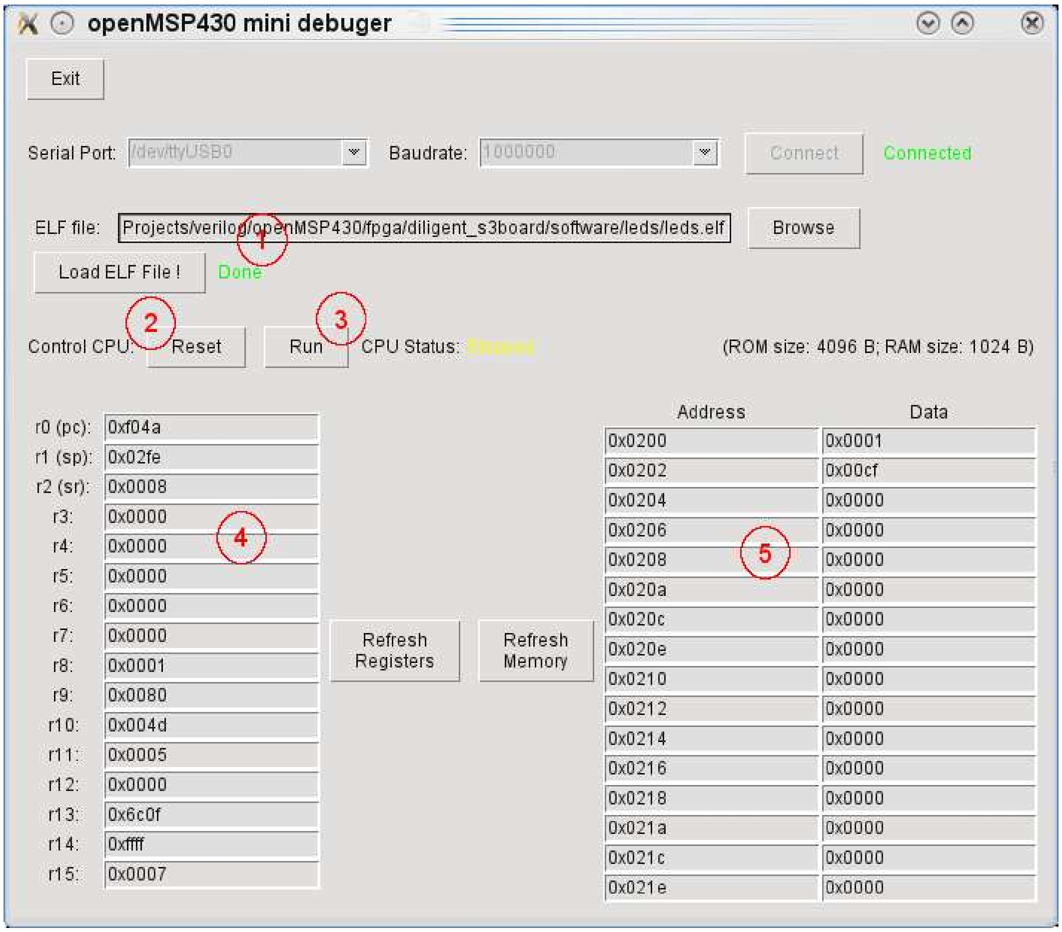
These screenshots show the script in action under Linux and Windows:





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:

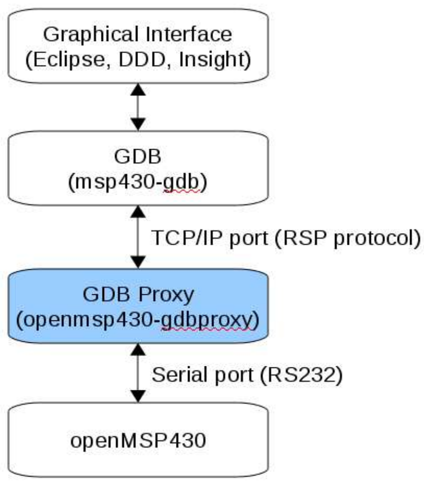
- **(1)** Load the program memory with an ELF file
- **(2)** Reset the CPU
- **(3)** Stop/Start the program execution
- **(4)** Read/Write access of the CPU registers
- **(5)** Read/Write access of the whole memory range (program, data, peripherals)

4. openmsp430-gdbproxy

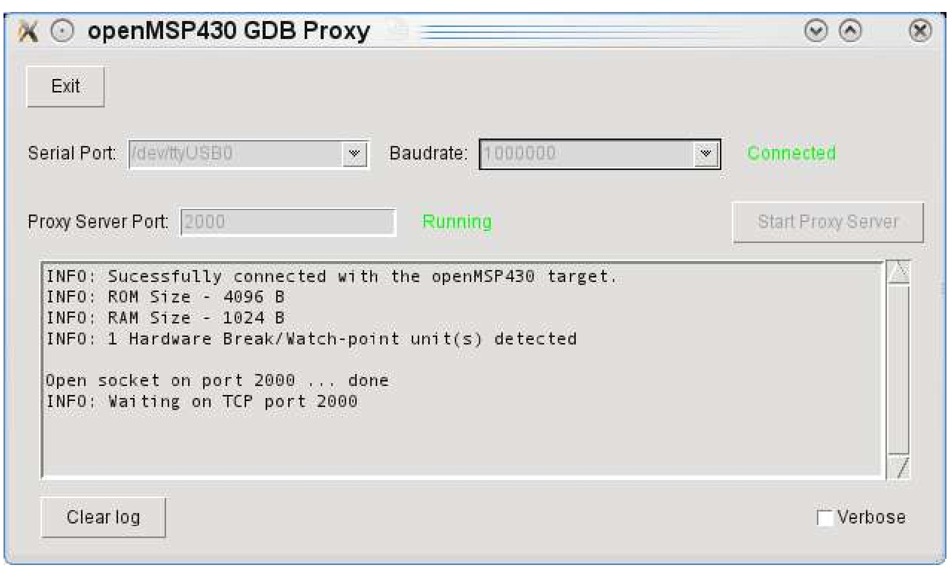
The purpose of this program is to replace the '***msp430-gdbproxy***' utility provided by the mspgcc toolchain.

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.

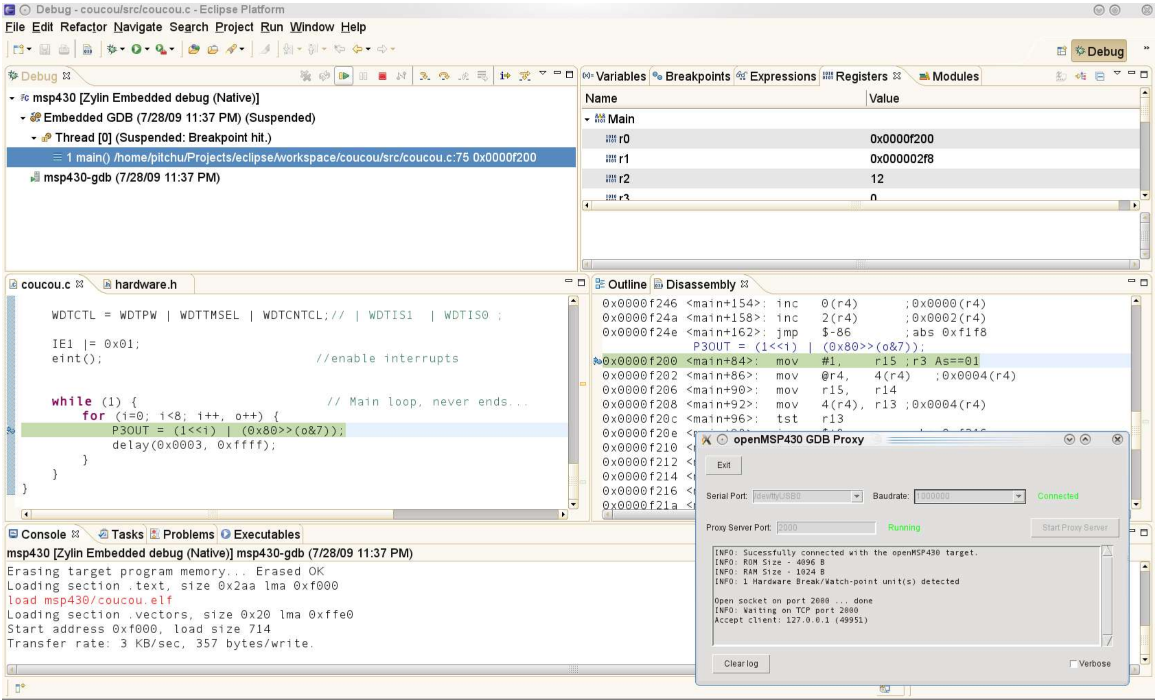
Schematically the communication flow looks as following:

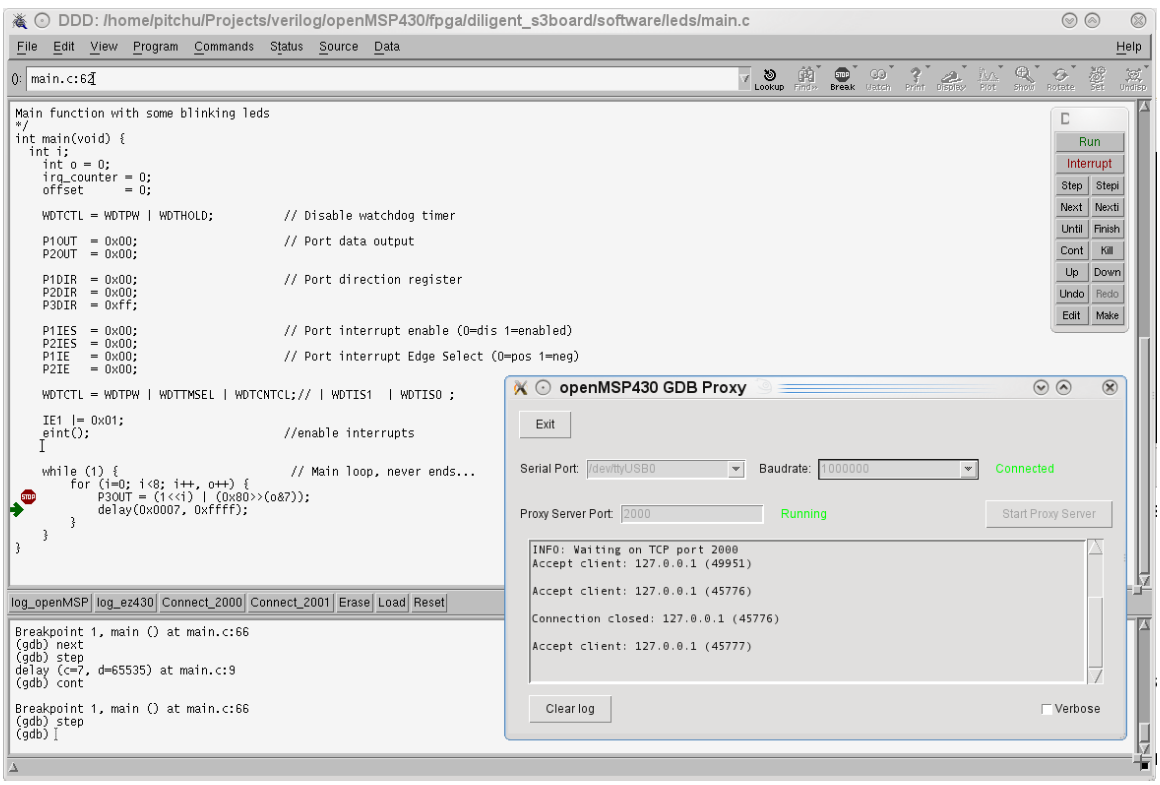


Like the original '***msp430-gdbproxy***' program, '***openmsp430-gdbproxy***' can be controlled from the command line. However, it also provides a small graphical interface:



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:

-mmcu option	Program Memory	Data Memory	Hardware Multiplier
Program Memory Size: 1 kB			
msp430x110	1 kB	128 B	No
msp430x1101	1 kB	128 B	No
msp430x2001	1 kB	128 B	No
msp430x2002	1 kB	128 B	No
msp430x2003	1 kB	128 B	No
msp430x2101	1 kB	128 B	No
Program Memory Size: 2 kB			
msp430x1111	2 kB	128 B	No
msp430x2011	2 kB	128 B	No
msp430x2012	2 kB	128 B	No

msp430x2013	2 kB	128 B	No
msp430x2111	2 kB	128 B	No
msp430x2112	2 kB	128 B	No
msp430x311	2 kB	128 B	No
Program Memory Size: 4 kB			
msp430x112	4 kB	256 B	No
msp430x1121	4 kB	256 B	No
msp430x1122	4 kB	256 B	No
msp430x122	4 kB	256 B	No
msp430x1222	4 kB	256 B	No
msp430x2122	4 kB	256 B	No
msp430x2121	4 kB	256 B	No
msp430x312	4 kB	256 B	No
msp430x412	4 kB	256 B	No
Program Memory Size: 8 kB			
msp430x123	8 kB	256 B	No
msp430x133	8 kB	256 B	No
msp430x313	8 kB	256 B	No
msp430x323	8 kB	256 B	No
msp430x413	8 kB	256 B	No
msp430x423	8 kB	256 B	**Yes**
msp430xE423	8 kB	256 B	**Yes**
msp430xE4232	8 kB	256 B	**Yes**
msp430xW423	8 kB	256 B	No
msp430x1132	8 kB	256 B	No
msp430x1232	8 kB	256 B	No
msp430x1331	8 kB	256 B	No
msp430x2131	8 kB	256 B	No
msp430x2132	8 kB	256 B	No
msp430x2232	8 kB	512 B	No
msp430x2234	8 kB	512 B	No
msp430x233	8 kB	1024 B	**Yes**
msp430x2330	8 kB	1024 B	**Yes**

Program Memory Size: 12 kB			
msp430xE4242	12 kB	512 B	**Yes**
msp430x314	12 kB	512 B	No
Program Memory Size: 16 kB			
msp430x4250	16 kB	256 B	No
msp430xG4250	16 kB	256 B	No
msp430x135	16 kB	512 B	No
msp430x1351	16 kB	512 B	No
msp430x155	16 kB	512 B	No
msp430x2252	16 kB	512 B	No
msp430x2254	16 kB	512 B	No
msp430x315	16 kB	512 B	No
msp430x325	16 kB	512 B	No
msp430x415	16 kB	512 B	No
msp430x425	16 kB	512 B	**Yes**
msp430xE425	16 kB	512 B	**Yes**
msp430xW425	16 kB	512 B	No
msp430xE4252	16 kB	512 B	**Yes**
msp430x435	16 kB	512 B	No
msp430x4351	16 kB	512 B	No
msp430x235	16 kB	2048 B	**Yes**
msp430x2350	16 kB	2048 B	**Yes**
Program Memory Size: 24 kB			
msp430x4260	24 kB	256 B	No
msp430xG4260	24 kB	256 B	No
msp430x156	24 kB	512 B	No
msp430x4361	24 kB	1024 B	No
msp430x436	24 kB	1024 B	No
msp430x336	24 kB	1024 B	**Yes**
Program Memory Size: 32 kB			
msp430x4270	32 kB	256 B	No
msp430xG4270	32 kB	256 B	No
msp430x147	32 kB	1024 B	**Yes**

msp430x1471	32 kB	1024 B	**Yes**
msp430x157	32 kB	1024 B	No
msp430x167	32 kB	1024 B	**Yes**
msp430x2272	32 kB	1024 B	No
msp430x2274	32 kB	1024 B	No
msp430x337	32 kB	1024 B	**Yes**
msp430x417	32 kB	1024 B	No
msp430x427	32 kB	1024 B	**Yes**
msp430xE427	32 kB	1024 B	**Yes**
msp430xE4272	32 kB	1024 B	**Yes**
msp430xW427	32 kB	1024 B	No
msp430x437	32 kB	1024 B	No
msp430xG437	32 kB	1024 B	No
msp430x4371	32 kB	1024 B	No
msp430x447	32 kB	1024 B	**Yes**
msp430x2370	32 kB	2048 B	**Yes**
msp430x247	32 kB	4096 B	**Yes**
msp430x2471	32 kB	4096 B	**Yes**
msp430x1610	32 kB	5120 B	**Yes**
Program Memory Size: 41 kB			
msp430x5438	41 kB	16384 B	No
msp430x5437	41 kB	16384 B	No
msp430x5436	41 kB	16384 B	No
msp430x5435	41 kB	16384 B	No
msp430x5419	41 kB	16384 B	No
msp430x54	41 kB	16384 B	No
Program Memory Size: 48 kB			
msp430x1611	48 kB	10240 B	**Yes**
msp430x248	48 kB	4096 B	**Yes**
msp430x2481	48 kB	4096 B	**Yes**
msp430x4783	48 kB	2048 B	**Yes**
msp430xG438	48 kB	2048 B	No
msp430x4784	48 kB	2048 B	

msp430x148	48 kB	2048 B	**Yes**
msp430x168	48 kB	2048 B	**Yes**
msp430x1481	48 kB	2048 B	**Yes**
msp430x448	48 kB	2048 B	**Yes**
Program Memory Size: 51 kB			
msp430xG4617	51 kB	8192 B	**Yes**
msp430x2418	51 kB	8192 B	**Yes**
msp430x2618	51 kB	8192 B	**Yes**
msp430x2417	51 kB	8192 B	**Yes**
msp430xG4618	51 kB	8192 B	**Yes**
msp430x2617	51 kB	8192 B	**Yes**
Program Memory Size: 54 kB			
msp430x1612	54 kB	5120 B	**Yes**
Program Memory Size: 55 kB			
msp430x2619	55 kB	4096 B	**Yes**
msp430xG4619	55 kB	4096 B	**Yes**
msp430xG4616	55 kB	4096 B	**Yes**
msp430x2416	55 kB	4096 B	**Yes**
msp430x2419	55 kB	4096 B	**Yes**
msp430x2616	55 kB	4096 B	**Yes**
msp430x2410	55 kB	4096 B	**Yes**
Program Memory Size: 59 kB			
msp430x4794	59 kB	2560 B	**Yes**
msp430x4793	59 kB	2560 B	**Yes**
msp430x2491	59 kB	2048 B	**Yes**
msp430x1491	59 kB	2048 B	**Yes**
msp430x149	59 kB	2048 B	**Yes**
msp430xG439	59 kB	2048 B	No
msp430x249	59 kB	2048 B	**Yes**
msp430x449	59 kB	2048 B	**Yes**
msp430x169	59 kB	2048 B	**Yes**

Note 1: the program memory size should imperatively match the openMSP430 configuration.
Note 2: the **-mforce-hwmul** parameter will force **msp430-gcc** to use the hardware multiplier.
Note 3: the **-mdisable-hwmul** parameter will force **msp430-gcc** not to use the hardware multiplier.

7.

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 (Program and Data memory size definition, Debug Interface configuration)*
openMSP430_undefines.v	*openMSP430 Verilog `undef file*
openMSP430.v	*openMSP430 top level*
omsp_frontend.v	*Instruction fetch and decode*
omsp_execution_unit.v	*Execution unit*
omsp_alu.v	*ALU*
omsp_register_file.v	*Register file*
omsp_mem_backbone.v	*Memory backbone*
omsp_clock_module.v	*Basic Clock Module*
omsp_sfr.v	*Special function registers*
omsp_watchdog.v	*Watchdog Timer*
omsp_multiplier.v	*16x16 Hardware Multiplier*
omsp_dbg.v	*Serial Debug Interface main block*
omsp_dbg_hwbrk.v	*Serial Debug Interface hardware breakpoint unit*
omsp_dbg_uart.v	*Serial Debug Interface UART communication block*
timescale.v	*Global time scale definition for simulation.*
periph	***Peripherals directory***
omsp_gpio.v	*Digital I/O (Port 1 to 6)*
omsp_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 program 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 the program memory 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*
				mpy_*.s43	*16x16 Multiplier assembler vector files*
				mpy_*.v	*16x16 Multiplier 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*
		actel			*Actel synthesis setup for area & speed analysis*
		altera			*Altera synthesis setup for area & speed analysis*
		xilinx			*Xilinx synthesis setup for area & speed analysis*

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)*
ram_8x2k_hi.*	*2 kByte RAM (upper byte)*
ram_8x2k_lo.*	*2 kByte RAM (lower byte)*
sim	***Top level simulations directory***
rtl_sim	***RTL simulations***
bin	***RTL simulation scripts***
msp430sim	*Main simulation script*
ihex2mem.tcl	*Verilog program 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 the program memory 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 program memory***
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_pmem.sh	*Update bitstream's program memory with a given software ELF file in a Linux environment*
load_pmem.bat	*Update bitstream's program memory with*

					a given software ELF file in a Windows environment
				memory.bmm	*FPGA memory description for bitstream's program memory 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 program 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 the program memory 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 the program memory***
			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*