openMSP430

Author: Olivier GIRARD

olgirard@gmail.com

Rev. 1.12

November 27, 2012

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..
1.8	March 1st, 2011	GIRARD	- Update openmsp430-minidebug tool section. - Add. Actel ProASIC3 example to the file and directory description section.
1.9	June 6th, 2011	GIRARD	- General update to reflect the latest RTL implementation (cpu_en/dbg_en ports, configurable peripheral address space, software development tools update...)..
1.10	March 20th, 2012	GIRARD	- Global update reflecting the ASIC support and corresponding configuration options.
1.11	July 15th, 2012	GIRARD	- Add benchmark results - Add custom memory size configuration
1.12	November 27th, 2012	GIRARD	- Global update reflecting the I2C based serial debug interface update.

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 near cycle accurate way.

The core comes with some peripherals (**16x16 Hardware Multiplier**, Watchdog, GPIO, TimerA, generic templates) and most notably with a two-wire **Serial Debug Interface** supporting **the MSPGCC GNU Debugger** (GDB) for in-system software debugging.

While being fully FPGA friendly, this design is also particularly suited for ASIC implementations (typically mixed signal ICs with strong area and low-power requirements).

In a nutshell, the openMSP430 brings with it:

- Low area (8k-Gates), without hidden extra infrastructure overhead (memory backbone, IRQ controller and watchdog timer are already included).

- Excellent code density.

- Good performances.

- Build-in power and clock management options.

- Multiple time **Silicon Proven**.

1

Download

Design

The complete tar archive of the project can be downloaded here (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
```

Changelog

- The Core's ChangeLog lists the CPU updates
- The Tools' ChangeLog lists the Software development tools updates.
- Subscribe to the following RSS feed to keep yourself informed about ALL updates.

Documentation

Being fully compatible with the original MSP430 architecture, TI's official documentation is applicable: SLAU49F.PDF

In addition, the openMSP430 online documentation is also available in pdf.

Features & Limitations

Features

- **Core:**
  - Full instruction set support.
  - Interrupts: IRQs (x14), NMI (x1).
  - Power saving modes.
  - Configurable memory size for both program and data.
  - Scalable peripheral address space.
  - Two-wire Serial Debug Interface (Nexus class 3, w/o trace) with GDB support (I^2C or UART based).
  - FPGA friendly (option for single clock domain, no clock gate).
  - ASIC friendly (options for full power & clock management support).
  - Small size (Xilinx: 1650 LUTs / Altera: 1550 LEs / ASIC: 8k gates).

- **Peripherals:**
    - 16x16 Hardware Multiplier.
    - Basic Clock Module.
    - Watchdog.
    - Timer A (FPGA only).
    - GPIO (FPGA only).
    - Templates for 8 and 16 bit peripherals.

Limitations

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

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 free FPGA synthesis tool.

A few MSP430 links:

- Wikipedia: MSP430
- TI: MSP430x1xx Family User's Guide
- TI: MSP430 Competitive Benchmarking
- 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.

Depending on the selected configuration, this design can either be:

- **FPGA friendly**: the core doesn't contain any clock gate and has only a single clock domain. As a consequence, in this mode, the *Basic Clock Module* peripheral has a few limitations.

- **ASIC friendly**: the core contains up to all clock management options (clock muxes & low-power modes, fine grained clock gating, …) and is also ready for scan insertion. In this mode, the *Basic Clock Module* offers all features listed in the official documentation.

It is to be noted that 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:
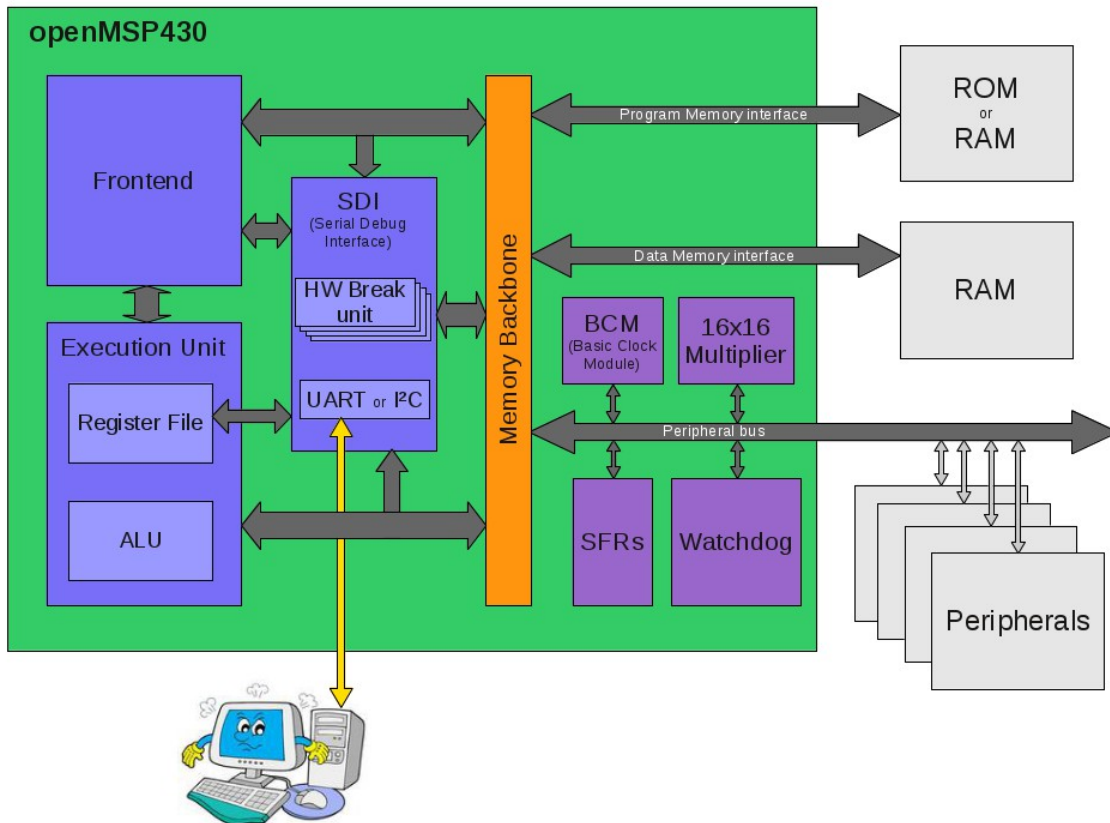
- **Frontend**: This module performs the instruction Fetch and Decode tasks. It also contains the execution state machine.
- **Execution unit**: Containing the ALU and the register file, this module executes the current decoded instruction according to the execution state.
- **Serial Debug Interface**: Contains all the required logic for a Nexus class 3 debugging unit (without trace). Communication with the host is done with a standard two-wire interface following either the UART 8N1 or I^2C serial protocol.
- **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 MCLK, ACLK, SMCLK and manage the low power modes.
- **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 directly included in the core because of its tight links with the NMI interrupts and the PUC reset generation.
- **16x16 Multiplier**: The hardware multiplier peripheral is transparently supported by the GCC compiler and is therefore located in the core. It can be included or excluded at will through a Verilog define.

2.1.2 Limitations

The known core limitations are the following:

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

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).

In this section, three sets of adjustable user parameters are discussed in order to customize the core. A fourth set is available for ASIC specific options and will be discussed in the ASIC implementation section.

2.1.3.1 Basic System Configuration

The basic system can be adjusted with the following set of defines in order to match the target system requirements.

```
//=====================================================================
//=====================================================================
//BASIC SYSTEM CONFIGURATION
//=====================================================================
//=====================================================================
//
// Note: the sum of program, data and peripheral memory spaces must not
// exceed 64 kB
//

// Program Memory Size:
//                       Uncomment the required memory size
//-------------------------------------------------------
//`define PMEM_SIZE_CUSTOM
//`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:
     //                 Uncomment the required memory size
     //------------------------------------------------------
     //`define DMEM_SIZE_CUSTOM
     //`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

     // Include/Exclude Hardware Multiplier
     `define MULTIPLIER

     // Include/Exclude Serial Debug interface
     `define DBG_EN
```

The only design considerations at this stage are:

- Make sure that the program and data memories have the correct size :-P
- The sum of program, data and peripheral memory space **MUST NOT** exceed 64kB.

Note: when selected, custom memory sizes can be specified in the "Expert System Configuration" section.

2.1.3.2 Advanced System Configuration

In this section, some additional features are available in order to match the needs of more experienced users.

```
//======================================================================
//======================================================================
// ADVANCED SYSTEM CONFIGURATION (FOR EXPERIENCED USERS)
//======================================================================
//======================================================================

//--------------------------------------------------------
// Custom user version number
//--------------------------------------------------------
// This 5 bit field can be freely used in order to allow
// custom identification of the system through the debug
// interface.
// (see CPU_ID.USER_VERSION field in the documentation)
//--------------------------------------------------------
```

```
`define USER_VERSION 5'b00000


//----------------------------------------------------------
// Include/Exclude Watchdog timer
//----------------------------------------------------------
// When excluded, the following functionality will be
// lost:
//          - Watchog (both interval and watchdog modes)
//          - NMI interrupt edge selection
//          - Possibility to generate a software PUC reset
//----------------------------------------------------------
`define WATCHDOG



///----------------------------------------------------------
// Include/Exclude Non-Maskable-Interrupt support
//----------------------------------------------------------
`define NMI


//----------------------------------------------------------
// Input synchronizers
//----------------------------------------------------------
// In some cases, the asynchronous input ports might
// already be synchronized externally.
// If an extensive CDC design review showed that this
// is really the case,  the individual synchronizers
// can be disabled with the following defines.
//
// Notes:
//          - all three signals are all sampled in the MCLK domain
//
//          - the dbg_en signal reset the debug interface
//           when 0. Therefore make sure it is glitch free.
//
//----------------------------------------------------------
`define SYNC_NMI
//`define SYNC_CPU_EN
//`define SYNC_DBG_EN



//----------------------------------------------------------
// Peripheral Memory Space:
//----------------------------------------------------------
// The original MSP430 architecture map the peripherals
// from 0x0000 to 0x01FF (i.e. 512B of the memory space).
// The following defines allow you to expand this space
// up to 32 kB (i.e. from 0x0000 to 0x7fff).
// As a consequence, the data memory mapping will be
// shifted up and a custom linker script will therefore
// be required by the GCC compiler.
//----------------------------------------------------------
//`define PER_SIZE_CUSTOM
//`define PER_SIZE_32_KB
//`define PER_SIZE_16_KB
//`define PER_SIZE_8_KB
//`define PER_SIZE_4_KB
//`define PER_SIZE_2_KB
//`define PER_SIZE_1_KB
`define PER_SIZE_512_B
```

```
//--------------------------------------------------------
// Defines the debugger CPU_CTL.RST_BRK_EN reset value
// (CPU break on PUC reset)
//--------------------------------------------------------
// When defined, the CPU will automatically break after
// a PUC occurrence by default. This is typically useful
// when the program memory can only be initialized through
// the serial debug interface.
//--------------------------------------------------------
`define DBG_RST_BRK_EN
```

Design consideration at this stage are:

- Setting a peripheral memory space to something else than 512B will shift the data memory mapping up, which in turn will require the use of a custom linker script. If you don't know what a linker script is and if you don't want to know what it is, you should probably not modify this section.
- The sum of program, data and peripheral memory space **MUST NOT** exceed 64kB.

Note: when selected, custom peripheral memory space can be specified in the "Expert System Configuration" section.

2.1.3.3 Expert System Configuration

In this section, you will find configuration options which will be relevant for roughly 0.1% of the users (according to a highly reliable market analysis ;-)).

```
//===========================================================================
//===========================================================================
// EXPERT SYSTEM CONFIGURATION ( !!!! EXPERTS ONLY !!!! )
//===========================================================================
//===========================================================================
//
// IMPORTANT NOTE: Please update following configuration options ONLY if
//                 you have a good reason to do so... and if you know what
//                 you are doing :-P
//
//===========================================================================

//--------------------------------------------------------
// Select serial debug interface protocol
//--------------------------------------------------------
//    DBG_UART -> Enable UART (8N1) debug interface
//    DBG_I2C  -> Enable I2C debug interface
//--------------------------------------------------------
`define DBG_UART
//`define DBG_I2C


//--------------------------------------------------------
// Enable the I2C broadcast address
```

```
//----------------------------------------------------------
// For multicore systems, a common I2C broadcast address
// can be given to all oMSP cores in order to
// synchronously RESET, START, STOP, or STEP all CPUs
// at once with a single I2C command.
// If you have a single openMSP430 in your system,
// this option can stay commented-out.
//----------------------------------------------------------
//`define DBG_I2C_BROADCAST


//----------------------------------------------------------
// Number of hardware breakpoint units (each unit contains
// two hardware address breakpoints):
// - DBG_HWBRK_0 -> Include hardware breakpoints unit 0
// - DBG_HWBRK_1 -> Include hardware breakpoints unit 1
// - DBG_HWBRK_2 -> Include hardware breakpoints unit 2
// - DBG_HWBRK_3 -> Include hardware breakpoints unit 3
//----------------------------------------------------------
// Please keep in mind that hardware breakpoints only
// make sense whenever the program memory is not an SRAM
// (i.e. Flash/OTP/ROM/...) or when you are interested
// in data breakpoints (btw. not supported by GDB).
//----------------------------------------------------------
//`define DBG_HWBRK_0
//`define DBG_HWBRK_1
//`define DBG_HWBRK_2
//`define DBG_HWBRK_3


//----------------------------------------------------------
// Enable/Disable the hardware breakpoint RANGE mode
//----------------------------------------------------------
// When enabled this feature allows the hardware breakpoint
// units to stop the cpu whenever an instruction or data
// access lays within an address range.
// Note that this feature is not supported by GDB.
//----------------------------------------------------------
//`define DBG_HWBRK_RANGE


//----------------------------------------------------------
// Custom Program/Data and Peripheral Memory Spaces
//----------------------------------------------------------
// The following values are valid only if the
// corresponding *_SIZE_CUSTOM defines are uncommented:
//
//   - *_SIZE   : size of the section in bytes.
//   - *_AWIDTH : address port width, this value must allow
//                to address all WORDS of the section
//                (i.e. the *_SIZE divided by 2)
//----------------------------------------------------------

// Custom Program memory (enabled with PMEM_SIZE_CUSTOM)
`define PMEM_CUSTOM_AWIDTH      10
`define PMEM_CUSTOM_SIZE      2048

// Custom Data memory    (enabled with DMEM_SIZE_CUSTOM)
`define DMEM_CUSTOM_AWIDTH       6
`define DMEM_CUSTOM_SIZE       128

// Custom Peripheral memory  (enabled with PER_SIZE_CUSTOM)
```

```
`define PER_CUSTOM_AWIDTH        8
`define PER_CUSTOM_SIZE        512

//--------------------------------------------------------
// ASIC version
//--------------------------------------------------------
// When uncommented, this define will enable the
// ASIC system configuration section (see below) and
// will activate scan support for production test.
//
// WARNING: if you target an FPGA, leave this define
//          commented.
//--------------------------------------------------------
//`define ASIC
```

Design consideration at this stage are:

- This is the expert section... so you know what your are doing right ;-)

All remaining defines located after the ASIC section in the ***openMSP430_defines.v*** file are system constants and **MUST NOT** be edited.

2.1.3.4 Parameters For Multi-Core Systems

In addition to the define file, two Verilog parmaeters are available to facilitate software development on multi-core systems.

For example, in a dual-core openMSP430 system, the cores can be instantiated as following:

```
openMSP430 #(.INST_NR (0), .TOTAL_NR(1)) openMSP430_core_0 (
...
);

openMSP430 #(.INST_NR (1), .TOTAL_NR(1)) openMSP430_core_1 (
...
);
```

The values of these parameters are then directly accessible through the CPU_NR register of the SFR peripheral.

For example, if both cores share the same program memory, the software can take advantage of this information as following:

```
"...
int main(void) {
  if (CPU_NR==0x0100) {
    main_core_0(); // Main routine call for core 0
  }
  if (CPU_NR==0x0101) {
    main_core_1(); // Main routine call for core 1
  }
}
…"
```

2.1.4 Memory mapping

As discussed earlier, the openMSP430 memory mapping is fully configurable.

The basic system configuration section allows to adjust program and data memory sizes while keeping 100% compatibility with the pre-existing linker scripts provided by MSPGCC (or any other toolchain for that matter).

However, an increasing number of users saw the 512B space available for peripherals in the standard MSP430 architecture as a limitation. Therefore, the advanced system configuration section gives the possibility to up-scale the reserved peripheral address space anywhere between 512B and 32kB. As a consequence, the data memory space will be shifted up, which means that the linker script of your favorite toolchain will have to be modified accordingly.

The following schematic should hopefully summarize this:
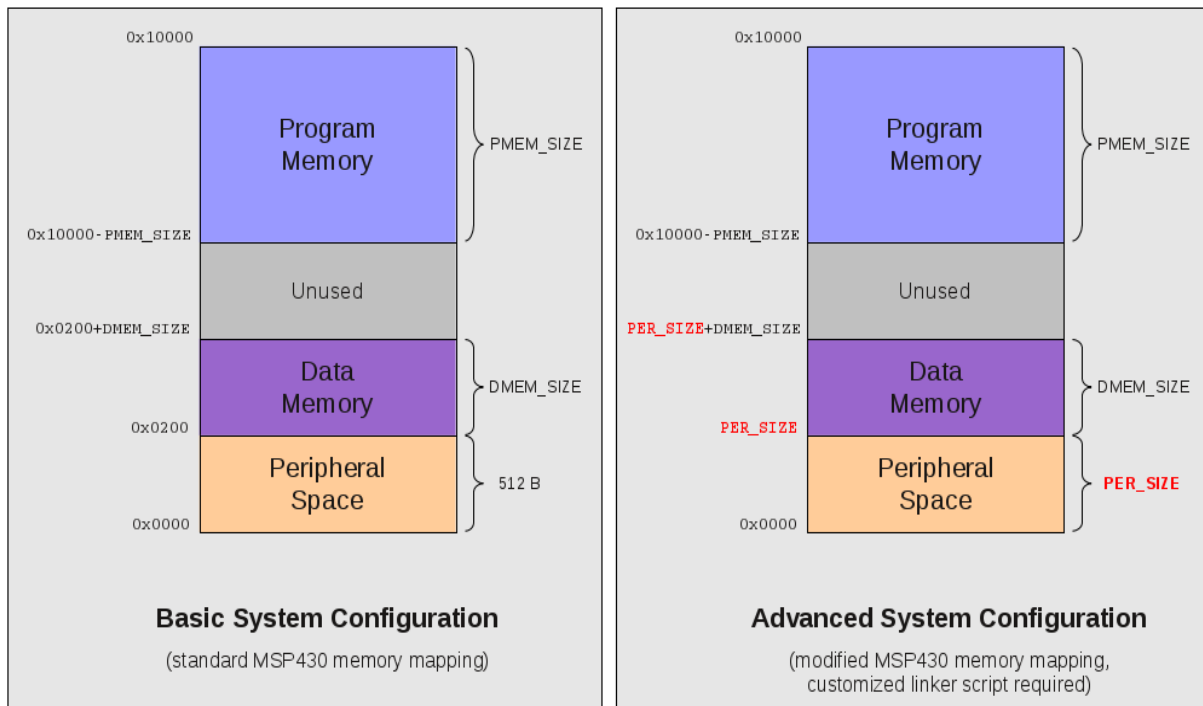
2.1.5 Pinout

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

Port Name	Direction	Width	Clock Domain	Description
Clocks				
cpu_en	Input	1	\<async\> or mclk[4]	Enable CPU code execution (asynchronous and non-glitchy). Set to 1 if unused.
dco_clk	Input	1	-	Fast oscillator (fast clock)
lfxt_clk	Input	1	-	Low frequency oscillator (typ. 32kHz) Set to 0 if unused.
mclk	Output	1	-	Main system clock
aclk_en	Output	1	mclk	FPGA ONLY: ACLK enable
smclk_en	Output	1	mclk	FPGA ONLY: SMCLK enable
dco_enable	Output	1	dco_clk	ASIC ONLY: Fast oscillator enable
dco_wkup	Output	1	\<async\>	ASIC ONLY: Fast oscillator wakeup (asynchronous)
lfxt_enable	Output	1	lfxt_clk	ASIC ONLY: Low frequency oscillator enable
lfxt_wkup	Output	1	\<async\>	ASIC ONLY: Low frequency oscillator wakeup (asynchronous)
aclk	Output	1	-	ASIC ONLY: ACLK
smclk	Output	1	-	ASIC ONLY: SMCLK
wkup	Input	1	\<async\>	ASIC ONLY: System Wake-up (asynchronous and non-glitchy) Set 0 if unused.
Resets				
puc_rst	Output	1	mclk	Main system reset
reset_n	Input	1	\<async\>	Reset Pin (active low, asynchronous and non-glitchy)
Interrupts				
irq	Input	14	mclk	Maskable interrupts (one-hot signal)
nmi	Input	1	\<async\> or mclk[4]	Non-maskable interrupt (asynchronous and non-glitchy) Set to 0 if unused.
irq_acc	Output	14	mclk	Interrupt request accepted (one-hot signal)
Program Memory interface				

pmem_addr	Output	`PMEM_AWIDTH[1]	mclk	Program Memory address
pmem_cen	Output	1	mclk	Program Memory chip enable (low active)
pmem_din	Output	16	mclk	Program Memory data input (optional[2])
pmem_dout	Input	16	mclk	Program Memory data output
pmem_wen	Output	2	mclk	Program Memory write byte enable (low active) (optional[2])
Data Memory interface				
dmem_addr	Output	`DMEM_AWIDTH[1]	mclk	Data Memory address
dmem_cen	Output	1	mclk	Data Memory chip enable (low active)
dmem_din	Output	16	mclk	Data Memory data input
dmem_dout	Input	16	mclk	Data Memory data output
dmem_wen	Output	2	mclk	Data Memory write byte enable (low active)
External Peripherals interface				
per_addr	Output	14	mclk	Peripheral address
per_din	Output	16	mclk	Peripheral data input
per_dout	Input	16	mclk	Peripheral data output
per_en	Output	1	mclk	Peripheral enable (high active)
per_we	Output	2	mclk	Peripheral write enable (high active)
Serial Debug interface				
dbg_en	Input	1	<async> or mclk[4]	Debug interface enable (asynchronous) [3]
dbg_freeze	Output	1	mclk	Freeze peripherals
dbg_uart_txd	Output	1	mclk	Debug interface: UART TXD
dbg_uart_rxd	Input	1	<async>	Debug interface: UART RXD (asynchronous)
dbg_i2c_addr	Input	7	mclk	Debug interface: I2C Address
dbg_i2c_broadcast	Input	7	mclk	Debug interface: I2C Broadcast Address (for multicore systems)
dbg_i2c_scl	Input	1	<async>	Debug interface: I2C SCL (asynchronous)
dbg_i2c_sda_in	Input	1	<async>	Debug interface: I2C SDA IN (asynchronous)
dbg_i2c_sda_out	Output	1	mclk	Debug interface: I2C SDA OUT
Scan				
scan_enable	Input	1	dco_clk	ASIC ONLY: Scan enable (active during

				scan shifting)
scan_mode	Input	1	<stable>	ASIC ONLY: Scan mode

[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.

[3]: When disabled, the debug interface is hold into reset (and clock gated in ASIC mode). As a consequence, the **dbg_en** port can be used to reset the debug interface without disrupting the CPU execution.

[4]: Clock domain is selectable through configuration in the "openMSP430_defines.v" file (see Advanced System Configuration).

Note: in the FPGA configuration, the ASIC ONLY signals must be left unconnected (for the outputs) and tied low (for the inputs).

2.1.6 Instruction Cycles and Lengths

Please note that a detailed description of the instruction and addressing modes can be found in the MSP430x1xx Family User's Guide (Chapter 3).

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

2.1.7 Serial Debug Interface

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

2.1.8 Benchmark results

2.1.8.1 Dhrystone (DMIPS/MHz)

Dhrystone is known for being susceptible to compiler optimizations (among other issues). However, as it is still quite a popular metric, some results are provided here (ranging from 0.30 to 0.45 DMIPS/MHz depending on the compiler version and options).

Note that the used C-code is available in the repository here and here.

Dhrystone flavor	Compiler options		-Os	-O2	-O3
	Compiler version				
Dhrystone v2.1 (common version)	mspgcc v4.4.5		0.30	0.32	0.33
	mspgcc v4.6.3		0.37	0.39	0.40
Dhrystone v2.1 (MCU adapted)	mspgcc v4.4.5		0.30	0.30	0.31
	mspgcc v4.6.3		0.37	0.44	0.45

2.1.8.2 CoreMark (Coremark/MHz)

CoreMark tries to address most of Dhrystone's pitfall by preventing the compiler to optimize some code away and using "real-life" algorithm.

Note that the used C-code is available in the repository here.

	Compiler options		-Os	-O2	-O3
	Compiler version				
CoreMark v1.0 (official version)	mspgcc v4.4.5		0.78	0.85	0.83
	mspgcc v4.6.3		0.74	0.91	0.87

2.2 System Peripherals

In addition to the CPU core itself, several peripherals are also provided and can be easily connected to the core during integration. The followings are directly integrated within the core because of their tight links with the CPU.

It is to be noted that **ALL** system peripherals support both ASIC and FPGA versions.

2.2.1 Basic Clock Module: FPGA

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

With these constrains, the Basic Clock Module is implemented as following:
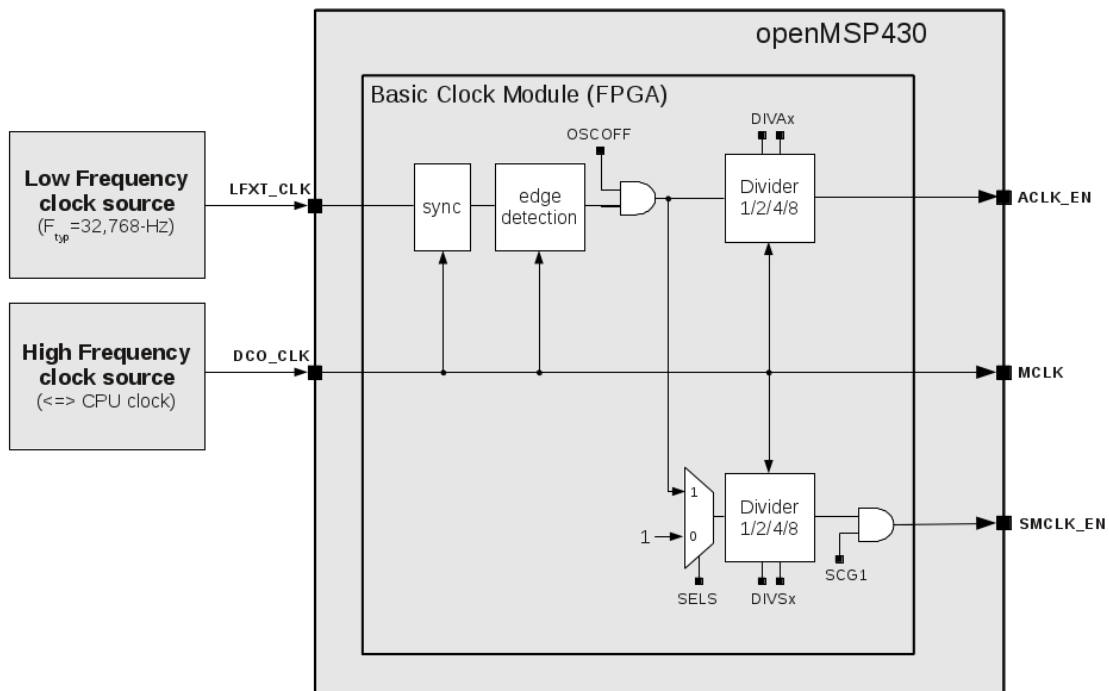
Note: CPUOFF doesn't switch MCLK off and will instead bring the CPU state machines in an IDLE state while MCLK will still be running.

In order to '*clock*' a register with ACLK or SMCLK, the following structure needs to be implemented:
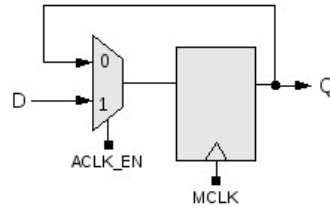
For example, the following Verilog code would implement a counter clocked with SMCLK:

```
reg [7:0] test_cnt;

always @ (posedge mclk or posedge puc_rst)
if (puc_rst)        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 Basic Clock Module: ASIC

When targeting an ASIC, up to all clock management options available in the MSP430x1xx Family User's Guide (Chapter 4) can be included:
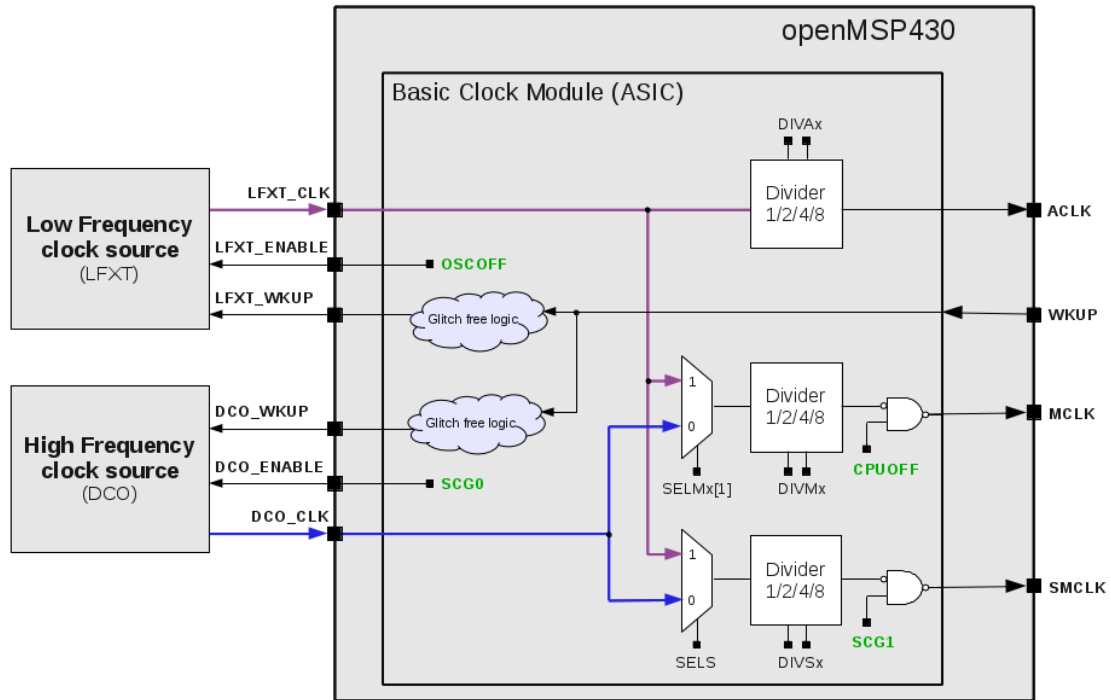
Additional info can be found in the ASIC implementation section.

2.2.3 SFR

Following the [MSP430x1xx Family User's Guide](), this peripheral implements flags and interrupt enable bits for the Watchdog Timer and NMI:

Register Name	Address	Bit Fields							
		7	6	5	4	3	2	1	0
IE1	0x0000	Reserved			NMIIE [1]	Reserved			WDTIE [2]
IFG1	0x0002	Reserved			NMIIFG [1]	Reserved			WDTIFG [2]

[1]: These fields are not available if the NMI is excluded (see *openMSP430_defines.v*)
[2]: These fields are not available if the Watchdog is excluded (see *openMSP430_defines.v*)

In addition, two 16-bit read-only registers have been added in order to let the software know with which version of the openMSP430 it is running:

Register Name	Address	Bit field															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CPU_ID_LO	0x0004	PER_SPACE							USER_VERSION			ASIC	CPU_VERSION				
CPU_ID_HI	0x0006	PMEM_SIZE						DMEM_SIZE									MPY
CPU_NR	0x0008	CPU_TOTAL_NR							CPU_INST_NR								

- **CPU_VERSION** : Current CPU version.

- **ASIC**   : Defines if the ASIC specific features are enabled in the current openMSP430 implementation.

- **USER_VERSION**   : Reflects the value defined in the ***openMSP430_defines.v*** file.

- **PER_SPACE**   : Peripheral address space for the current implementation (byte size = PER_SPACE*512)

- **MPY**   : This bit is set if the hardware multiplier is inclued in the current implementation.

- **DMEM_SIZE**   : Data memory size for the current implementation (byte size = DMEM_SIZE*128)

- **PMEM_SIZE**   : Program memory size for the current implementation (byte size = PMEM_SIZE*1024)

- **CPU_INST_NR**   : Current oMSP instance number (for multicore systems)

- **CPU_TOTAL_NR**   : Total number of oMSP instances-1 (for multicore systems)

Note: attentive readers will have noted that *CPU_ID_LO*, *CPU_ID_HI* and *CPU_NR* are identical to the Serial Debug Interface register counterparts.

2.2.4 Watchdog Timer

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

The following parameter in the *openMSP430_defines.v* file controls if the watchdog timer should be included or not:

```
//----------------------------------------------------------
// Include/Exclude Watchdog timer
//----------------------------------------------------------
// When excluded, the following functionality will be
// lost:
//          - Watchdog (both interval and watchdog modes)
//          - NMI interrupt edge selection
//          - Possibility to generate a software PUC reset
//----------------------------------------------------------
`define WATCHDOG
```

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
```

2.3 External Peripherals

The external peripherals labeled with the "FPGA ONLY" tag do not contain any clock gate nor clock muxes and are clocked with MCLK only. This mean that they don't support any of the low power modes and therefore are most likely not suited for an ASIC implementation.

2.3.1 Digital I/O (FPGA ONLY)

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_rst	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

Port Name	Direction	Width	Description
p6_dout_en	Output	8	Port 6 data output enable
p6_sel	Output	8	Port 6 function select

2.3.2 Timer A (FPGA ONLY)

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_rst	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 FPGA version, 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 allow in-system software debugging. In that case, the communication with the host computer is typically built 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 two-wire cable following either the UART or I²C serial protocol (interface is selectable in the Expert System Configuration section).

The debug unit provides all required features for Nexus Class 3 debugging (beside trace), namely:

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

Depending on the selected serial interface, the following features are available:

Debug unit features	
UART	**I²C**
Strengths: • No extra hardware required for most FPGA boards (almost all come with a UART interface, either RS232 or USB based. • Possibility to use USB to serial TTL cables. **Weaknesses:** • Need to reset the debug interface after cable insertion. • For ASICs, no possibility to change the MCLK frequency during a debug session.	**Strengths:** • Very stable interface (synchronous protocol, no synchronization frame required). • Multi-core chip support with a single I2C interface (i.e. TWO pins)... in such a system, each openMSP430 instance has its own I2C device address. • Possibility to combine the openMSP430 debug interface with an already existing "functional" I2C interface... effectively creating a ZERO wire serial debug interface. • Affordable USB-ISS adapter (≈23€). **Weaknesses:** • Extra I2C adapter required (USB-ISS currently supported).

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	PER_SPACE	USER_VERSION				ASIC	CPU_VERSION		
CPU_ID_HI	0x01	PMEM_SIZE	DMEM_SIZE							MPY
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]						
CPU_NR	0x18	CPU_TOTAL_NR	CPU_INST_NR					

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	PER_SPACE						USER_VERSION					ASIC	CPU_VERSION		
CPU_ID_HI	0x01	PMEM_SIZE						DMEM_SIZE									MPY

- **CPU_VERSION** : Current CPU version

- **ASIC** : Defines if the ASIC specific features are enabled in the current openMSP430 implementation.

- **USER_VERSION** : Reflects the value defined in the *openMSP430_defines.v* file

- **PER_SPACE** : Peripheral address space for the current implementation (byte size = PER_SPACE*512)

- **MPY** : This bit is set if the hardware multiplier is included in the current implementation.

- **DMEM_SIZE** : Data memory size for the current implementation (byte size = DMEM_SIZE * 128)

- **PMEM_SIZE** : Program memory size for the current implementation (byte size = PMEM_SIZE * 1024)

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 0x10 or 0x30 (depending on the **DBG_RST_BRK_EN** configuration option).

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.

2.2.4 CPU_NR

This 16 bit read only register gives useful information for multi-core systems.

Register Name	Address	Bit Field															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CPU_NR	0x18	CPU_TOTAL_NR								CPU_INST_NR							

- **CPU_TOTAL_NR** : Total number of oMSP instances – 1 (for multicore systems).

- **CPU_INST_NR** : Current oMSP instance number (for multicore systems).

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 gives (wr) or receive (rd) the Memory or CPU-Register data for 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 `DBG_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 `DBG_HWBRK_RANGE define is set to 1'b1 in the

- **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).

Typically, a USB to RS232 or USB to serial TTL cable will provide a reliable communication link between your host PC and the openMSP430 (speed being typically limited by the cable length).

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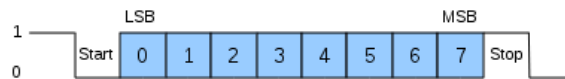
As you can see in the above diagram, data transmission starts with a Start bit, followed by the data bits (LSB sent first and MSB sent last), and ends with a "Stop" bit.

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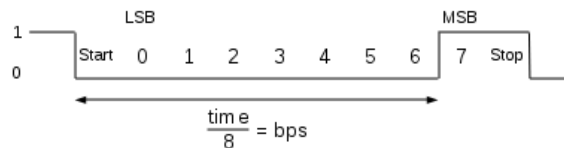
As you can see, the host simply sends the 0x80 value. The openMSP430 will then measure the time between the falling and rising edge, divide it by 8 and automatically deduce the baud rate it should use to properly communicate with the host.

Important note: if you want to change the communication speed between two debugging sessions, the Serial Debug Interface needs to go through a reset cycle (i.e. through the *reset_n* or *dbg_en* pins) and a new synchronization frame needs to be send.
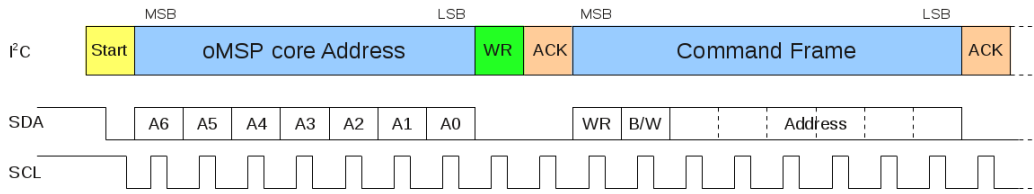
3.3 Read/Write access to the debug registers

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:

- <u>8-bit:</u>

Host TX:	1	1	Address	Data

Host RX:

- <u>16-bit:</u>

Host TX:	1	0	Address	Data[7:0]	Data[15:8]

Host RX:

3.3.3 Read access

A read access transaction looks like this:
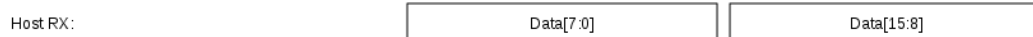
- <u>8-bit:</u>

Host TX:	0	1	Address

Host RX:			Data

- <u>16-bit:</u>

Host TX:	0	0	Address

Host RX:			Data[7:0]	Data[15:8]

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:
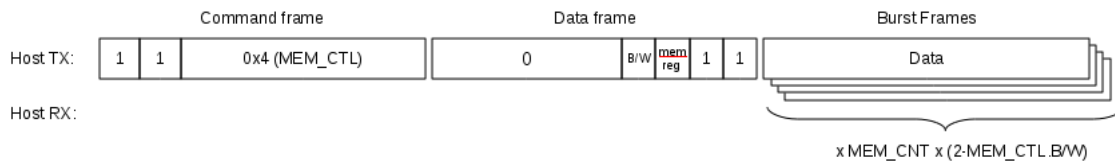
3.4.2 Read Burst access

A read burst transaction looks like this:
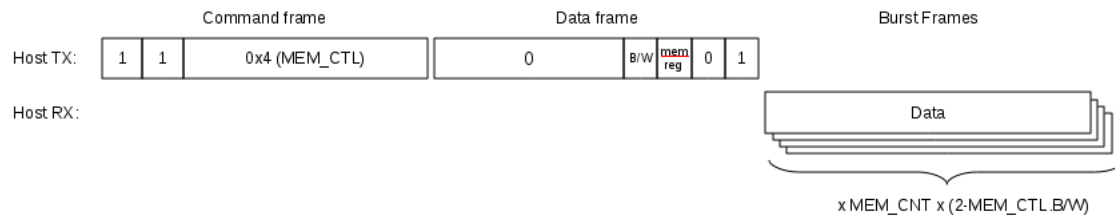
4. Debug Communication Interface: I2C

With its I2C interface, the openMSP430 debug unit can communicate with the host computer using an I2C adapter (connected to the dbg_i2c_scl and dbg_i2c_sda_in/dbg_i2c_sda_out ports of the IP).

Currently, the USB-ISS adapter from Devantech (Robot Electronics) is supported by the software development tools and provides a reliable communication link between your host PC and the openMSP430.

4.1 I2C communication protocol

There are plenty tutorials on Internet regarding the I2C protocol (see the official I2C specification for more info).

A simple byte read or write frame looks as following:
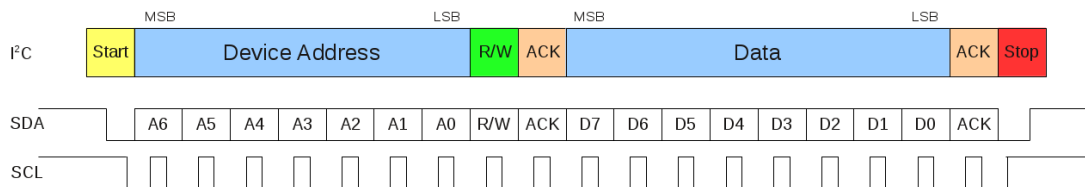
4.2 Synchronization frame

Unlike the UART interface, the I2C is a synchronous communication protocol.

A synchronization frame is therefore not required.

4.3 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.

4.3.1 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.

4.3.2 Write access
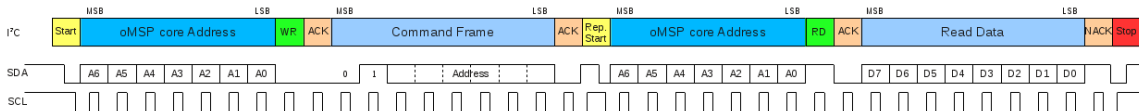
A write access transaction looks like this:
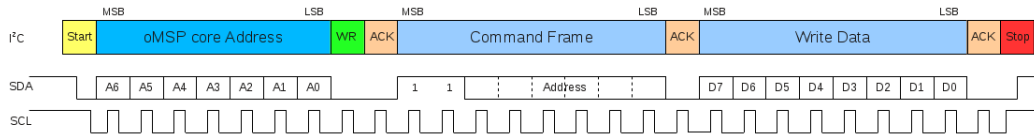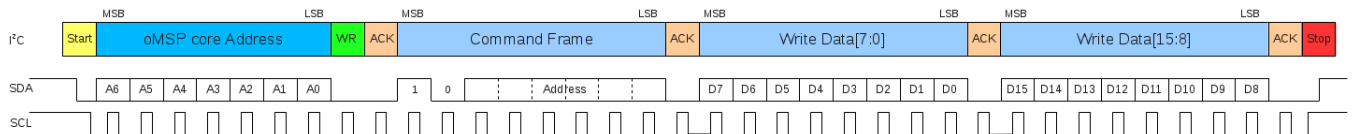
• 8-bit:

• 16-bit:
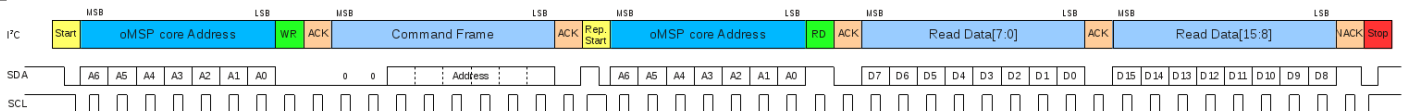

4.3.3 Read access

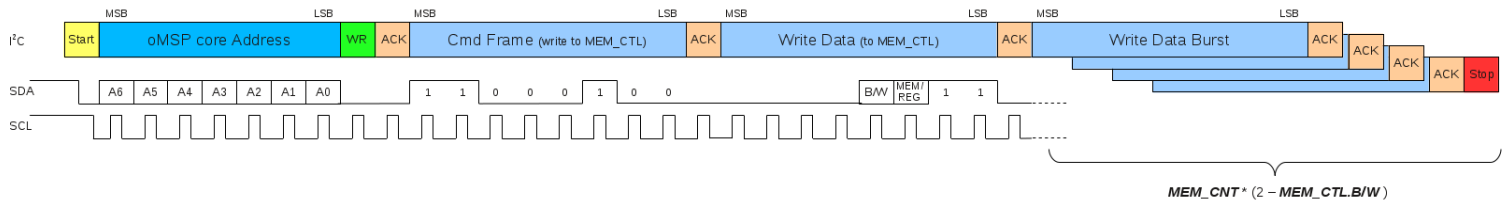A read access transaction looks like this:

• 8-bit:

• 16-bit:

4.4 Read/Write burst implementation for the CPU Memory access

In order to optimize the data burst transactions for the I2C, 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.

4.4.1 Write Burst access

A write burst transaction looks like this:

4.4.2 Read Burst access

A read burst transaction looks like this:
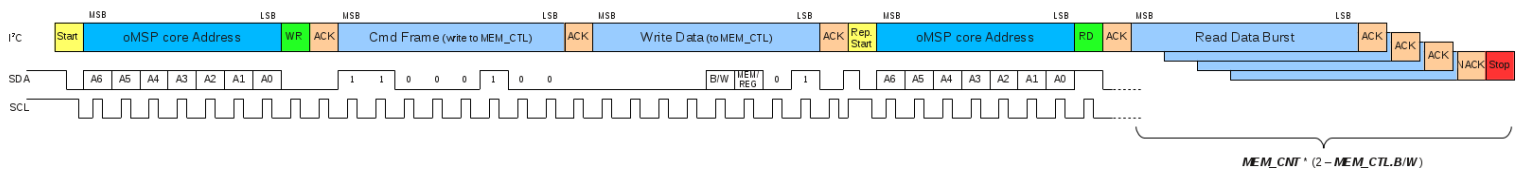
4.

Integration and Connectivity

Table of content

1. Overview

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

The following diagram shows an overview of the openMSP430 core connectivity in an FPGA system (i.e. all ASIC specific pins are left unused):
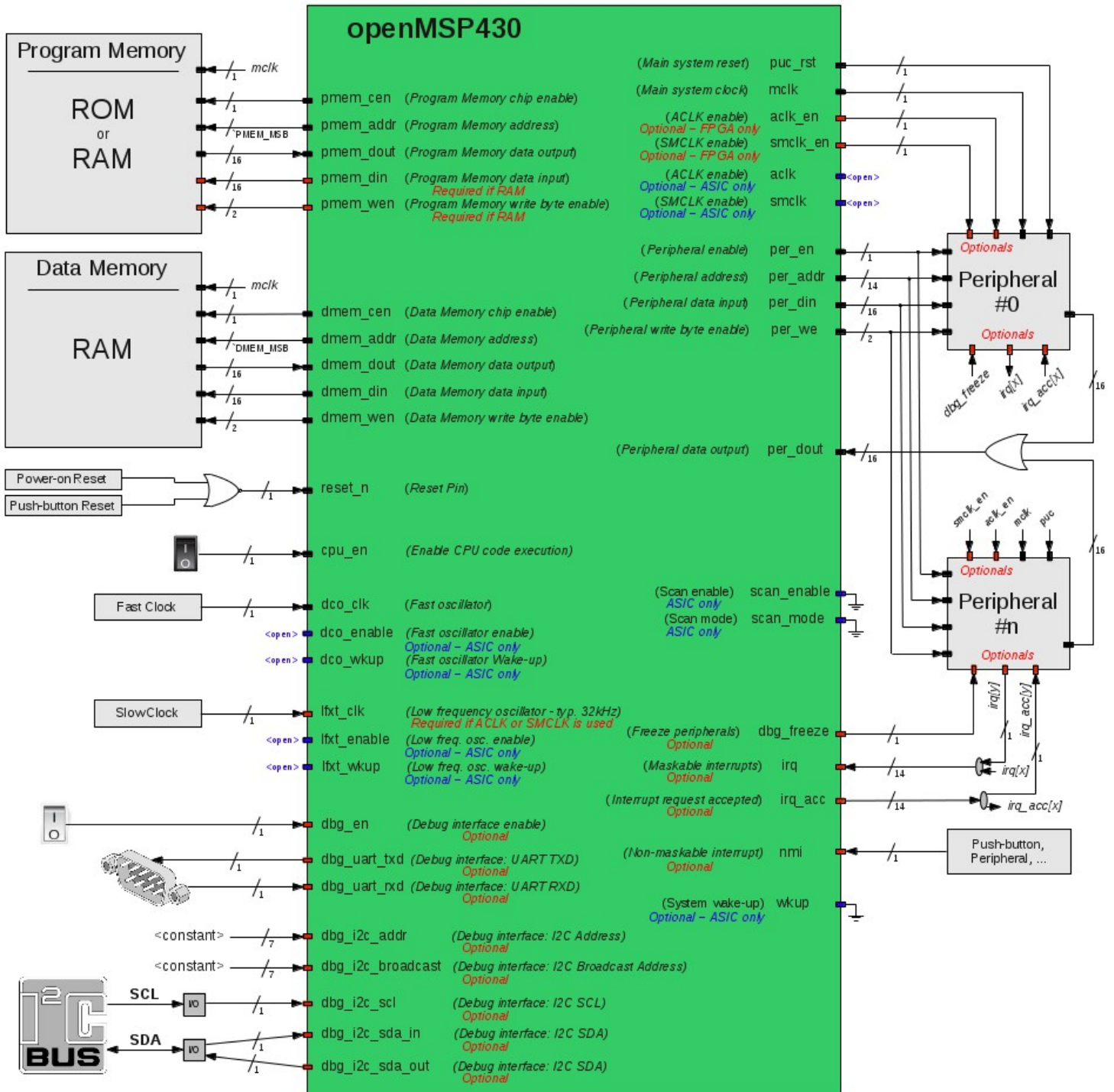
The full pinout of the core is summarized in the following table.

Port Name	Direc-tion	Width	Clock Domain	Description
Clocks				
cpu_en	Input	1	<async> or mclk[4]	Enable CPU code execution (asynchronous and non-glitchy). Set to 1 if unused.
dco_clk	Input	1	-	Fast oscillator (fast clock)
lfxt_clk	Input	1	-	Low frequency oscillator (typ. 32KHz) Set to 0 if unused.
mclk	Output	1	-	Main system clock
aclk_en	Output	1	mclk	FPGA ONLY: ACLK enable
smclk_en	Output	1	mclk	FPGA ONLY: SMCLK enable
dco_enable	Output	1	dco_clk	ASIC ONLY: Fast oscillator enable
dco_wkup	Output	1	<async>	ASIC ONLY: Fast oscillator wakeup (asynchronous)
lfxt_enable	Output	1	lfxt_clk	ASIC ONLY: Low frequency oscillator enable
lfxt_wkup	Output	1	<async>	ASIC ONLY: Low frequency oscillator wakeup (asynchronous)
aclk	Output	1	-	ASIC ONLY: ACLK
smclk	Output	1	-	ASIC ONLY: SMCLK
wkup	Input	1	<async>	ASIC ONLY: System Wake-up (asynchronous and non-glitchy) Set to 0 if unused.
Resets				
puc_rst	Output	1	mclk	Main system reset
reset_n	Input	1	<async>	Reset Pin (active low, asynchronous and non-glitchy)

Program Memory interface				
pmem_addr	Output	`PMEM_AWIDTH[1]	mclk	Program Memory address
pmem_cen	Output	1	mclk	Program Memory chip enable (low active)
pmem_din	Output	16	mclk	Program Memory data input (optional[2])
pmem_dout	Input	16	mclk	Program Memory data output
pmem_wen	Output	2	mclk	Program Memory write byte enable (low active) (optional[2])
Data Memory interface				
dmem_addr	Output	`DMEM_AWIDTH[1]	mclk	Data Memory address
dmem_cen	Output	1	mclk	Data Memory chip enable (low active)
dmem_din	Output	16	mclk	Data Memory data input
dmem_dout	Input	16	mclk	Data Memory data output
dmem_wen	Output	2	mclk	Data Memory write byte enable (low active)
External Peripherals interface				
per_addr	Output	14	mclk	Peripheral address
per_din	Output	16	mclk	Peripheral data input
per_dout	Input	16	mclk	Peripheral data output
per_en	Output	1	mclk	Peripheral enable (high active)
per_we	Output	2	mclk	Peripheral write byte enable (high active)
Interrupts				
irq	Input	14	mclk	Maskable interrupts (one-hot signal)

nmi	Input	1	\<async\> or mclk[4]	Non-maskable interrupt (asynchronous)
irq_acc	Output	14	mclk	Interrupt request accepted (one-hot signal)
Serial Debug interface				
dbg_en	Input	1	\<async\> or mclk[4]	Debug interface enable (asynchronous)[3]
dbg_freeze	Output	1	mclk	Freeze peripherals
dbg_uart_txd	Output	1	mclk	Debug interface: UART TXD
dbg_uart_rxd	Input	1	\<async\>	Debug interface: UART RXD (asynchronous)
dbg_i2c_addr	Input	7	mclk	Debug interface: I2C Address
dbg_i2c_broadcast	Input	7	mclk	Debug interface: I2C Broadcast Address (for multicore only)
dbg_i2c_scl	Input	1	\<async\>	Debug interface: I2C SCL
dbg_i2c_sda_in	Input	1	\<async\>	Debug interface: I2C SDA input
dbg_i2c_sda_out	Output	1	mclk	Debug interface: I2C SDA output
Scan				
scan_enable	Input	1	dco_clk	ASIC ONLY: Scyn enable (active during scan shifting)
scan_mode	Input	1	\<stable\>	ASIC ONLY: Scan mode

[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.

[3]: When disabled, the debug interface is hold into reset (and clock gated in ASIC mode). As a consequence, the **dbg_en** port can be used to reset the debug interface without disrupting the CPU execution.
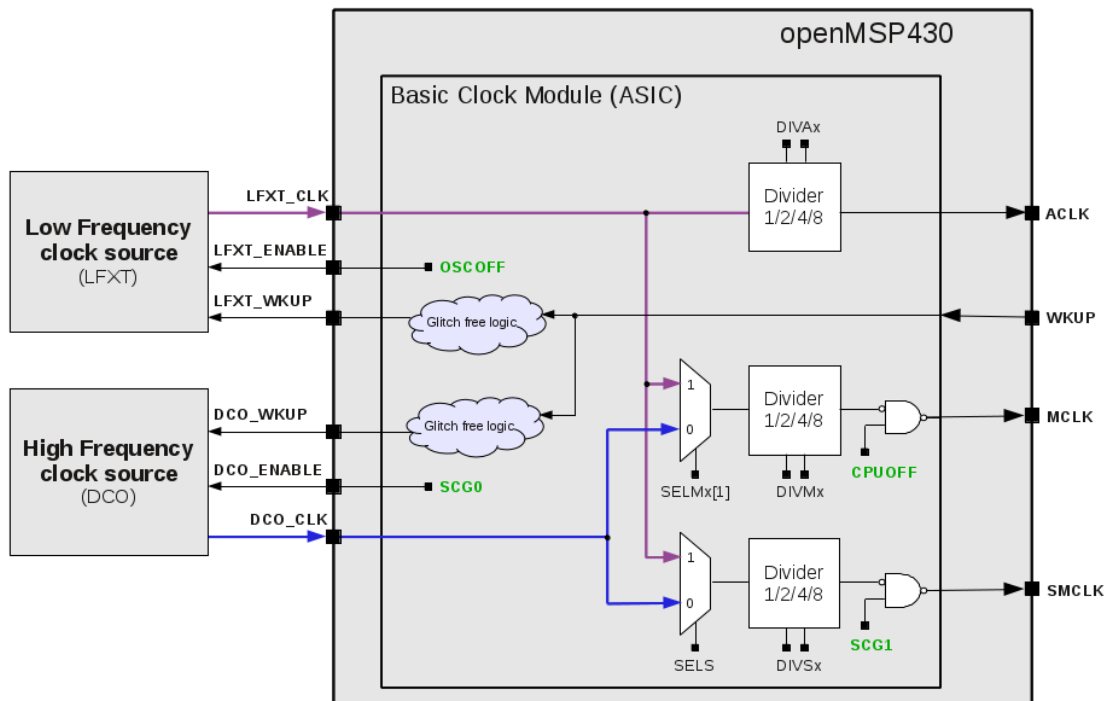
[4]: Clock domain is selectable through configuration in the *openMSP430_defines.v* file (see Advanced System Configuration).

2. Clocks

The different clocks in the design are managed by the Basic Clock Module as following in the FPGA configuration:
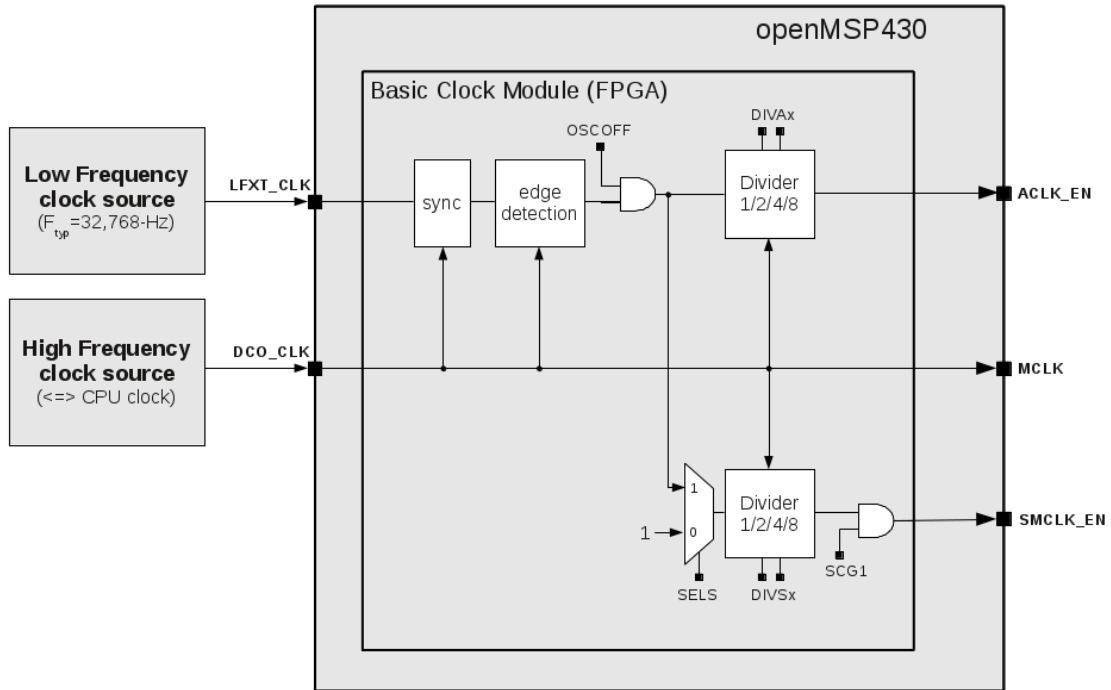


or as following in the ASIC configuration:

- **CPU_EN:** this input port provides a hardware mean to stop or resume CPU execution. When unused, this port should be set to 1.

- **DCO_CLK**: this input port is typically connected to a PLL, RC oscillator or any clock resource the target FPGA/ASIC 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**: in an FPGA system, 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.
  In an ASIC, if ACLK or SMCLK are used and if the clock muxes are included, then this port can be connected to any kind of clock source (it doesn't need to be low-frequency. The name was just kept to be consistent with TI's documentation).

- **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 when the core is targeting an FPGA.
  An example of this can be found in the Watchdog and TimerA modules, where it is implemented as following:
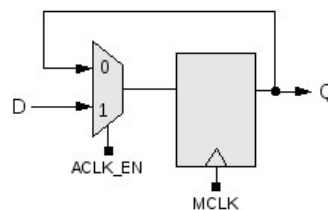
- **ACLK / SMCLK:** ACLK and MCLK are available through these two ports when targeting an ASIC.

- **DCO_ENABLE / DCO_WKUP:** ASIC specific signals controlling the fast clock generator for low power mode support (SCG0 bit in the status register).

- **LFXT_ENABLE / LFXT_WKUP:** ASIC specific signals controlling the low frequency clock generator for low power mode support (OSCOFF bit in the status register).

- **WKUP**: When activated, this signal allows a peripheral to restore all CPU clocks (i.e. wakeup the cpu) prior IRQ generation. Note that IRQs MUST always be generated from the MCLK clock domain.

As an FPGA system 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*.
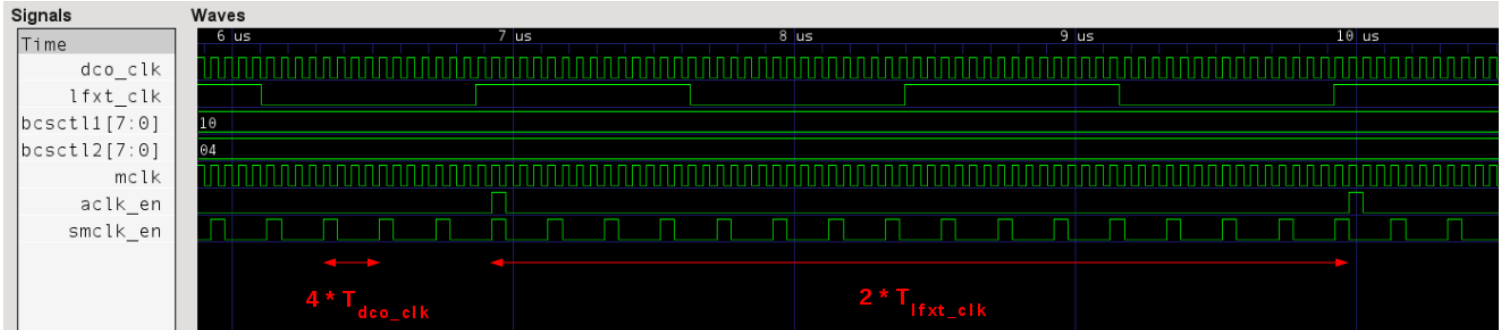
3. Resets

- **RESET_N**: this input port is typically connected to a board push button and is generally combined with the system power-on-reset.

- **PUC_RST**: 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. As a general rule, this signal should be used as the reset of the *MCLK* clock domain.

The following waveform illustrates this:
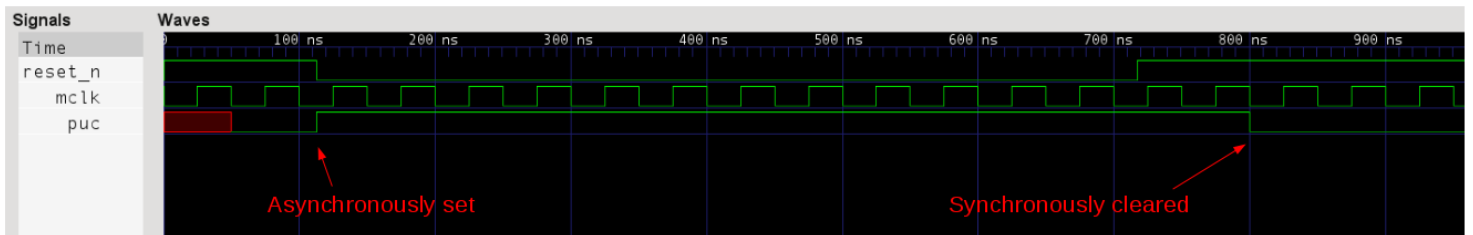
4. Program Memory

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... or a new ROM mask for an ASIC.

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):
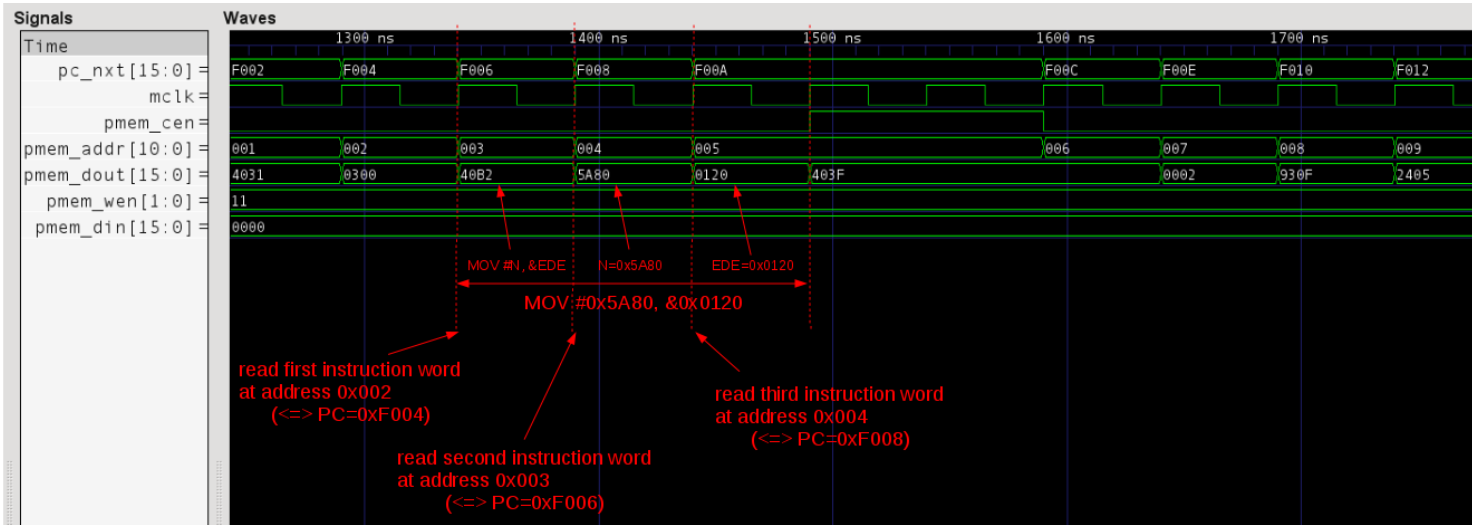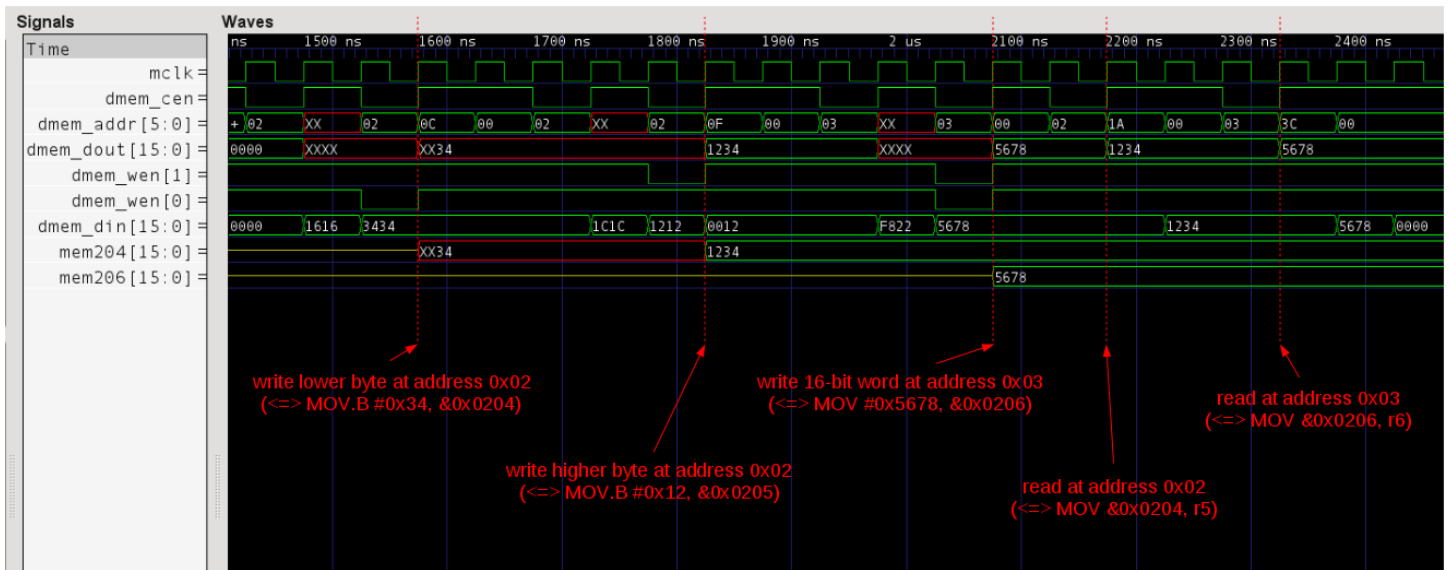

5. Data Memory

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:

Signals		Waves

The waveform shows the following annotations:
- write lower byte at address 0x02 (<=> MOV.B #0x34, &0x0204)
- write higher byte at address 0x02 (<=> MOV.B #0x12, &0x0205)
- write 16-bit word at address 0x03 (<=> MOV #0x5678, &0x0206)
- read at address 0x02 (<=> MOV &0x0204, r5)
- read at address 0x03 (<=> MOV &0x0206, r6)

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 regard 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 afterward 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. It is to be noted that a 14 bit address will always be provided from the openMSP430 to the peripheral in order to accommodate the biggest possible PER_SIZE Verilog configuration option (i.e. 32kB as opposed to 512B by default).
  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_WE[0] will activate a write on the lower byte, PER_WE[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:

56
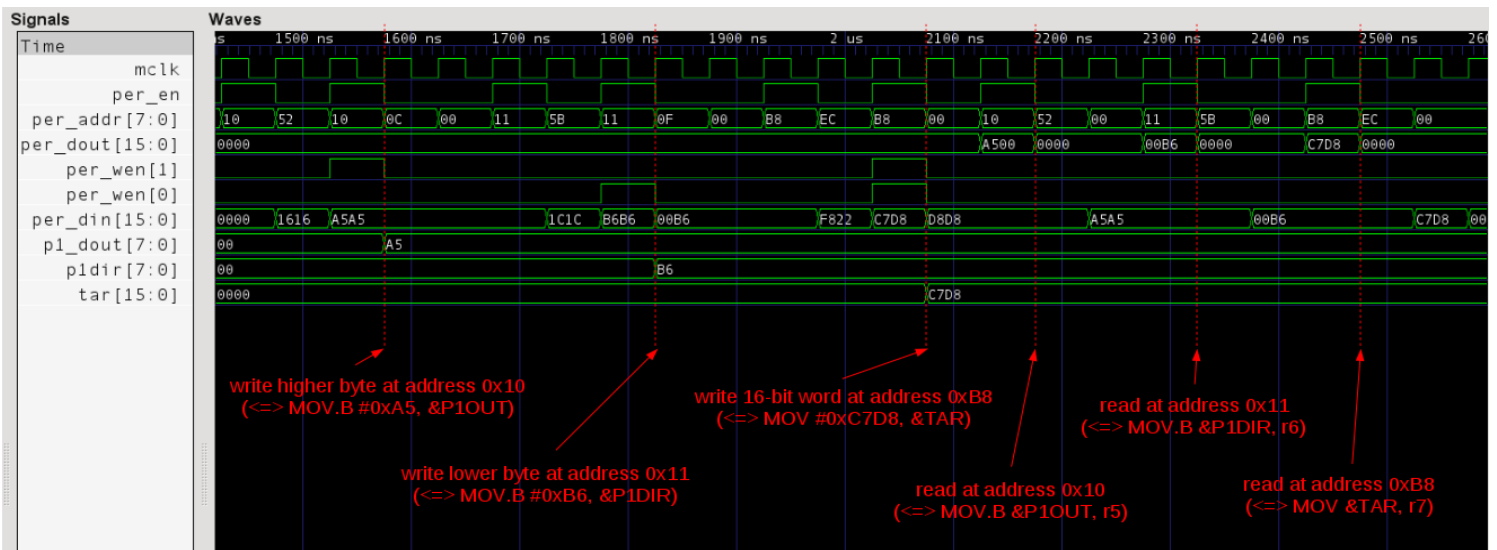
7. Interrupts

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 (typically timers).

- **DBG_EN:** this signal allows the user to enable or disable the serial debug interface without interfering with the CPU execution. It is to be noted that when disabled (i.e. DBG_EN=0), the debug interface is held into reset.

- **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.

8.1 UART Configuration

- **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 :
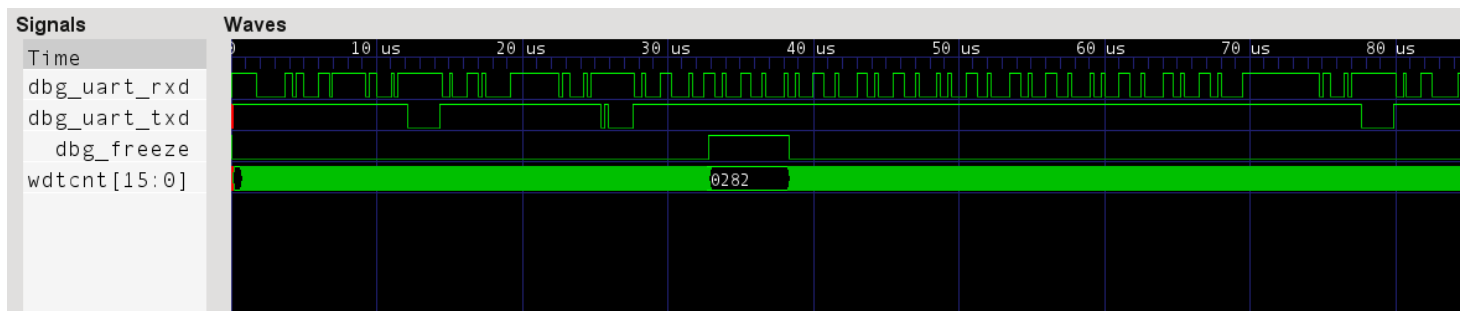
8.2 I2C Configuration

- **DBG_I2C_ADDR**: I2C Device address of the oMSP core (between 8 and 119). In a multi-core configuration, each core has its own address.

- **DBG_I2C_BROADCAST**: I2C Device broadcast address of the oMSP core (between 8 and 119). In a multi-core configuration, all cores have the same broadcast address.

- **DBG_I2C_SCL**: I2C bus clock input (SCL).

- **DBG_I2C_SDA_OUT** / **DBG_I2C_SDA_IN**: these signals are connected to the SDA I/O cell as following:
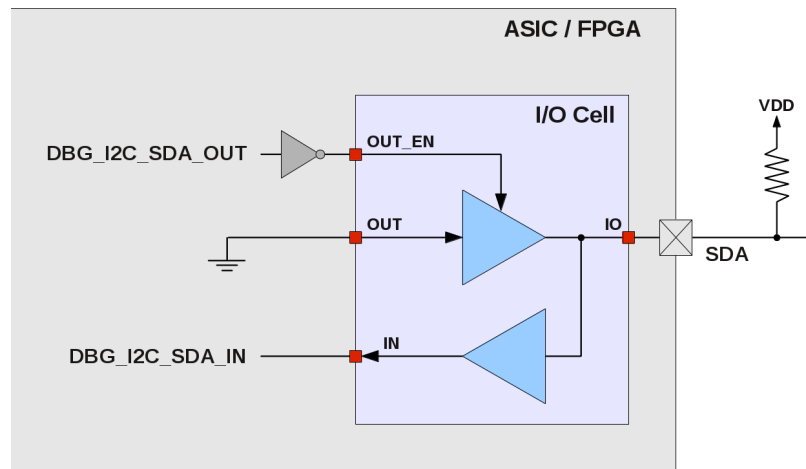
The following waveform shows some communication traffic on the I2C bus:

5.

ASIC Implementation

Table of content

1. Introduction

This section covers specific points of the openMSP430 **ASIC** implementation, in particular:

- The ASIC specific RTL configuration options.
- Some DFT considerations.
- A description of each ASIC sensitive module.

Keep in mind that as no exotic design technique were used in the openMSP430, following a standard implementation flow from Synthesis to P&R is the best way to go.

2. RTL Configuration

Whenever the "`` `define ASIC``" statement of the *Expert System Configuration* section is uncommented, all ASIC specific configuration options are enabled.

2.1 Basic Clock Module

In its ASIC configuration, the Basic clock module of the openMSP430 can support up to all features described in the MSP430x1xx Family User's Guide (Chapter 4). All these options are highlighted in the following diagram and discussed below:
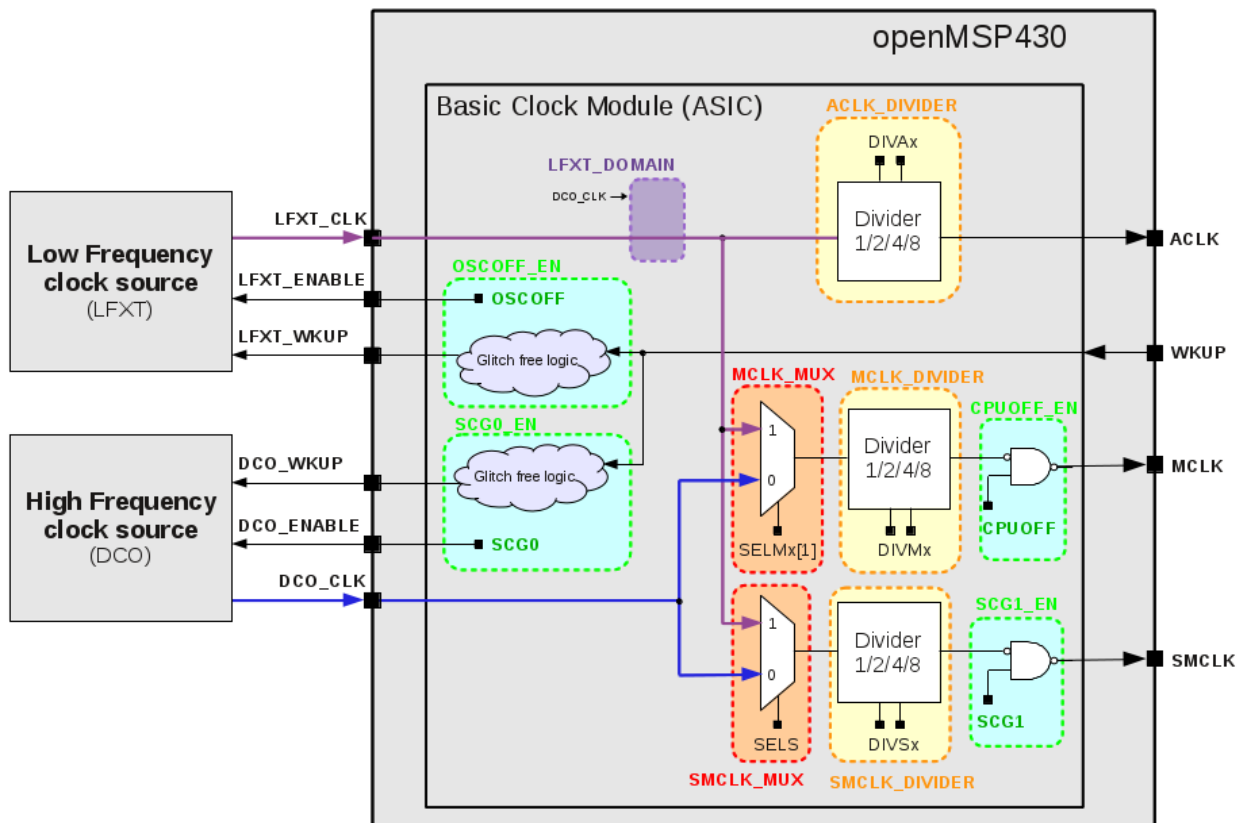
2.1.1 Low-Frequency Clock Domain

The LFXT clock domain can be enabled thanks to the following configuration option:

```
//=======================================================================
// LFXT CLOCK DOMAIN
//=======================================================================

//--------------------------------------------------------------
// When uncommented, this define will enable the lfxt_clk
// clock domain.
// When commented out, the whole chip is clocked with dco_clk.
//--------------------------------------------------------------
`define LFXT_DOMAIN
```

Note 1: When commented-out:

- *ACLK* is running on *DCO_CLK*
- MCLK_MUX and SMCLK_MUX options are not supported
- OSCOFF_EN low power mode is not supported

Note 2: Unlike its name suggest, there is no frequency limitation on *LFXT_CLK*. The name was simply kept in order to be consistent with the original MSP430 documentation, where *LFXT_CLK* is typically connected to a 32 kHz crystal oscillator.

2.1.2 Clock Muxes

The *MCLK* and *SMCLK* clock muxes can be enabled or disabled with the following options:

```
//=======================================================================
// CLOCK MUXES
//=======================================================================

//--------------------------------------------------------
// MCLK: Clock Mux
//--------------------------------------------------------
// When uncommented, this define will enable the
// MCLK clock MUX allowing the selection between
// DCO_CLK and LFXT_CLK with the BCSCTL2.SELMx register.
// When commented, DCO_CLK is selected.
//--------------------------------------------------------
`define MCLK_MUX

//--------------------------------------------------------
// SMCLK: Clock Mux
//--------------------------------------------------------
// When uncommented, this define will enable the
// SMCLK clock MUX allowing the selection between
// DCO_CLK and LFXT_CLK with the BCSCTL2.SELS register.
// When commented, DCO_CLK is selected.
//--------------------------------------------------------
`define SMCLK_MUX
```

Note 1: When a MUX is excluded, the concerned clock (*MCLK* and/or *SMCLK)* is running with *DCO_CLK*.

Note 2: If a MUX is included, the implementation and sign-off tools (in particular CTS and STA) must be aware that a new clock needs to be defined on the MUX output.

2.1.3 Clock Dividers

The *MCLK*, *SMCLK* and ACLK clock dividers can be enabled or disabled with the following options:

```
//=========================================================================
// CLOCK DIVIDERS
//=========================================================================

//----------------------------------------------------------
// MCLK: Clock divider
//----------------------------------------------------------
// When uncommented, this define will enable the
// MCLK clock divider (/1/2/4/8)
//----------------------------------------------------------
`define MCLK_DIVIDER

//----------------------------------------------------------
// SMCLK: Clock divider (/1/2/4/8)
//----------------------------------------------------------
// When uncommented, this define will enable the
// SMCLK clock divider
//----------------------------------------------------------
`define SMCLK_DIVIDER

//----------------------------------------------------------
// ACLK: Clock divider (/1/2/4/8)
//----------------------------------------------------------
// When uncommented, this define will enable the
// ACLK clock divider
//----------------------------------------------------------
`define ACLK_DIVIDER
```

The clock dividers instantiate a clock gate on the clock tree and are implemented as following:
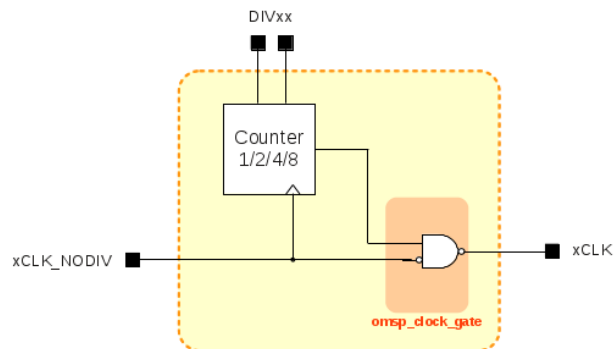
2.1.4 Low-Power Modes

2.1.4.1 Internal clocks (MCLK / SMCLK)

Two bit fields in the status register (R2) allow to control the system clocks:

- **CPUOFF** allows to switch-off *MCLK*
- **SCG1** allows to switch-off *SMCLK*

These control bits are supported by the openMSP430 and can be included in the design with the following defines:

```
//=========================================================================
// LOW POWER MODES
//=========================================================================

//--------------------------------------------------------
// LOW POWER MODE: CPUOFF
//--------------------------------------------------------
// When uncommented, this define will include the
// clock gate allowing to switch off MCLK in
// all low power modes: LPM0, LPM1, LPM2, LPM3, LPM4
//--------------------------------------------------------
`define CPUOFF_EN

//--------------------------------------------------------
// LOW POWER MODE: SCG1
//--------------------------------------------------------
// When uncommented, this define will include the
// clock gate allowing to switch off SMCLK in
// the following low power modes: LPM2, LPM3, LPM4
//--------------------------------------------------------
`define SCG1_EN
```

In order to keep the clock tree as flat as possible, the CPUOFF and SCG1 low power options share the same clock gate with the clock divider:

2.1.4.2 Clock oscillators (DCO_CLK / LFXT_CLK)

There are two bit fields in the status register (R2) allowing to control the clock oscillators:

- **SCG0** allows to switch-off the DCO oscillator
- **OSCOFF** allows to switch-off the LFXT oscillator
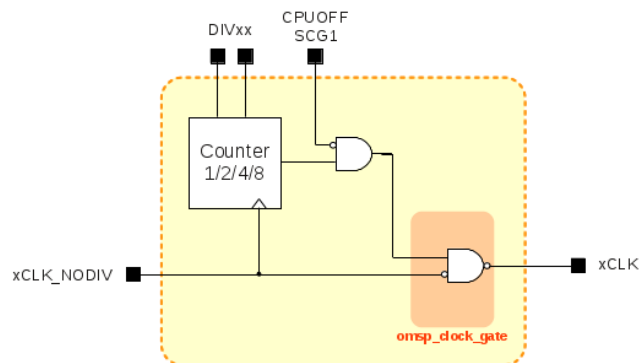
These control bits are supported by the openMSP430 and can be included in the design with the following defines:

```
//=======================================================================
// LOW POWER MODES
//=======================================================================

//-------------------------------------------------------
// LOW POWER MODE: SCG0
//-------------------------------------------------------
// When uncommented, this define will enable the
// DCO_ENABLE/WKUP port control (always 1 when commented).
// This allows to switch off the DCO oscillator in the
// following low power modes: LPM1, LPM3, LPM4
//-------------------------------------------------------
`define SCG0_EN

//-------------------------------------------------------
// LOW POWER MODE: OSCOFF
//-------------------------------------------------------
// When uncommented, this define will include the
// LFXT_CLK clock gate and enable the LFXT_ENABLE/WKUP
// port control (always 1 when commented).
// This allows to switch off the low frequency oscillator
// in the following low power modes: LPM4
//-------------------------------------------------------
`define OSCOFF_EN
```

The control logic of both DCO and LFXT oscillators is identical.

When disabled, the ***_WKUP** signal is used to asynchronously wake up the oscillator. Once the oscillator is awake (and therefore a clock is available), the ***_ENABLE** signal will take over and synchronously keep the oscillator enabled until the CPU clears the SCG0 or OSCOFF bit again.

The following two waveforms illustrate the CPU entering the LPM1 mode, and in particular the DCO oscillator being switched-off:

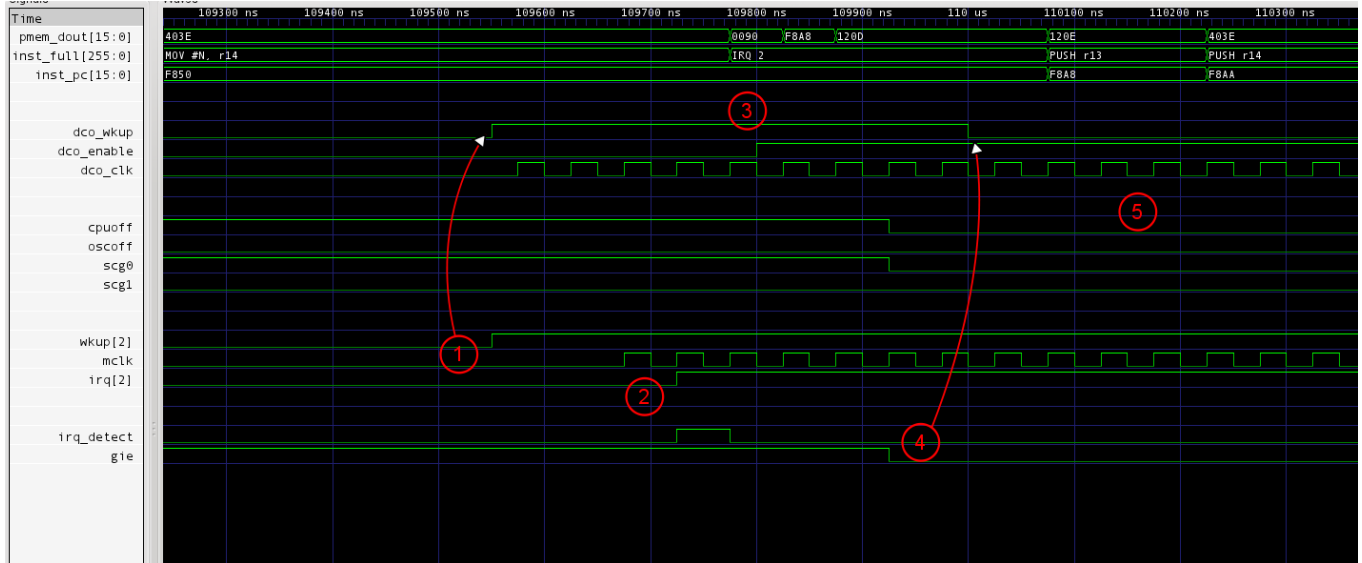- Entering LPM1 through a **BIS #N, R2** instruction:

- Entering LPM1 through a **RETI** instruction:

Note: the DCO oscillator is enabled until the BIS and RETI instruction are fully executed (i.e. until the CPU state machines reach their IDLE state).

At last, this waveform shows the CPU going out of LPM1 mode and in particular the DCO oscillator wake-up sequence:

In order to wake-up the CPU from ANY low power mode, the system **MUST ALWAYS** go through the following chain of events (as illustrated in the previous waveform):

1. The peripheral (for example a timer) asserts the *WKUP* input of the openMSP430 in order to asynchronously restore the clocks. At this stage, *DCO_WKUP* is activated and *DCO_ENABLE* is still cleared.
2. Once MCLK is available, the peripheral generates a synchronous IRQ signal in order to re-activate the CPU state machines.
3. The CPU state machines activated, *DCO_ENABLE* is synchronously set.
4. When the global interrupt enable flag (GIE) is cleared, *DCO_WKUP* is released two clock cycles later (i.e. same behavior as a reset synchronizer).
   Important note: the peripheral should release the *WKUP* input when its interrupt pending flag is cleared. Otherwise the *DCO_WKUP* signal will be set again as soon as the GIE flag is restored by the RETI instruction... which is probably not the intended behavior :-P
5. The DCO oscillator is now enabled until SCG0 is set again.

2.2 Other configuration options

2.2.1 Fine Grained Clock Gating

Nowadays, all synthesis tools support automatic (fine grained) clock gating insertion. However, as some design houses still prefer to have the clock gates directly instantiated in the RTL, there is the possibility to include the *'manual'* fine grained clock gates in the design with the following define:

```
//=======================================================================
// FINE GRAINED CLOCK GATING
//=======================================================================

//---------------------------------------------------------
// When uncommented, this define will enable the fine
// grained clock gating of all registers in the core.
//---------------------------------------------------------
`define CLOCK_GATING
```

2.2.2 Watchdog Clock Mux

The watchdog clock mux allows to select between *ACLK* and *SMCLK*. It can be enabled or disabled with the **WATCHDOG_MUX** define.
When excluded, the additional **WATCHDOG_NOMUX_ACLK** option allows the user to decide if the watchdog clock should be hard-wired to *ACLK* (if uncommented) or *SMCLK* (if commented-out)

```
//=======================================================================
// CLOCK MUXES
//=======================================================================

//---------------------------------------------------------
// WATCHDOG: Clock Mux
//---------------------------------------------------------
// When uncommented, this define will enable the
// Watchdog clock MUX allowing the selection between
// ACLK and SMCLK with the WDTCTL.WDTSSEL register.
// When commented out, ACLK is selected if the
// WATCHDOG_NOMUX_ACLK define is uncommented, SMCLK is
// selected otherwise.
//---------------------------------------------------------
`define WATCHDOG_MUX
//`define WATCHDOG_NOMUX_ACLK
```

3. DFT Considerations

The openMSP430 is designed to be fully scan friendly. During production, the ATE controls the core through the *scan_mode* and *scan_enable* signals. The *scan_mode* port is always asserted during scan testing and is used to switch between functional and scan mode.

3.1 Resets

When in scan mode (i.e. *scan_mode* input port is set), **ALL** internal resets of the openMSP430 are connected the *reset_n* input port.
Taking the *POR* generation as an example, it is implemented using the **omsp_scan_mux** module as following:
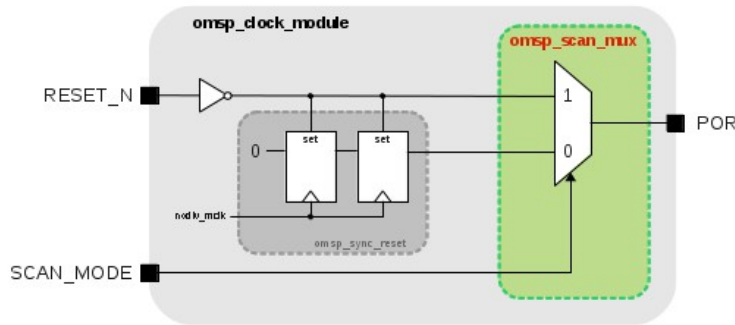
3.2 Clock Gates

When in scan mode (i.e. *scan_mode* input port is set), **ALL** clock gates instantiated in the design must be enabled during scan shifting. This is can be achieved by setting the *scan_enable* input port during the shift phase.
On the other hand, during the capture phase, the *scan_enable* port must be cleared in order to restore the functional behavior of the clock gate.

This feature is implemented in the **omsp_clock_gate** module as following:
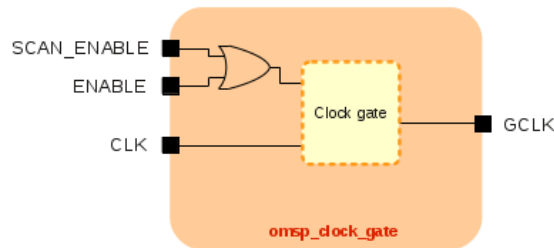
3.3 Clock Muxes

When in scan mode (i.e. *scan_mode* input port is set), the **MCLK** and **SMCLK** clock muxes are both running on **DCO_CLK**. The watchdog mux is running **SMCLK** (i.e. DCO_CLK).

This feature is implemented in the **omsp_clock_mux** module as following:
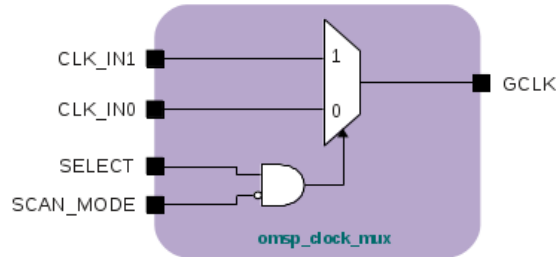
Note: if the LFXT clock domain is enabled, the **LFXT_CLK** input port should also be connected to the scan clock when in scan mode.

3.4 Coverage

After synthesizing the openMSP430 in its maximum configuration (in particular with ALL clock domains available and ALL clock muxes included), the core reaches **99.7%** stuck-at fault coverage:

4. Sensitive Modules

ALL modules discussed in this section have a simple and well defined functionality but nonetheless lay on sensitive parts of the design (clock tree, wake-up path, ...).

In the industry, it is common place for companies to have policies recommending designers to use textbook structures or specific standard cells when implementing circuits considered as 'sensitive'.
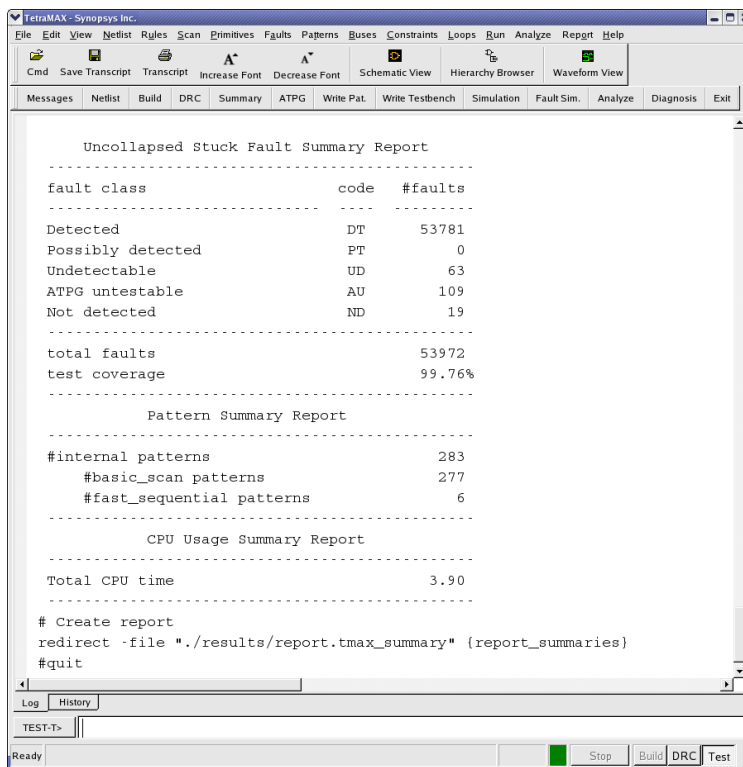This section will hopefully help to quickly identify these 'sensitive' circuits and adapt them to your requirements if necessary.

4.1 AND Gate (*omsp_and_gate.v*)

This module implements a simple AND2 gate and is instantiated several times on the wake-up paths in order to ensure a glitch free generation of the wake-up signals. The idea behind this block is to prevent the synthesis tool from optimizing the combinatorial wake-up path and potentially generate a glitchy logic.

There are three different ways to handle this block:

1. Do nothing
2. Modify the RTL by directly instantiating an AND2 cell from the target library and applying a *don't touch* or *size only* attribute on it before proceeding to the synthesis compile step
3. Keep the RTL unchanged and when running synthesis, first compile this module separately before going to the top down compile (don't forget the *don't touch* or *size only* attribute)

Note that the first option is actually acceptable because in low power mode, there are no clocks available, which means no glitch... However, in active mode, the wake-up line could see a lot of glitches, which is functionally not a problem (since the core is awake anyway) but could be considered as not really elegant...

4.2 Clock Gate (*omsp_clock_gate.v*)

Almost every company has a different policy for handling clock gates. Therefore, this module is probably the most likely to be modified.

So here are the facts:

- There are only rising edge flip-flop in the design[1]
  → as a consequence clock gates can indifferently park the clock high or low without affecting functionality.
- The enable signal of ALL clock gates in the openMSP430 are generated with the rising edge of the clock
  → this leaves the door open for both LATCH and NAND2 based clock gates.

[1]: *beside for the DCO_ENABLE and LFXT_ENABLE signals and the clock MUXes. However, these can be safely ignored*

As a consequence, you can feel free to use:

- A LATCH based clock gate. For example:
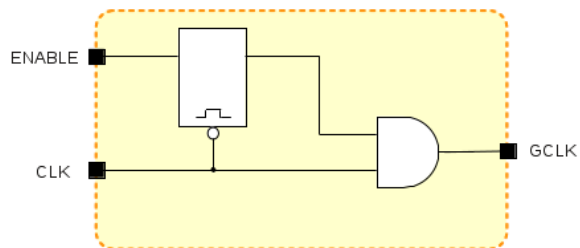
- Or a NAND2 based clock gate:
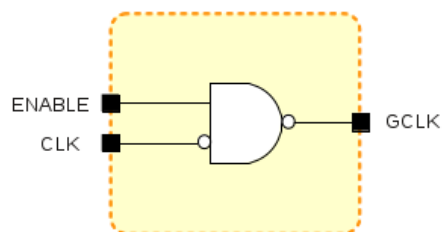
4.3 Clock Mux (*omsp_clock_mux.v*)

The clock muxes of the openMSP430 are implemented as following:
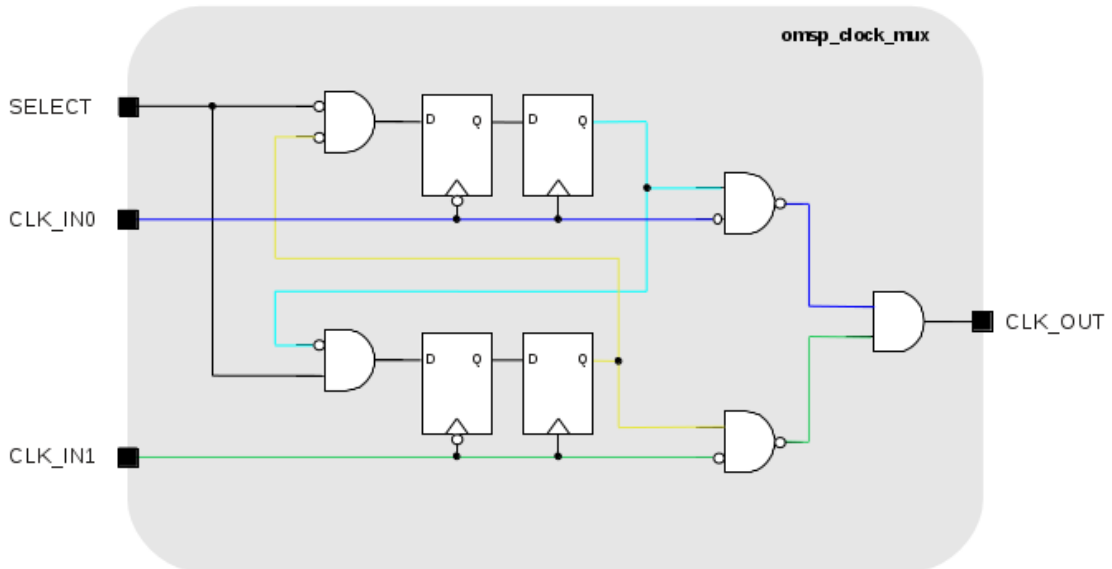
In order to make this implementation 100% bullet proof, the RTL could be modified by manually instantiating the NAND2 and AND2 cells directly from the target library (with the associated ***don't touch*** or ***size only*** attributes of course).

However, if you decide to compile this module as it is, the synthesis tool should normally be smart enough and not mess it up (but PLEASE PLEASE PLEASE double check manually the resulting gate netlist).

4.4 Scan Mux (*omsp_scan_mux.v*)

As illustrated in the section 3.1, the scan mux cell allows **ALL** internal resets to be controllable with the *reset_n* input port in scan mode.
In addition, the scan mux is also used by the *omsp_wakeup_cell* (see section 4.7 below).

4.5 Sync Cell (*omsp_sync_cell.v*)

The following synchronization cell is instantiated on all clock domain crossing data paths:
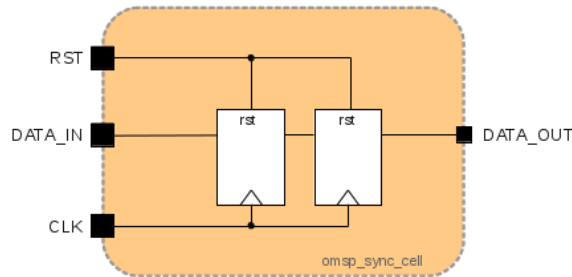
4.6 Sync Reset (*omsp_sync_reset.v*)

Internal resets are generated using the following standard reset synchronizer:
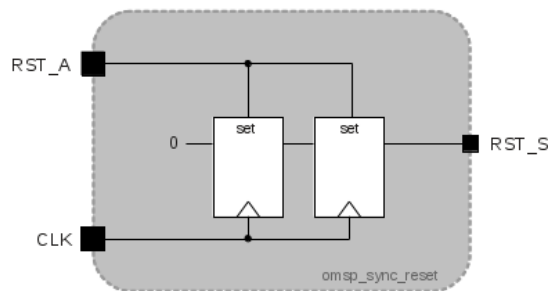
4.7 Wakeup Cell (*omsp_wakeup_cell.v*)

The wakeup cell is the most unconventional module of the openMSP430 design as it contains a flip-flop whose clock and reset are both coming from a data path.
In the openMSP430 core, it is instantiated a single time in the watchdog timer but can also be reused in external custom peripherals.

The implementation of the block looks as following:
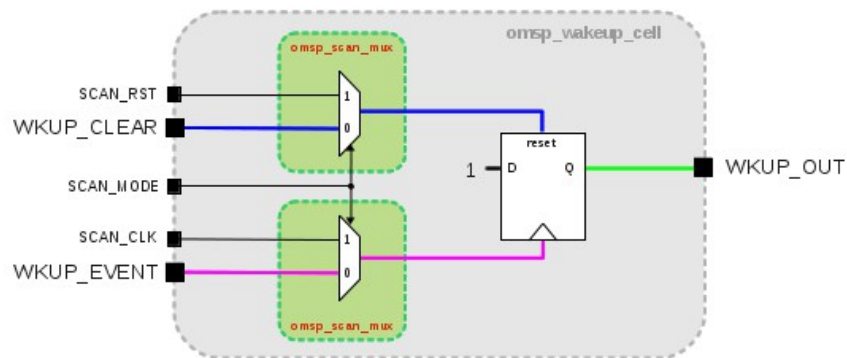
The basic idea here is simply to set the WKUP_OUT signal with a rising edge on the WKUP_EVENT port, and clear it when WKUP_CLEAR is active (i.e. level sensitive clear).

In order to give a better perspective from a system point of view, the following diagram shows how the wakeup cell has been used in the particular case of the watchdog timer (note that WDTIFG_CLR_REG and WDTQN_EDGE_REG are both output of a flip-flop and therefore glitch-free):
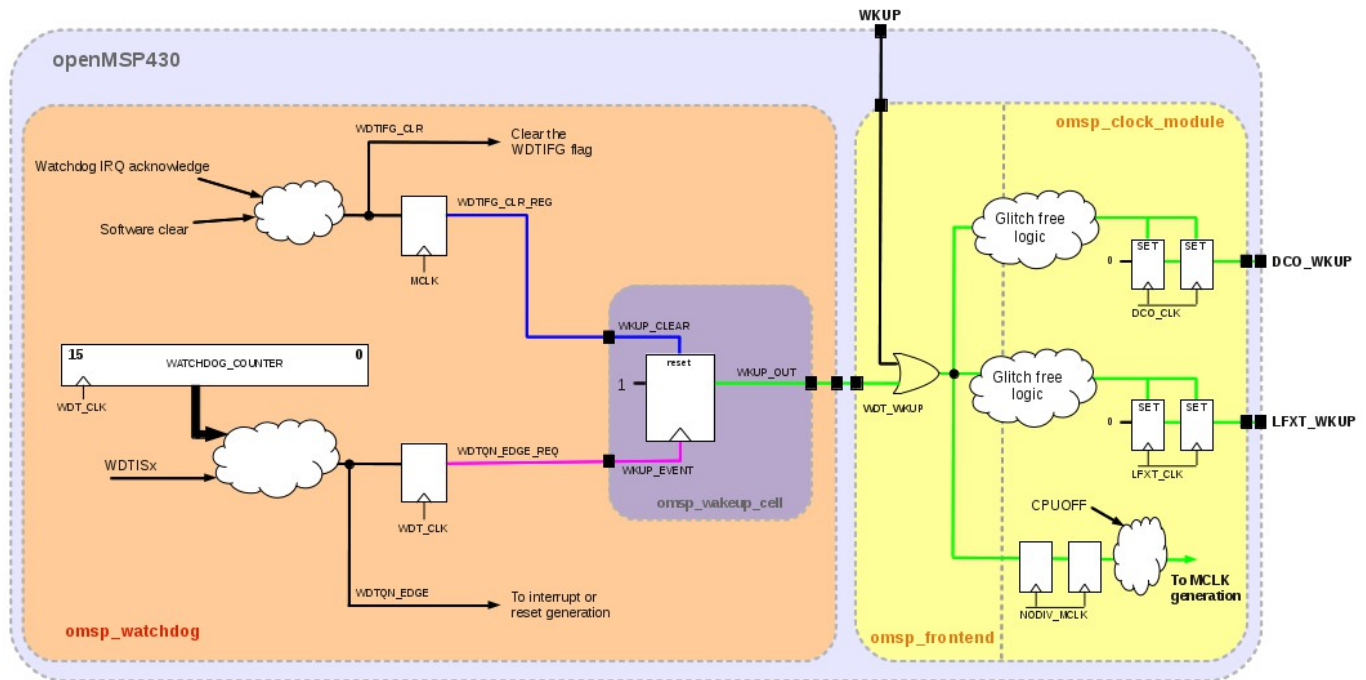
Note: Wake-up signals can of course be generated in a different way as long as they directly come from a flip-flop (or are certified to be non-glitchy).
For example a simple handshake between the WDT_CLK and MCLK clock domains could have been used to clear the WDT_WKUP signal in a fully synchronous manner. However, it is to be noted that this handshake would introduce some synchronization delay, which might not be negligible if MCLK and WDT_CLK frequencies are orders of magnitude apart (i.e. several MHz for MCLK and 32kHz for WDT_CLK).
As getting the oscillators back to sleep as fast as possible might prove to be extremely important for low-power designs, this asynchronous solution was selected for the *omsp_watchdog* implementation.

6.

Area and Speed Analysis

Table of content

Warning: 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

Manu-facturer	Devices	Info	Utilization			
			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

Process	Target Frequency	Info	Area			
			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

Detailed results can be found in the PDF documentation (see the online download section).

7.

Software Development Tools

Table of content

1. Introduction

Building on the serial debug interface capabilities provided by the openMSP430, three 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/Vista/7).

Note: to be able to execute the scripts, TCL/TK needs to be installed on your system.

In order to connect the host PC to the openMSP430 serial debug interface, a UART or I2C serial cable/adapter is required.

Typically, the following solutions will suit any kind of development board:

UART	I²C
USB to RS232 converter: USB to Serial TTL converter:	Devantech USB-ISS adapter:

2. openmsp430-loader

This simple program allows the user to load the openMSP430 program memory with an executable file (ELF or Intel-HEX 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:

```
USAGE   : openmsp430-loader.tcl [-device    <communication port>]
                                [-adaptor   <adaptor type>]
                                [-speed     <communication speed>]
                                [-i2c_addr <cpu address>]            <elf/ihex-file>

DEFAULT : <communication port>  = /dev/ttyUSB0
          <adaptor type>        = uart_generic
          <communication speed> = 115200 (for UART) / I2C_S_100KHZ (for I2C)
          <core address>        = 42

EXAMPLES: openmsp430-loader.tcl -device /dev/ttyUSB0 -adaptor uart_generic -speed 9600  leds.elf

          openmsp430-loader.tcl -device COM2:        -adaptor i2c_usb-iss  -speed I2C_S_100KHZ
                                -i2c_addr 75 ta_uart.ihex
```

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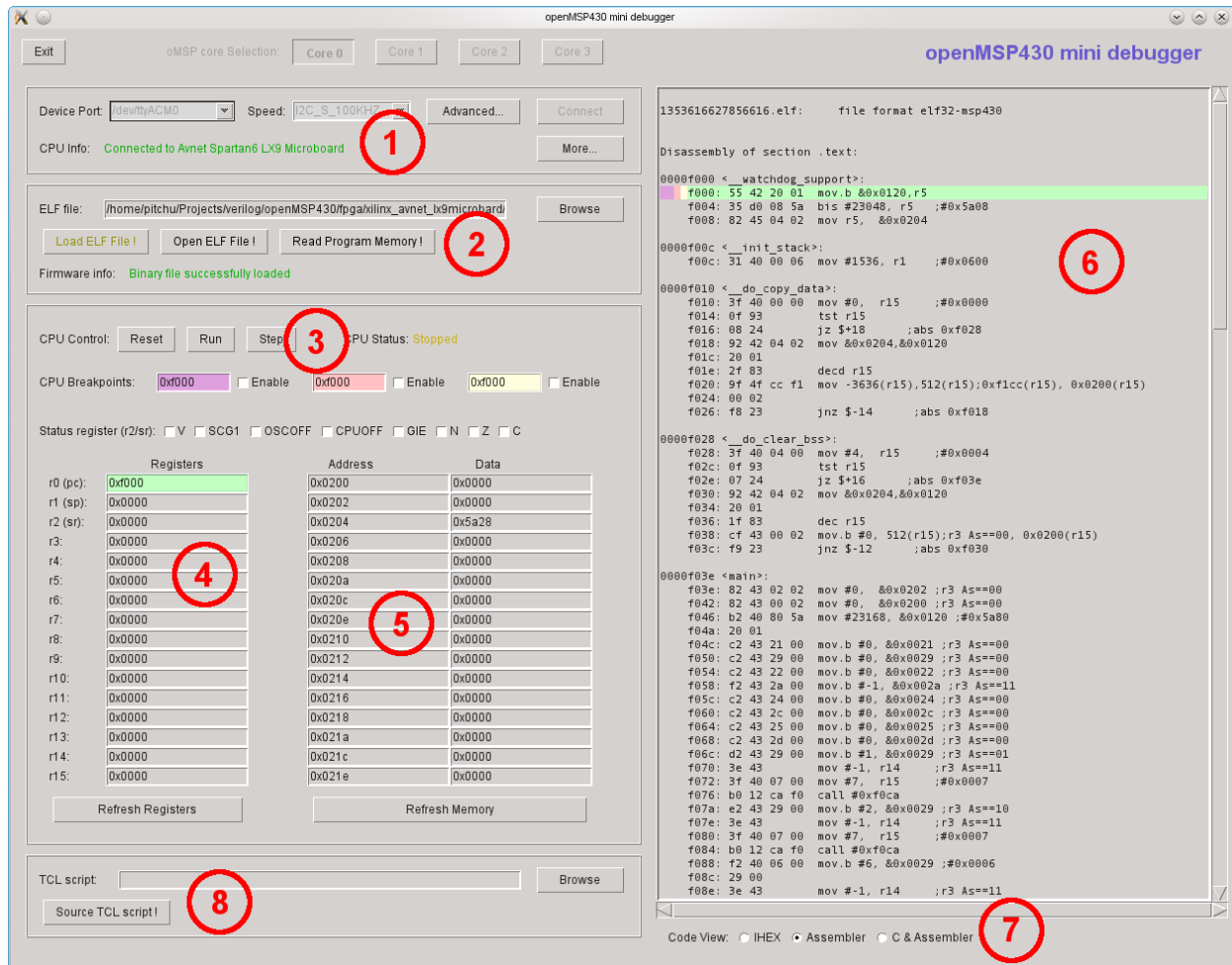
As you can see from the screenshot, it allows the following actions:

- **(1)** Connect to the openMSP430 Serial Debug Interface
- **(2)** Load the program memory with an ELF or Intel-HEX file
- **(3)** Control the CPU: Reset, Stop, Start and Single-Step and Software breakpoints
- **(4)** Read/Write access of the CPU registers
- **(5)** Read/Write access of the whole memory range (program, data, peripherals)
- **(6)** Basic disassembled view of the loaded program (current PC location is highlighted in green, software breakpoints in yellow, pink and violet)
- **(7)** Choose the disassembled view type
- **(8)** Source a custom external TCL script.

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:
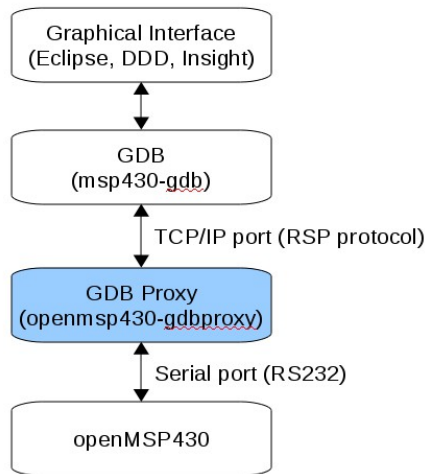
Like the original '***msp430-gdbproxy***' program, '***openmsp430-gdbproxy***' can be controlled from the command line. However, it also provides a simple graphical interface:
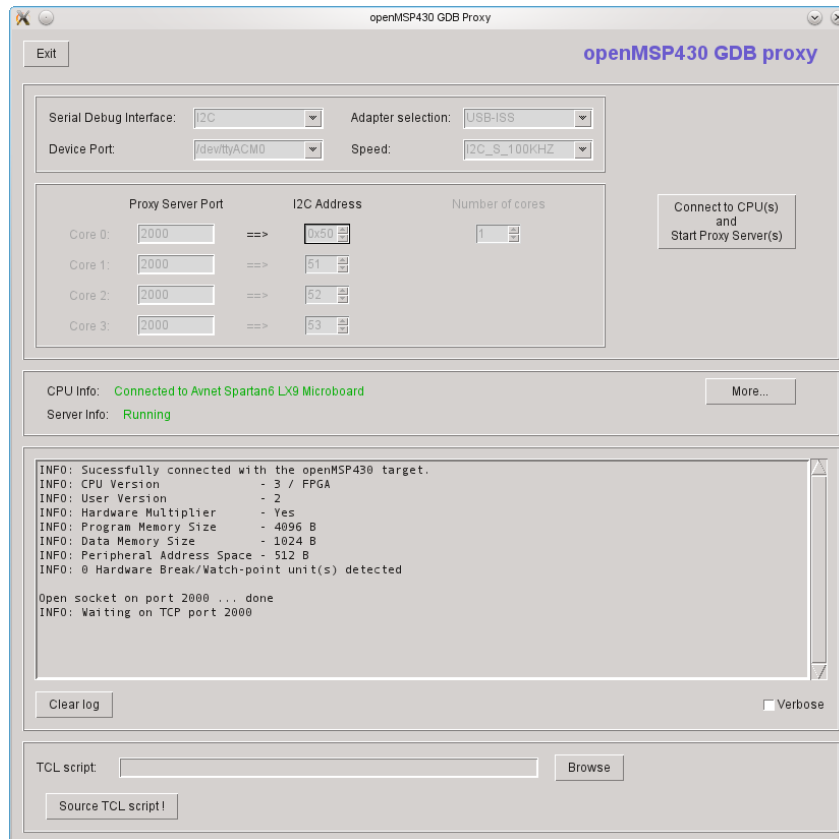
These two additional screenshots show the script in action together with the Eclipse and DDD graphical frontends:
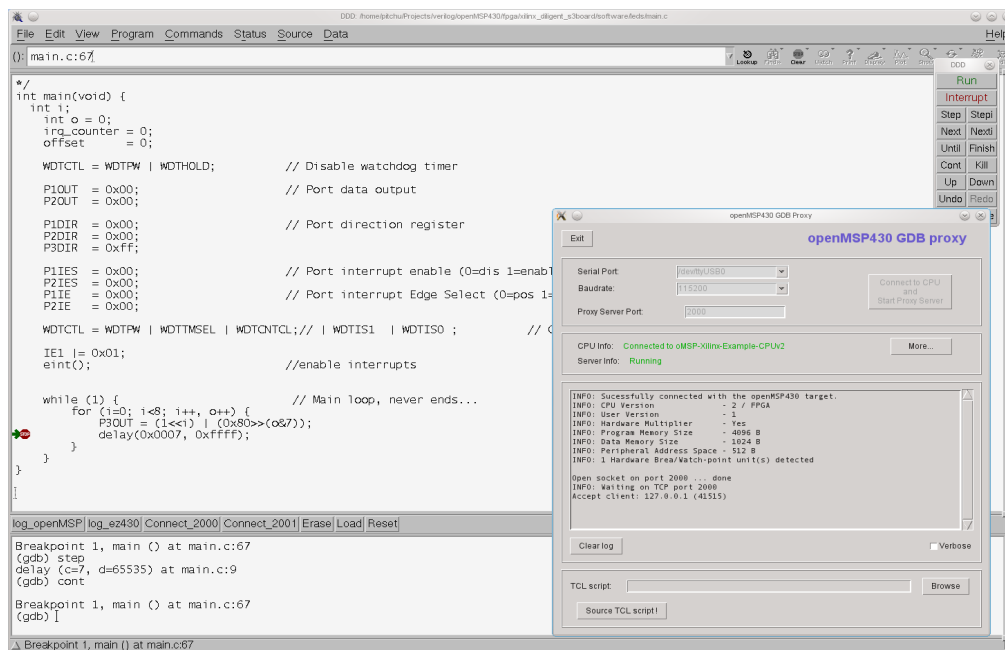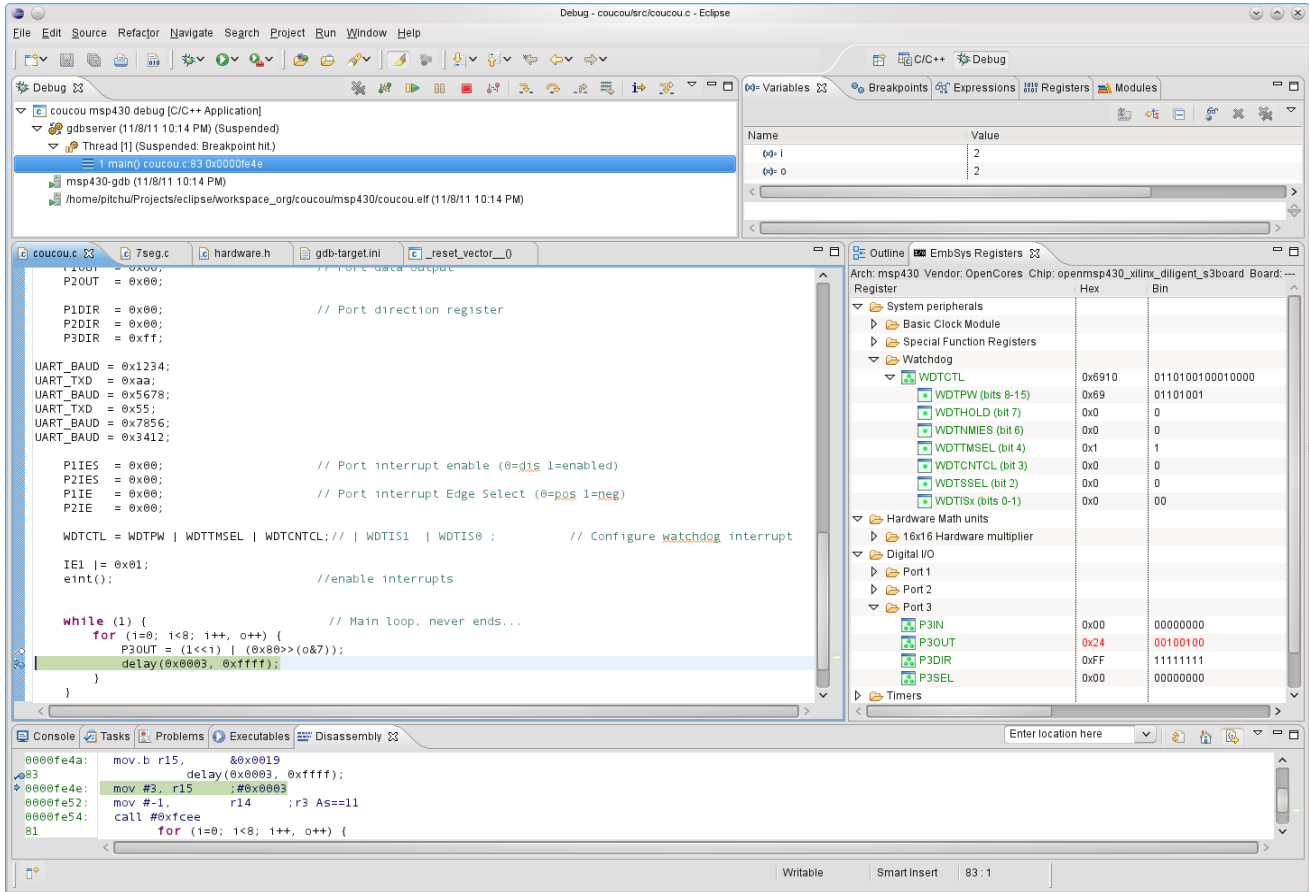
Tip 1: There are several tutorials on Internet explaining how to configure Eclipse for the MSP430. As an Eclipse newbie, I found the followings quite helpful (the *msp430-gdbproxy* sections should of course be ignored as we are using our own o*penmsp430-gdbproxy* program :-)):

- [A Step By Step Guide To MSP430 Programming Under Linux](#) (English)
- [MSP430 eclipse helios mspgcc4](#) (German)

Tip 2: You probably want to install this excellent Eclipse plugin (see screenshot above):

- **[EmbSysRegView](#)**

5. MSPGCC Toolchain

5.1 Compiler options

The **msp430-gcc** compiler accepts the following MSP430 specific command line parameters (copied from the MSPGCC [manual page](#)):

-mmcu=	Specify the MCU name
-mno-volatile-workaround	Do not perform a volatile workaround for bitwise operations.
-mno-stack-init	Do not initialize the stack as *main()*starts.
-minit-stack=	Specify the initial stack address.
-mendup-at=	Jump to the specified routine at the end of *main()*.
-mforce-hwmul	Force use of a hardware multiplier.
-mdisable-hwmul	Do not use the hardware multiplier.
-minline-hwmul	Issue inline code for 32-bit integer operations for devices with a hardware multiplier.
-mnoint-hwmul	Do not disable and enable interrupts around hardware multiplier operations. This makes multiplication faster when you are certain no hardware multiplier operations will occur at deeper interrupt levels.
-mcall-shifts	Use subroutine calls for shift operations. This may save some space for shift intensive applications.

5.2 MCU selection

The following table aims to help selecting the proper MCU name for the **-mmcu** <u>option</u> during 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	60 kB	2048 B	**Yes**
msp430x149	60 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: the program memory size should imperatively match the openMSP430 configuration.

5.3 Custom linker script

The use of the **-mmcu** switch is of course **NOT** mandatory. It is simply a convenient way to use the pre-existing linker scripts provided with the MSPGCC4 toolchain.

However, if the peripheral address space is larger than the standard 512B of the original MSP430 (see the Advanced System Configuration section), a customized linker script **MUST** be provided.

To create a custom linker script, the simplest way is to start from an existing one:

- The MSPGCC(4) toolchain provides a wide range of examples for all supported MSP430 models (see "*msp430/lib/ldscripts/*" sub-directory in the MSPGCC(4) installation directory).

- The openMSP430 project also provide a simple linker script example: ldscript_example.x

From there, the script can be modified to match YOUR openMSP430 configuration:

- In the *text (rx)* section definition, update the ***ORIGIN*** and ***LENGTH*** fields to match the ***PROGRAM MEMORY*** configuration.

- In the *data (rwx)* section definition, update the ***ORIGIN*** field to match the ***PERIPHERAL SPACE*** configuration and the ***LENGTH*** field to match the ***DATA MEMORY*** configuration.

- At last, update the stack pointer initialization value (look for the "***PROVIDE (__stack =***" section) and make sure that it falls in the data memory space (the stack size should also matches your application requirements, i.e. not too small... and not too big :-P).

<div style="text-align: right">

8.

</div>

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
			dbg_i2c_tasks.v	I2C tasks for the serial debug interface
			io_cell.v	Generic I/O cell model for building the serial debug interface I2C bus
			msp_debug.v	Testbench instruction decoder and ASCII chain generator for easy debugging
			timescale.v	Global time scale definition for simulation.
	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
			omsp_dbg_i2c.v	Serial Debug Interface I2C communication block
			omsp_sync_cell.v	Simple synchronization module (double flip-flop).

				omsp_sync_reset.v		*Generic Reset synchronizer (double flip-flop).*
				omsp_clock_gate.v		*Generic Clock gate (NAND2 or LATCH-AND based).*
				omsp_clock_mux.v		*Standard Clock Mux (used in the clock module & watchdog timer).*
				omsp_and_gate.v		*AND gate module used on sensitive glitch free data paths.*
				omsp_wakeup_cell.v		*Generic Wake-up module.*
				omsp_scan_mux.v		*Scan MUX.*
			periph			***Peripherals directory***
				omsp_gpio.v		*Digital I/O (Port 1 to 6)*
				omsp_timerA_defines.v		*Timer A configuration file*
				omsp_timerA_undefines.v		*Timer A Verilog 'undef file*
				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 for assembler vector sources (located in the **src** directory)*
				msp430sim_c		*Main simulation script for C vector sources (located in the **src-c** directory).*
				asm2ihex.sh		*Assembly file compilation (Intel HEX file generation)*
				ihex2mem.tcl		*Verilog program memory file generation*
				rtlsim.sh		*Verilog Icarus simulation script*
				template.x		*ASM linker definition file template*
				template_defs.asm		*Common ASM definition file included in all ".s43" files*
				omsp_config.sh		*oMSP configuration file.*
				parse_results		*Script parsing regression log files and generating summary report.*
				cov_*		*Code coverage scripts for NC-Verilog and*

						ICM
				run		***For running RTL simulations***
					run	*Run single simulation of a given assembler vector*
					run_c	*Run single simulation of a given C vector*
					run_all	*Run regression of all vectors*
					run_all_mpy	*Run regression of all hardware multiplier vectors (!!! very long simulation time !!!)*
					run_disassemble	*Disassemble the program memory content of the latest simulation*
					run_coverage_analysis	*Performs the coverage report merging of the regression run and starts ICM for the analysis.*
					load_waveform.sav	*SAV file for gtkWave*
				src		***RTL simulation vectors sources (ASM based)***
					ldscript_example.x	*MSPGCC toolchain linker script example*
					submit.prj	*ISIM simulator verilog command file*
					submit.f	*Verilog simulator command file*
					core.f	*Command file listing the CPU files only.*
					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*
					nmi.s43	*NMI assembler vector files*
					nmi.v	*NMI verilog stimulus vector files*
					cpu_startup_asic.s43	*CPU startup assembler vector files*
					cpu_startup_asic.v	*CPU startup 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
				lp_modes_*.s43	*Low Power modes assembler vector files*
				lp_modes_*.v	*Low Power modes verilog stimulus vector files*
				dbg_*.s43	*Serial Debug Interface assembler vector files*
				dbg_*.v	*Serial Debug Interface verilog stimulus vector files*
				sfr.s43	*SFR assembler vector files*
				sfr.v	*SFR 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*
				scan.s43	*Scan test assembler vector files*
				scan.v	*Scan test verilog stimulus vector files*
			src-c		***RTL simulation vectors sources (C based)***
				coremask_v1.0	*CoreMark benchmark*
				dhrystone_v2.1	*Dhrystone benchmark ("official" version)*
				dhrystone_4mcu	*Dhrystone benchmark (MCU adapted)*
				sandbox	*Small playground :-)*
	synthesis				***Top level synthesis directory***
		synopsys			*Synopsys (Design Compiler) directory*
			run_syn		*Run synthesis*
			run_tmax		*Run ATPG*
			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
			tmax.tcl	Main TetraMax (ATPG) script
			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
				timescale.v	Global time scale definition for simulation.
		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	

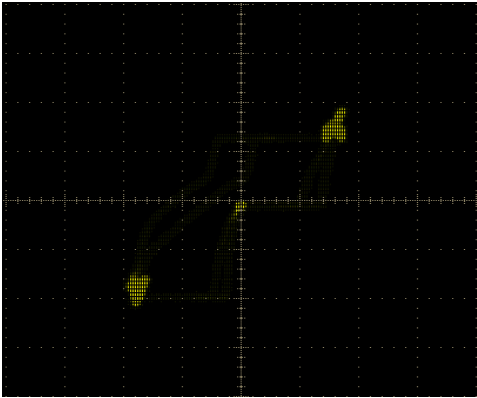							7seg.c	
					ta_uart		*Software UART with Timer_A (from the CDK4MSP project)*
		synthesis					***Top level synthesis directory***
			xilinx				
				0_create_bitstream.sh			*Run Xilinx ISE synthesis in a Linux environment*
				1_initialize_pmem.sh			*Update bitstream's program memory with a given software ELF file*
				2_generate_prom_file.sh			*Generate PROM file*
				3_program_fpga.sh			*Program FPGA and on-board flash memory*
				bitstreams			
					*.bit		*Bitstream files*
					*.mcs		*PROM files*
					README.jpg		*README file*
				scripts			
					ihex2mem.tcl		*TCL script converting Intel-HEX format to Verilog memory file.*
					impact_generate_prom_file.batch		*iMPACT TCL script for PROM file generation.*
					impact_program_fpga.batch		*iMPACT TCL script for programing FPGA and on-board flash memory.*
					memory.bmm		*FPGA memory description for bitstream's program memory update*
					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.*

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..*
				timescale.v	*Global time scale definition for simulation.*
		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*

3.3 Actel ProASIC3 example

fpga				***openMSP430 FPGA Projects top level directory***
	actel_m1a3pl_dev_kit			***Actel FPGA Project based on the ProASIC3 M1A3PL development kit***
		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*
			dbg_uart_tasks.v	*UART tasks for the serial debug interface.*
			timescale.v	*Global time scale definition for simulation.*
			proasic3l.v	*Actel ProASIC3L library file.*
			DAC121S101.v	*Verilog model of National's DAC121S101 12 bit DAC*
		doc		***Diverse documentation***
			M1A3PL_DEV_KIT_QS.pdf	*Development Kit Quickstart Card*
			M1IGLOO_StarterKit_v1_5_UG.pdf	*Development Kit User's Guide*
		rtl		***RTL sources***
			verilog	
			openMSP430_fpga.v	*FPGA top level file*
			dac_spi_if.v	*SPI interface to National's DAC121S101 12 bit DAC*
			openmsp430	***Local copy of the openMSP430 core.*** *The *define.v file has been adjusted to the requirements of the project.*
			smartgen	*Xilinx's coregen directory*
			dmem_128B.v	*128 Byte RAM (for data memory)*
			pmem_2kB.v	*2 kByte RAM (for program memory)*
		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*
					spacewar		*SpaceWar oscilloscope game.*
				synthesis			*Top level synthesis directory*
					actel		
						prepare_implementation.tcl	*Generate required files prior synthesis and P&R.*
						synplify.tcl	*Synplify template for the synthesis run.*
						libero_designer.tcl	*Libero Designer template for the P&R run.*
						design_files.v	*RTL file list to be synthesized*
						design_constraints.pre.sdc	*Synthesis timing constraints.*

				design_constraints.post.sdc	*P&R timing constraints.*
				design_constraints.pdc	*P&R physical constraints.*

4. Directory structure: Software Development Tools

tools			*openMSP430 Software Development Tools top level directory*
	omsp_alias.xml		*This XML file allows the software development tools to identify a openMSP430 implementation, and add customized extra information (Alias, URL, ...).*
	bin		***Contains the main TCL scripts (and the windows executable files if generated)***
		openmsp430-loader.tcl	*Simple command line boot loader*
		openmsp430-minidebug.tcl	*Minimalistic debugger with simple GUI*
		openmsp430-gdbproxy.tcl	*GDB Proxy server to be used together with MSP430-GDB and the Eclipse, DDD, or Insight graphical front-ends*
		README.TXT	*README file regarding the use of TCL scripts in a Windows environment.*
	lib		***Common library***
		tcl-lib	***Common TCL library***
			dbg_uart_generic.tcl — *Low level Generic UART communication functions*
			dbg_i2c_usb-iss.tcl — *Low level I2C communication functions for the USB-ISS adapter*
			dbg_utils.tcl — *Low level "COMx:" "/dev/tty" 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)*
			xml.tcl — *Simple XML parser (from Keith Vetter).*
	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_Protoc ol.pdf	*Document from Jeremy Bennett (Embecosm): Howto: GDB Remote Serial Protocol - Writing a RSP Server*