Multi-Processor System on Chip with MSP430

QueenField

1. INTRODUCTION

1.1. OPEN SOURCE PHILOSOPHY

For Windows Users!

1. Settings → Apps → Apps & features → Related settings, Programs and Features → Turn Windows features on or off → Windows Subsystem for Linux
2. Microsoft Store → INSTALL UBUNTU

type:

```
sudo apt update
sudo apt upgrade
```

1.2.1. Open Source Hardware

1.2.1.1. MSP430 Processing Unit

1.2.1.2. OpenRISC Processing Unit

1.2.1.3. RISC-V Processing Unit

1.2.2. Open Source Software

1.2.2.1. MSP430 GNU Compiler Collection

1.2.2.2. OpenRISC GNU Compiler Collection

1.2.2.3. RISC-V GNU Compiler Collection

1.2. MSP430 ISA

1.2.1. ISA Bases

1.2.2.1. MSP430 32

1.2.2.2. MSP430 64

1.2.2.3. MSP430 128

1.2.2. ISA Extensions

1.2.3. ISA Modes

1.2.3.1. MSP430 User

1.2.3.2. MSP430 Supervisor

1.2.3.3. MSP430 Hypervisor

1.2.3.4. MSP430 Machine

2. PROJECTS

2.1. CORE-MSP430

2.1.1. Definition

2.1.2. RISC Pipeline

2.1.3. CORE-MSP430 Organization

Core	Module description
`msp430_core`	
`...msp430_bcm`	
`...msp430_frontend`	
`...msp430_execution`	
`.....msp430_register_file`	
`.....msp430_alu`	
`...msp430_memory`	
`...msp430_sfr`	
`...msp430_watchdog`	
`...msp430_multiplier`	
`...msp430_dbg`	
`.....msp430_dbg_hwbrk`	
`.....msp430_dbg_uart`	
`.....msp430_dbg_i2c`	

2.1.4. Parameters

2.1.4.1. Basic System Configuration

Description	Parameter	Type	Default
Program Memory Size	PMEM_SIZE	integer	16384
Data Memory Size	DMEM_SIZE	integer	4096

Description	Parameter	Type	Default
Include/Exclude Hardware Multiplier	MULTIPLYING	bit	1
Include/Exclude Serial Debug interface	DBG_ON	bit	1

2.1.4.2. Advanced System Configuration (for experienced users)

Description	Parameter	Type	Default
Peripheral Memory Space	PER_SIZE	integer	512
Custom user version number	USER_VERSION	bit vector	0
Watchdog timer	WATCHDOG	bit	1
Non-Maskable-Interrupt support	NMI_EN	bit	1
Number of available IRQs	IRQ_16	bit	1
Number of available IRQs	IRQ_32	bit	0
Number of available IRQs	IRQ_64	bit	0
Input synchronizers	SYNC_NMI	bit	1
Input synchronizers	SYNC_CPU_EN	bit	0
Input synchronizers	SYNC_DBG_EN	bit	0
Debugger definition	DBG_RST_BRK_EN	bit	0

2.1.4.3. Expert System Configuration (experts only)

Description	Parameter	Type	Default
Hardware breakpoint/watchpoint units	DBG_HWBRK	bit vector	1
Select serial debug interface protocol	DBG_UART	bit	0
Select serial debug interface protocol	DBG_I2C	bit	1
I2C broadcast address	DBG_I2C_BROADCASTC	bit	1
Hardware breakpoint RANGE mode	HWBRK_RANGE	bit	1
ASIC version	ASIC	bit	1

2.1.4.4. ASIC System Configuration (experts/professionals only)

Description	Parameter	Type	Default
LOW POWER MODE: SCG	SCG_EN	bit vector	1
FINE GRAINED CLOCK GATING	CLOCK_GATING	bit	1
ASIC CLOCKING	ASIC_CLOCKING	bit	1
LFXT CLOCK DOMAIN	LFXT_DOMAIN	bit	1
MCLK: Clock Mux	MCLK_MUX	bit	1
SMCLK: Clock Mux	SMCLK_MUX	bit	1
WATCHDOG: Clock Mux	WATCHDOG_MUX	bit	1
WATCHDOG: Clock No-Mux	WATCHDOG_NOMUX_ACLK	bit	0
MCLK: Clock divider	MCLK_DIVIDER	bit	1
SMCLK: Clock divider (/1/2/4/8)	SMCLK_DIVIDER	bit	1
ACLK: Clock divider (/1/2/4/8)	ACLK_DIVIDER	bit	1
LOW POWER MODE: CPUOFF	CPUOFF_EN	bit	1
LOW POWER MODE: OSCOFF	OSCOFF_EN	bit	1

2.1.4.5. System Constants (do not edit)

Description	Parameter	Type	Default
Program Memory Size	PMEM_AWIDTH	integer	13
Data Memory Size	DMEM_AWIDTH	integer	11
Peripheral Memory Size	PER_AWIDTH	integer	8
Data Memory Base Adresses	DMEM_BASE	integer	N
Program Memory	PMEM_MSB	integer	N
Data Memory	DMEM_MSB	integer	N
Peripheral Memory	PER_MSB	integer	N
Number of available IRQs	IRQ_NR	integer	16
Instructions type	INST_SOC	integer	0
Instructions type	INST_JMPC	integer	1
Instructions type	INST_TOC	integer	2
Single-operand arithmetic	RRC	integer	0
Single-operand arithmetic	SWPB	integer	1
Single-operand arithmetic	RRA	integer	2
Single-operand arithmetic	SXTC	integer	3
Single-operand arithmetic	PUSH	integer	4
Single-operand arithmetic	CALL	integer	5
Single-operand arithmetic	RETI	integer	6
Single-operand arithmetic	IRQX	integer	7
Conditional jump	JNE	integer	0
Conditional jump	JEQ	integer	1
Conditional jump	JNC	integer	2
Conditional jump	JC	integer	3
Conditional jump	JN	integer	4
Conditional jump	JGE	integer	5
Conditional jump	JL	integer	6
Conditional jump	JMP	integer	7
Two-operand arithmetic	MOV	integer	0
Two-operand arithmetic	ADD	integer	1
Two-operand arithmetic	ADDC	integer	2
Two-operand arithmetic	SUBC	integer	3
Two-operand arithmetic	SUBB	integer	4
Two-operand arithmetic	CMP	integer	5
Two-operand arithmetic	DADD	integer	6
Two-operand arithmetic	BITC	integer	7
Two-operand arithmetic	BIC	integer	8
Two-operand arithmetic	BIS	integer	9
Two-operand arithmetic	XORX	integer	10
Two-operand arithmetic	ANDX	integer	11
Addressing modes	DIR	integer	0
Addressing modes	IDX	integer	1
Addressing modes	INDIR	integer	2
Addressing modes	INDIR_I	integer	3
Addressing modes	SYMB	integer	4
Addressing modes	IMM	integer	5
Addressing modes	ABSC	integer	6
Addressing modes	CONST	integer	7
Instruction state machine	I_IRQ_FETCH	bit vector	000
Instruction state machine	I_IRQ_DONE	bit vector	001
Instruction state machine	I_DEC	bit vector	010
Instruction state machine	I_EXT1	bit vector	011
Instruction state machine	I_EXT2	bit vector	100

Description	Parameter	Type	Default
Instruction state machine	I_IDLE	bit vector	101
Execution state machine	E_SRC_AD	bit vector	X5
Execution state machine	E_SRC_RD	bit vector	X6
Execution state machine	E_SRC_WR	bit vector	X7
Execution state machine	E_DST_AD	bit vector	X8
Execution state machine	E_DST_RD	bit vector	X9
Execution state machine	E_DST_WR	bit vector	XA
Execution state machine	E_EXEC	bit vector	XB
Execution state machine	E_JUMP	bit vector	XC
Execution state machine	E_IDLE	bit vector	XD
Execution state machine	E_IRQ_0	bit vector	X2
Execution state machine	E_IRQ_1	bit vector	X1
Execution state machine	E_IRQ_2	bit vector	X0
Execution state machine	E_IRQ_3	bit vector	X3
Execution state machine	E_IRQ_4	bit vector	X4
ALU control signals	ALU_SRC_INV	integer	0
ALU control signals	ALU_INC	integer	1
ALU control signals	ALU_INC_C	integer	2
ALU control signals	ALU_ADD	integer	3
ALU control signals	ALU_AND	integer	4
ALU control signals	ALU_OR	integer	5
ALU control signals	ALU_XOR	integer	6
ALU control signals	ALU_DADD	integer	7
ALU control signals	ALU_STAT_7	integer	8
ALU control signals	ALU_STAT_F	integer	9
ALU control signals	ALU_SHIFT	integer	10
ALU control signals	EXEC_NO_WR	integer	11
Debug interface	DBG_UART_WR	integer	18
Debug interface	DBG_UART_BW	integer	17
Debug interface CPU_CTL register	HALT	integer	0
Debug interface CPU_CTL register	RUN	integer	1
Debug interface CPU_CTL register	ISTEP	integer	2
Debug interface CPU_CTL register	SW_BRK_EN	integer	3
Debug interface CPU_CTL register	FRZ_BRK_EN	integer	4
Debug interface CPU_CTL register	RST_BRK_EN	integer	5
Debug interface CPU_CTL register	CPU_RST	integer	6
Debug interface BRKx_CTL register	BRK_MODE_RD	integer	0
Debug interface BRKx_CTL register	BRK_MODE_WR	integer	1
Debug interface BRKx_CTL register	BRK_EN	integer	2
Debug interface BRKx_CTL register	BRK_I_EN	integer	3
Debug interface BRKx_CTL register	BRK_RANGE	integer	4
Basic clock module: BCSCTL2 Control Register	SELMX	integer	7
Basic clock module: BCSCTL2 Control Register	SELS	integer	3
MCLK Clock gate	MCLK_CGATE	bit	1
SMCLK Clock gate	SMCLK_CGATE	bit	1
Debug interface: CPU version	CPU_VERSION	bit vector	010
Debug interface: Software breakpoint opcode	DBG_SWBRK_OP	bit vector	X4343
UART interface auto data synchronization	DBG_UART_AUTO_SYNC	bit	1
Counter width for the debug interface UART	DBG_UART_XFER_CNT_W	integer	16
Debug UART interface data rate	DBG_UART_BAUD	integer	2000000
Debug UART interface data rate	DBG_DCO_FREQ	integer	20000000
Debug UART interface data rate	DBG_UART_CNT	integer	N

Description	Parameter	Type	Default
Debug UART interface data rate	DBG_UART_CNTB	bit vector	N
Debug interface input synchronizer	SYNC_DBG_UART_RXD	bit	1
MULTIPLIER CONFIGURATION	MPY_16X16	bit	1

2.1.5. Instruction Inputs/Outputs Bus

2.1.6. Data Inputs/Outputs Bus

2.2. PU-MSP430

2.1. CORE-MSP430

2.1.1. Functionality

2.1.1.1. Organization

Core	Module description
`msp430_core`	
`...msp430_bcm`	
`...msp430_frontend`	
`...msp430_execution`	
`.....msp430_register_file`	
`.....msp430_alu`	
`...msp430_memory`	
`...msp430_sfr`	
`...msp430_watchdog`	
`...msp430_multiplier`	
`...msp430_dbg`	
`.....msp430_dbg_hwbrk`	
`.....msp430_dbg_uart`	
`.....msp430_dbg_i2c`	

2.1.1.2. Pipeline

2.1.2. Interface

2.1.1.1. Constants

2.1.1.1.1. Basic System Configuration

Description	Parameter	Type	Default
Program Memory Size	PMEM_SIZE	integer	16384
Data Memory Size	DMEM_SIZE	integer	4096
Include/Exclude Hardware Multiplier	MULTIPLYING	bit	1
Include/Exclude Serial Debug interface	DBG_ON	bit	1

2.1.1.1.2. Advanced System Configuration (for experienced users)

Description	Parameter	Type	Default
Peripheral Memory Space	PER_SIZE	integer	512
Custom user version number	USER_VERSION	bit vector	0
Watchdog timer	WATCHDOG	bit	1
Non-Maskable-Interrupt support	NMI_EN	bit	1
Number of available IRQs	IRQ_16	bit	1
Number of available IRQs	IRQ_32	bit	0
Number of available IRQs	IRQ_64	bit	0
Input synchronizers	SYNC_NMI	bit	1
Input synchronizers	SYNC_CPU_EN	bit	0
Input synchronizers	SYNC_DBG_EN	bit	0
Debugger definition	DBG_RST_BRK_EN	bit	0

2.1.1.1.3. Expert System Configuration (experts only)

Description	Parameter	Type	Default
Hardware breakpoint/watchpoint units	DBG_HWBRK	bit vector	1
Select serial debug interface protocol	DBG_UART	bit	0
Select serial debug interface protocol	DBG_I2C	bit	1
I2C broadcast address	DBG_I2C_BROADCASTC	bit	1
Hardware breakpoint RANGE mode	HWBRK_RANGE	bit	1
ASIC version	ASIC	bit	1

2.1.1.1.4. ASIC System Configuration (experts/professionals only)

Description	Parameter	Type	Default
LOW POWER MODE: SCG	SCG_EN	bit vector	1
FINE GRAINED CLOCK GATING	CLOCK_GATING	bit	1
ASIC CLOCKING	ASIC_CLOCKING	bit	1
LFXT CLOCK DOMAIN	LFXT_DOMAIN	bit	1
MCLK: Clock Mux	MCLK_MUX	bit	1
SMCLK: Clock Mux	SMCLK_MUX	bit	1
WATCHDOG: Clock Mux	WATCHDOG_MUX	bit	1
WATCHDOG: Clock No-Mux	WATCHDOG_NOMUX_ACLK	bit	0
MCLK: Clock divider	MCLK_DIVIDER	bit	1
SMCLK: Clock divider (/1/2/4/8)	SMCLK_DIVIDER	bit	1
ACLK: Clock divider (/1/2/4/8)	ACLK_DIVIDER	bit	1
LOW POWER MODE: CPUOFF	CPUOFF_EN	bit	1
LOW POWER MODE: OSCOFF	OSCOFF_EN	bit	1

2.1.1.1.5. System Constants (do not edit)

Description	Parameter	Type	Default
Program Memory Size	PMEM_AWIDTH	integer	13
Data Memory Size	DMEM_AWIDTH	integer	11
Peripheral Memory Size	PER_AWIDTH	integer	8

Description	Parameter	Type	Default
Data Memory Base Adresses	DMEM_BASE	integer	N
Program Memory	PMEM_MSB	integer	N
Data Memory	DMEM_MSB	integer	N
Peripheral Memory	PER_MSB	integer	N
Number of available IRQs	IRQ_NR	integer	16
Instructions type	INST_SOC	integer	0
Instructions type	INST_JMPC	integer	1
Instructions type	INST_TOC	integer	2
Single-operand arithmetic	RRC	integer	0
Single-operand arithmetic	SWPB	integer	1
Single-operand arithmetic	RRA	integer	2
Single-operand arithmetic	SXTC	integer	3
Single-operand arithmetic	PUSH	integer	4
Single-operand arithmetic	CALL	integer	5
Single-operand arithmetic	RETI	integer	6
Single-operand arithmetic	IRQX	integer	7
Conditional jump	JNE	integer	0
Conditional jump	JEQ	integer	1
Conditional jump	JNC	integer	2
Conditional jump	JC	integer	3
Conditional jump	JN	integer	4
Conditional jump	JGE	integer	5
Conditional jump	JL	integer	6
Conditional jump	JMP	integer	7
Two-operand arithmetic	MOV	integer	0
Two-operand arithmetic	ADD	integer	1
Two-operand arithmetic	ADDC	integer	2
Two-operand arithmetic	SUBC	integer	3
Two-operand arithmetic	SUBB	integer	4
Two-operand arithmetic	CMP	integer	5
Two-operand arithmetic	DADD	integer	6
Two-operand arithmetic	BITC	integer	7
Two-operand arithmetic	BIC	integer	8
Two-operand arithmetic	BIS	integer	9
Two-operand arithmetic	XORX	integer	10
Two-operand arithmetic	ANDX	integer	11
Addressing modes	DIR	integer	0
Addressing modes	IDX	integer	1
Addressing modes	INDIR	integer	2
Addressing modes	INDIR_I	integer	3
Addressing modes	SYMB	integer	4
Addressing modes	IMM	integer	5
Addressing modes	ABSC	integer	6
Addressing modes	CONST	integer	7
Instruction state machine	I_IRQ_FETCH	bit vector	000
Instruction state machine	I_IRQ_DONE	bit vector	001
Instruction state machine	I_DEC	bit vector	010
Instruction state machine	I_EXT1	bit vector	011
Instruction state machine	I_EXT2	bit vector	100
Instruction state machine	I_IDLE	bit vector	101
Execution state machine	E_SRC_AD	bit vector	X5
Execution state machine	E_SRC_RD	bit vector	X6

Description	Parameter	Type	Default
Execution state machine	E_SRC_WR	bit vector	X7
Execution state machine	E_DST_AD	bit vector	X8
Execution state machine	E_DST_RD	bit vector	X9
Execution state machine	E_DST_WR	bit vector	XA
Execution state machine	E_EXEC	bit vector	XB
Execution state machine	E_JUMP	bit vector	XC
Execution state machine	E_IDLE	bit vector	XD
Execution state machine	E_IRQ_0	bit vector	X2
Execution state machine	E_IRQ_1	bit vector	X1
Execution state machine	E_IRQ_2	bit vector	X0
Execution state machine	E_IRQ_3	bit vector	X3
Execution state machine	E_IRQ_4	bit vector	X4
ALU control signals	ALU_SRC_INV	integer	0
ALU control signals	ALU_INC	integer	1
ALU control signals	ALU_INC_C	integer	2
ALU control signals	ALU_ADD	integer	3
ALU control signals	ALU_AND	integer	4
ALU control signals	ALU_OR	integer	5
ALU control signals	ALU_XOR	integer	6
ALU control signals	ALU_DADD	integer	7
ALU control signals	ALU_STAT_7	integer	8
ALU control signals	ALU_STAT_F	integer	9
ALU control signals	ALU_SHIFT	integer	10
ALU control signals	EXEC_NO_WR	integer	11
Debug interface	DBG_UART_WR	integer	18
Debug interface	DBG_UART_BW	integer	17
Debug interface CPU_CTL register	HALT	integer	0
Debug interface CPU_CTL register	RUN	integer	1
Debug interface CPU_CTL register	ISTEP	integer	2
Debug interface CPU_CTL register	SW_BRK_EN	integer	3
Debug interface CPU_CTL register	FRZ_BRK_EN	integer	4
Debug interface CPU_CTL register	RST_BRK_EN	integer	5
Debug interface CPU_CTL register	CPU_RST	integer	6
Debug interface BRKx_CTL register	BRK_MODE_RD	integer	0
Debug interface BRKx_CTL register	BRK_MODE_WR	integer	1
Debug interface BRKx_CTL register	BRK_EN	integer	2
Debug interface BRKx_CTL register	BRK_I_EN	integer	3
Debug interface BRKx_CTL register	BRK_RANGE	integer	4
Basic clock module: BCSCTL2 Control Register	SELMX	integer	7
Basic clock module: BCSCTL2 Control Register	SELS	integer	3
MCLK Clock gate	MCLK_CGATE	bit	1
SMCLK Clock gate	SMCLK_CGATE	bit	1
Debug interface: CPU version	CPU_VERSION	bit vector	010
Debug interface: Software breakpoint opcode	DBG_SWBRK_OP	bit vector	X4343
UART interface auto data synchronization	DBG_UART_AUTO_SYNC	bit	1
Counter width for the debug interface UART	DBG_UART_XFER_CNT_W	integer	16
Debug UART interface data rate	DBG_UART_BAUD	integer	2000000
Debug UART interface data rate	DBG_DCO_FREQ	integer	20000000
Debug UART interface data rate	DBG_UART_CNT	integer	N
Debug UART interface data rate	DBG_UART_CNTB	bit vector	N
Debug interface input synchronizer	SYNC_DBG_UART_RXD	bit	1
MULTIPLIER CONFIGURATION	MPY_16X16	bit	1

2.1.1.2. Signals

2.1.1.2.1. Instruction Inputs/Outputs Bus

2.1.1.2.2. Data Inputs/Outputs Bus

2.1.3. Registers

2.1.4. Interruptions

2.2. PU-MSP430

2.2.1. Processing Unit

The MSP430 implementation has a 16 bit Microarchitecture, 3 stages data pipeline and an Instruction Set Architecture based on Reduced Instruction Set Computer. Compatible with Wishbone Bus. Only For Researching.

Processing Unit	Module description
`msp430_pu`	Processing Unit
`...msp430_core`	Core
`...msp430_gpio`	General Purpose Input Output
`...msp430_ta`	Timer A
`...msp430_uart`	Universal Asynchronous Receiver-Transmitter

A PU cache is a hardware cache used by the PU to reduce the average cost (time or energy) to access instruction/data from the main memory. A cache is a smaller, faster memory, closer to a core, which stores copies of the data from frequently used main memory locations. Most CPUs have different independent caches, including instruction and data caches.

2.2.2. Instruction Cache

2.2.2.1. Functionality

Instruction Memory	Module description
`riscv_imem_ctrl`	Instruction Memory Access Block
`...riscv_membuf`	Memory Access Buffer
`.....riscv_ram_queue`	Fall-through Queue
`...riscv_memmisaligned`	Misalignment Check
`...riscv_mmu`	Memory Management Unit
`...riscv_pmachk`	Physical Memory Attributes Checker
`...riscv_pmpchk`	Physical Memory Protection Checker
`...riscv_icache_core`	Instruction Cache (Write Back)
`.....riscv_ram_1rw`	RAM 1RW
`.......riscv_ram_1rw_generic`	RAM 1RW Generic
`...riscv_dext`	Data External Access Logic
`...riscv_ram_queue`	Fall-through Queue
`...riscv_mux`	Bus-Interface-Unit Mux
`riscv_biu`	Bus Interface Unit

2.2.2.2. Interface

2.2.2.2.1. Instruction INPUTS/OUTPUTS AMBA4 AXI-Lite Bus

2.2.2.2.1.1. Signals of the Read and Write Address channels

Write Port	Read Port	Size	Direction	Description
AWID	ARID	AXI_ID_WIDTH	Output	Address ID, to identify multiple streams
AWADDR	ARADDR	AXI_ADDR_WIDTH	Output	Address of the first beat of the burst
AWLEN	ARLEN	8	Output	Number of beats inside the burst
AWSIZE	ARSIZE	3	Output	Size of each beat
AWBURST	ARBURST	2	Output	Type of the burst
AWLOCK	ARLOCK	1	Output	Lock type, to provide atomic operations
AWCACHE	ARCACHE	4	Output	Memory type, progress through the system
AWPROT	ARPROT	3	Output	Protection type
AWQOS	ARQOS	4	Output	Quality of Service of the transaction
AWREGION	ARREGION	4	Output	Region identifier, physical to logical
AWUSER	ARUSER	AXI_USER_WIDTH	Output	User-defined data
AWVALID	ARVALID	1	Output	xVALID handshake signal
AWREADY	ARREADY	1	Input	xREADY handshake signal

2.2.2.2.1.2. Signals of the Read and Write Data channels

Write Port	Read Port	Size	Direction	Description
WID	RID	AXI_ID_WIDTH	Output	Data ID, to identify multiple streams
WDATA	RDATA	AXI_DATA_WIDTH	Output	Read/Write data
--	RRESP	2	Output	Read response, current RDATA status
WSTRB	--	AXI_STRB_WIDTH	Output	Byte strobe, WDATA signal
WLAST	RLAST	1	Output	Last beat identifier
WUSER	RUSER	AXI_USER_WIDTH	Output	User-defined data
WVALID	RVALID	1	Output	xVALID handshake signal
WREADY	RREADY	1	Input	xREADY handshake signal

2.2.2.2.1.3. Signals of the Write Response channel

Write Port	Size	Direction	Description
BID	AXI_ID_WIDTH	Input	Write response ID, to identify multiple streams
BRESP	2	Input	Write response, to specify the burst status
BUSER	AXI_USER_WIDTH	Input	User-defined data
BVALID	1	Input	xVALID handshake signal
BREADY	1	Output	xREADY handshake signal

2.2.2.2.2. Instruction INPUTS/OUTPUTS AMBA3 AHB-Lite Bus

Port	Size	Direction	Description
HRESETn	1	Input	Asynchronous Active Low Reset
HCLK	1	Input	System Clock Input
IHSEL	1	Output	Instruction Bus Select

Port	Size	Direction	Description
IHADDR	PLEN	Output	Instruction Address Bus
IHRDATA	XLEN	Input	Instruction Read Data Bus
IHWDATA	XLEN	Output	Instruction Write Data Bus
IHWRITE	1	Output	Instruction Write Select
IHSIZE	3	Output	Instruction Transfer Size
IHBURST	3	Output	Instruction Transfer Burst Size
IHPROT	4	Output	Instruction Transfer Protection Level
IHTRANS	2	Output	Instruction Transfer Type
IHMASTLOCK	1	Output	Instruction Transfer Master Lock
IHREADY	1	Input	Instruction Slave Ready Indicator
IHRESP	1	Input	Instruction Transfer Response

2.2.2.2.3. Instruction INPUTS/OUTPUTS Wishbone Bus

Port	Size	Direction	Description
rst	1	Input	Synchronous Active High Reset
clk	1	Input	System Clock Input
iadr	AW	Input	Instruction Address Bus
idati	DW	Input	Instruction Input Bus
idato	DW	Output	Instruction Output Bus
isel	DW/8	Input	Byte Select Signals
iwe	1	Input	Write Enable Input
istb	1	Input	Strobe Signal/Core Select Input
icyc	1	Input	Valid Bus Cycle Input
iack	1	Output	Bus Cycle Acknowledge Output
ierr	1	Output	Bus Cycle Error Output
iint	1	Output	Interrupt Signal Output

2.2.3. Data Cache

2.2.3.1. Functionality

Data Memory	Module description
riscv_dmem_ctrl	Data Memory Access Block
...riscv_membuf	Memory Access Buffer
.....riscv_ram_queue	Fall-through Queue
...riscv_memmisaligned	Misalignment Check
...riscv_mmu	Memory Management Unit
...riscv_pmachk	Physical Memory Attributes Checker
...riscv_pmpchk	Physical Memory Protection Checker
...riscv_dcache_core	Data Cache (Write Back)
.....riscv_ram_1rw	RAM 1RW
.......riscv_ram_1rw_generic	RAM 1RW Generic
...riscv_dext	Data External Access Logic
...riscv_mux	Bus-Interface-Unit Mux
riscv_biu	Bus Interface Unit

2.2.3.2. Interface

2.2.3.2.1. Data INPUTS/OUTPUTS AMBA4 AXI-Lite Bus

2.2.3.2.1.1. Signals of the Read and Write Address channels

Write Port	Read Port	Size	Direction	Description
AWID	ARID	AXI_ID_WIDTH	Output	Address ID, to identify multiple streams
AWADDR	ARADDR	AXI_ADDR_WIDTH	Output	Address of the first beat of the burst
AWLEN	ARLEN	8	Output	Number of beats inside the burst
AWSIZE	ARSIZE	3	Output	Size of each beat
AWBURST	ARBURST	2	Output	Type of the burst
AWLOCK	ARLOCK	1	Output	Lock type, to provide atomic operations
AWCACHE	ARCACHE	4	Output	Memory type, progress through the system
AWPROT	ARPROT	3	Output	Protection type
AWQOS	ARQOS	4	Output	Quality of Service of the transaction
AWREGION	ARREGION	4	Output	Region identifier, physical to logical
AWUSER	ARUSER	AXI_USER_WIDTH	Output	User-defined data
AWVALID	ARVALID	1	Output	xVALID handshake signal
AWREADY	ARREADY	1	Input	xREADY handshake signal

2.2.3.2.1.2. Signals of the Read and Write Data channels

Write Port	Read Port	Size	Direction	Description
WID	RID	AXI_ID_WIDTH	Output	Data ID, to identify multiple streams
WDATA	RDATA	AXI_DATA_WIDTH	Output	Read/Write data
--	RRESP	2	Output	Read response, current RDATA status
WSTRB	--	AXI_STRB_WIDTH	Output	Byte strobe, WDATA signal
WLAST	RLAST	1	Output	Last beat identifier
WUSER	RUSER	AXI_USER_WIDTH	Output	User-defined data
WVALID	RVALID	1	Output	xVALID handshake signal
WREADY	RREADY	1	Input	xREADY handshake signal

2.2.3.2.1.3. Signals of the Write Response channel

Write Port	Size	Direction	Description
BID	AXI_ID_WIDTH	Input	Write response ID, to identify multiple streams
BRESP	2	Input	Write response, to specify the burst status
BUSER	AXI_USER_WIDTH	Input	User-defined data
BVALID	1	Input	xVALID handshake signal
BREADY	1	Output	xREADY handshake signal

2.2.3.2.2. Data INPUTS/OUTPUTS AMBA3 AHB-Lite Bus

Port	Size	Direction	Description
HRESETn	1	Input	Asynchronous Active Low Reset
HCLK	1	Input	System Clock Input
DHSEL	1	Output	Data Bus Select

Port	Size	Direction	Description
DHADDR	PLEN	Output	Data Address Bus
DHRDATA	XLEN	Input	Data Read Data Bus
DHWDATA	XLEN	Output	Data Write Data Bus
DHWRITE	1	Output	Data Write Select
DHSIZE	3	Output	Data Transfer Size
DHBURST	3	Output	Data Transfer Burst Size
DHPROT	4	Output	Data Transfer Protection Level
DHTRANS	2	Output	Data Transfer Type
DHMASTLOCK	1	Output	Data Transfer Master Lock
DHREADY	1	Input	Data Slave Ready Indicator
DHRESP	1	Input	Data Transfer Response

2.2.3.2.3. Data INPUTS/OUTPUTS Wishbone Bus

Port	Size	Direction	Description
rst	1	Input	Synchronous Active High Reset
clk	1	Input	System Clock Input
dadr	AW	Input	Data Address Bus
ddati	DW	Input	Data Input Bus
ddato	DW	Output	Data Output Bus
dsel	DW/8	Input	Byte Select Signals
dwe	1	Input	Write Enable Input
dstb	1	Input	Strobe Signal/Core Select Input
dcyc	1	Input	Valid Bus Cycle Input
dack	1	Output	Bus Cycle Acknowledge Output
derr	1	Output	Bus Cycle Error Output
dint	1	Output	Interrupt Signal Output

2.3. SoC-MSP430

2.3.1. MPSoC-DBG

2.3.2. MPSoC-DMA

2.3.3. MPSoC-GPIO

2.3.4. MPSoC-MPI

2.3.5. MPSoC-MPRAM

2.3.6. MPSoC-MSI

2.3.7. MPSoC-NoC

2.3.8. MPSoC-SPRAM

2.3.9. MPSoC-UART

2.4. MPSoC-MSP430

3. WORKFLOW

3.1. HARDWARE

1. System Level (SystemC/SystemVerilog)

The System Level abstraction of a system only looks at its biggest building blocks like processing units or peripheral devices. At this level the circuit is usually described using traditional programming languages like SystemC or SystemVerilog. Sometimes special software libraries are used that are aimed at simulation circuits on the system level. The IEEE 1685-2009 standard defines the IP-XACT file format that can be used to represent designs on the system level and building blocks that can be used in such system level designs.

2. Behavioral & Register Transfer Level (VHDL/Verilog)

At the Behavioural Level abstraction a language aimed at hardware description such as Verilog or VHDL is used to describe the circuit, but so-called behavioural modeling is used in at least part of the circuit description. In behavioural modeling there must be a language feature that allows for imperative programming to be used to describe data paths and registers. This is the always -block in Verilog and the process -block in VHDL.

A design in Register Transfer Level representation is usually stored using HDLs like Verilog and VHDL. But only a very limited subset of features is used, namely minimalistic always blocks (Verilog) or process blocks (VHDL) that model the register type used and unconditional assignments for the datapath logic. The use of HDLs on this level simplifies simulation as no additional tools are required to simulate a design in Register Transfer Level representation.

3. Logical Gate

At the Logical Gate Level the design is represented by a netlist that uses only cells from a small number of single-bit cells, such as basic logic gates (AND, OR, NOT, XOR, etc.) and registers (usually D-Type Flip-flops). A number of netlist formats exists that can be used on this level such as the Electronic Design Interchange Format (EDIF), but for ease of simulation often a HDL netlist is used. The latter is a HDL file (Verilog or VHDL) that only uses the most basic language constructs for instantiation and connecting of cells.

4. Physical Gate

On the Physical Gate Level only gates are used that are physically available on the target architecture. In some cases this may only be NAND, NOR and NOT gates as well as D-Type registers. In the case of an FPGA-based design the Physical Gate Level representation is a netlist of LUTs with optional output registers, as these are the basic building blocks of FPGA logic cells.

5. Switch Level

A Switch Level representation of a circuit is a netlist utilizing single transistors as cells. Switch Level modeling is possible in Verilog and VHDL, but is seldom used in modern designs, as in modern digital ASIC or FPGA flows the physical gates are considered the atomic build blocks of the logic circuit.

3.1.1. Front-End Open Source Tools

3.1.1.1. Modeling System Level of Hardware

A System Description Language Editor is a computer tool allows to generate software code. A System Description Language is a formal language, which comprises a Programming Language (input), producing a Hardware Description (output). Programming languages are used in computer programming to implement algorithms. The description of a programming language is split into the two components of syntax (form) and semantics (meaning).

SystemVerilog System Description Language Editor

type:

```
git clone --recursive https://github.com/emacs-mirror/emacs

cd emacs
./configure
make
sudo make install
```

3.1.1.2. Simulating System Level of Hardware

A System Description Language Simulator (translator) is a computer program that translates computer code written in a Programming Language (the source language) into a Hardware Description Language (the target language). The compiler is primarily used for programs that translate source code from a high-level programming language to a low-level language to create an executable program.

SystemVerilog System Description Language Simulator

type:

```
git clone --recursive http://git.veripool.org/git/verilator

cd verilator
autoconf
./configure
make
sudo make install

cd sim/verilog/tests/wb/verilator
source SIMULATE-IT

cd sim/verilog/tests/ahb3/verilator
source SIMULATE-IT

cd sim/verilog/tests/axi4/verilator
source SIMULATE-IT
```

3.1.1.3. Verifying System Level of Hardware

A UVM standard improves interoperability and reduces the cost of repurchasing and rewriting IP for each new project or Electronic Design Automation tool. It also makes it easier to reuse verification components. The UVM Class Library provides generic utilities, such as component hierarchy, Transaction Library Model or configuration database, which enable the user to create virtually any structure wanted for the testbench.

SystemVerilog System Description Language Verifier

type:

```
git clone --recursive https://github.com/QueenField/UVM
```

```
cd sim/verilog/pu/riscv/wb/msim
source SIMULATE-IT
```

```
cd sim/verilog/pu/riscv/ahb3/msim
source SIMULATE-IT
```

```
cd sim/verilog/pu/riscv/axi4/msim
source SIMULATE-IT
```

3.1.1.4. Describing Register Transfer Level of Hardware

A Hardware Description Language Editor is any editor that allows to generate hardware code. Hardware Description Language is a specialized computer language used to describe the structure and behavior of digital logic circuits. It allows for the synthesis of a HDL into a netlist, which can then be synthesized, placed and routed to produce the set of masks used to create an integrated circuit.

VHDL/Verilog Hardware Description Language Editor

type:

```
git clone --recursive https://github.com/emacs-mirror/emacs
```

```
cd emacs
./configure
make
sudo make install
```

3.1.1.5. Simulating Register Transfer Level of Hardware

A Hardware Description Language Simulator uses mathematical models to replicate the behavior of an actual hardware device. Simulation software allows for modeling of circuit operation and is an invaluable analysis tool. Simulating a circuit's behavior before actually building it can greatly improve design efficiency by making faulty designs known as such, and providing insight into the behavior of electronics circuit designs.

Verilog Hardware Description Language Simulator

type:

```
git clone --recursive https://github.com/steveicarus/iverilog
```

```
cd iverilog
sh autoconf.sh
./configure
make
sudo make install
```

```
cd sim/verilog/tests/wb/iverilog
source SIMULATE-IT
```

```
cd sim/verilog/tests/ahb3/iverilog
source SIMULATE-IT
```

```
cd sim/verilog/tests/axi4/iverilog
source SIMULATE-IT
```

VHDL Hardware Description Language Simulator

type:

```
git clone --recursive https://github.com/ghdl/ghdl

cd ghdl
./configure --prefix=/usr/local
make
sudo make install
```

```
cd sim/vhdl/tests/wb/ghdl
source SIMULATE-IT
```

```
cd sim/vhdl/tests/ahb3/ghdl
source SIMULATE-IT
```

```
cd sim/vhdl/tests/axi4/ghdl
source SIMULATE-IT
```

3.1.1.6. Synthesizing Register Transfer Level of Hardware

A Hardware Description Language Synthesizer turns a RTL implementation into a Logical Gate Level implementation. Logical design is a step in the standard design cycle in which the functional design of an electronic circuit is converted into the representation which captures logic operations, arithmetic operations, control flow, etc. In EDA parts of the logical design is automated using synthesis tools based on the behavioral description of the circuit.

Verilog Hardware Description Language Synthesizer

type:

```
git clone --recursive https://github.com/YosysHQ/yosys

cd yosys
make
sudo make install
```

VHDL Hardware Description Language Synthesizer

type:

```
git clone --recursive https://github.com/ghdl/ghdl-yosys-plugin
cd ghdl-yosys-plugin
make GHDL=/usr/local
sudo yosys-config --exec mkdir -p --datdir/plugins
sudo yosys-config --exec cp "ghdl.so" --datdir/plugins/ghdl.so
```

3.1.1.7. Optimizing Register Transfer Level of Hardware

A Hardware Description Language Optimizer finds an equivalent representation of the specified logic circuit under specified constraints (minimum area, pre-specified delay). This tool combines scalable logic optimization based on And-Inverter Graphs (AIGs), optimal-delay DAG-based technology mapping for look-up tables and standard cells, and innovative algorithms for sequential synthesis and verification.

Verilog Hardware Description Language Optimizer

type:

```
git clone --recursive https://github.com/YosysHQ/yosys

cd yosys
make
sudo make install
```

3.1.1.8. Verifying Register Transfer Level of Hardware

A Hardware Description Language Verifier proves or disproves the correctness of intended algorithms underlying a hardware system with respect to a certain formal specification or property, using formal methods of mathematics. Formal verification uses modern techniques (SAT/SMT solvers, BDDs, etc.) to prove correctness by essentially doing an exhaustive search through the entire possible input space (formal proof).

Verilog Hardware Description Language Verifier

type:

```
git clone --recursive https://github.com/YosysHQ/SymbiYosys
```

3.1.2. Back-End Open Source Tools

I. Back-End Workflow Qflow for ASICs

type:

```
sudo apt install bison cmake flex freeglut3-dev libcairo2-dev libgsl-dev \
libncurses-dev libx11-dev m4 python-tk python3-tk swig tcl tcl-dev tk-dev tcsh
```

type:

```
git clone --recursive https://github.com/RTimothyEdwards/qflow

cd qflow
./configure
make
sudo make install
```

3.1.2.1. Planning Switch Level of Hardware

A Floor-Planner of an Integrated Circuit (IC) is a schematic representation of tentative placement of its major functional blocks. In modern electronic design process floor-plans are created during the floor-planning design stage, an early stage in the hierarchical approach to Integrated Circuit design. Depending on the design methodology being followed, the actual definition of a floor-plan may differ.

Floor-Planner

type:

```
git clone --recursive https://github.com/RTimothyEdwards/magic

cd magic
./configure
make
sudo make install
```

3.1.2.2. Placing Switch Level of Hardware

A Standard Cell Placer takes a given synthesized circuit netlist together with a technology library and produces a valid placement layout. The layout is optimized according to the aforementioned objectives and ready for cell resizing and buffering, a step essential for timing and signal integrity satisfaction. Physical design flow are iterated a number of times until design closure is achieved.

Standard Cell Placer

type:

```
git clone --recursive https://github.com/rubund/graywolf

cd graywolf
mkdir build
cd build
cmake ..
make
sudo make install
```

3.1.2.3. Timing Switch Level of Hardware

A Standard Cell Timing-Analizer is a simulation method of computing the expected timing of a digital circuit without requiring a simulation of the full circuit. High-performance integrated circuits have traditionally been characterized by the clock frequency at which they operate. Measuring the ability of a circuit to operate at the specified speed requires an ability to measure, during the design process, its delay at numerous steps.

Standard Cell Timing-Analizer

type:

```
git clone --recursive https://github.com/The-OpenROAD-Project/OpenSTA

cd OpenSTA
mkdir build
cd build
cmake ..
make
sudo make install
```

3.1.2.4. Routing Switch Level of Hardware

A Standard Cell Router takes pre-existing polygons consisting of pins on cells, and pre-existing wiring called pre-routes. Each of these polygons are associated with a net. The primary task of the router is to create geometries such that all terminals assigned to the same net are connected, no terminals assigned to different nets are connected, and all design rules are obeyed.

Standard Cell Router

type:

```
git clone --recursive https://github.com/RTimothyEdwards/qrouter

cd qrouter
./configure
make
sudo make install
```

3.1.2.5. Simulating Switch Level of Hardware

A Standard Cell Simulator treats transistors as ideal switches. Extracted capacitance and lumped resistance values are used to make the switch a little bit more realistic than the ideal, using the RC time constants to predict the relative timing of events. This simulator represents a circuit in terms of its exact transistor structure but describes the electrical behavior in a highly idealized way.

Standard Cell Simulator

type:

```
git clone --recursive https://github.com/RTimothyEdwards/irsim

cd irsim
./configure
make
sudo make install
```

3.1.2.6. Verifying Switch Level of Hardware LVS

A Standard Cell Verifier compares netlists, a process known as LVS (Layout vs. Schematic). This step ensures that the geometry that has been laid out matches the expected circuit. The greatest need for LVS is in large analog or mixed-signal circuits that cannot be simulated in reasonable time. LVS can be done faster than simulation, and provides feedback that makes it easier to find errors.

Standard Cell Verifier

type:

```
git clone --recursive https://github.com/RTimothyEdwards/netgen

cd netgen
./configure
make
sudo make install
```

3.1.2.7. Checking Switch Level of Hardware DRC

A Standard Cell Checker is a geometric constraint imposed on Printed Circuit Board (PCB) and Integrated Circuit (IC) designers to ensure their designs function properly, reliably, and can be produced with acceptable yield. Design Rules for production are developed by hardware engineers based on the capability of their processes to realize design intent. Design Rule Checking (DRC) is used to ensure that designers do not violate design rules.

Standard Cell Checker

type:

```
git clone --recursive https://github.com/RTimothyEdwards/magic

cd magic
./configure
make
sudo make install
```

3.1.2.8. Printing Switch Level of Hardware GDS

A Standard Cell Editor allows to print a set of standard cells. The standard cell methodology is an abstraction, whereby a low-level VLSI layout is encapsulated into a logical representation. A standard cell is a group *of transistor and interconnect structures that provides a boolean logic function (AND, OR, XOR, XNOR, inverters) or a storage function (flipflop or latch).*

Standard Cell Editor

type:

```
git clone --recursive https://github.com/RTimothyEdwards/magic

cd magic
./configure
make
sudo make install
```

II. Back-End Workflow Symbiflow for FPGAs

3.2. SOFTWARE

3.2.1. Compilers

type:

```
sudo apt install autoconf automake autotools-dev curl python3 libmpc-dev \
libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf \
libtool patchutils bc zlib1g-dev libexpat-dev
```

3.2.1.1. MSP430 GNU C/C++

type:

```
git clone --recursive https://github.com/riscv/riscv-gnu-toolchain

cd riscv-gnu-toolchain

./configure --prefix=/opt/riscv-elf-gcc
sudo make clean
sudo make

./configure --prefix=/opt/riscv-elf-gcc
sudo make clean
sudo make linux

./configure --prefix=/opt/riscv-elf-gcc --enable-multilib
sudo make clean
sudo make linux
```

3.2.1.2. MSP430 GNU Go

type:

```
git clone --recursive https://go.googlesource.com/go riscv-go
cd riscv-go/src
./all.bash
cd ../..
sudo mv riscv-go /opt
```

3.2.2. Simulators

type:

```
sudo apt install device-tree-compiler libglib2.0-dev libpixman-1-dev pkg-config
```

3.2.2.1. Spike (For Hardware Engineers)

Building Proxy Kernel

type:

```
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

git clone --recursive https://github.com/riscv/riscv-pk

cd riscv-pk
mkdir build
cd build
../configure --prefix=/opt/riscv-elf-gcc --host=riscv64-unknown-elf
make
sudo make install
```

Building Spike

type:

```
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

git clone --recursive https://github.com/riscv/riscv-isa-sim

cd riscv-isa-sim
mkdir build
cd build
../configure --prefix=/opt/riscv-elf-gcc
make
sudo make install
```

3.2.2.2. QEMU (For Software Engineers)

type:

```
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

git clone --recursive https://github.com/qemu/qemu

cd qemu
./configure --prefix=/opt/riscv-elf-gcc \
--target-list=riscv64-softmmu,riscv32-softmmu,riscv64-linux-user,riscv32-linux-user
make
sudo make install
```

4. CONCLUSION

4.1. HARDWARE

```
cd synthesis/yosys
source SYNTHESIZE-IT
```

4.1.1. GSCL 45 nm ASIC

type:

```
cd synthesis/qflow
source FLOW-IT
```

4.1.2. Lattice iCE40 FPGA

type:

```
cd synthesis/symbiflow
source FLOW-IT
```

4.2. SOFTWARE

4.2.1. MSP430 Tests

type:

```
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

rm -rf tests
rm -rf riscv-tests

mkdir tests
mkdir tests/dump
mkdir tests/hex

git clone --recursive https://github.com/riscv/riscv-tests
cd riscv-tests

autoconf
./configure --prefix=/opt/riscv-elf-gcc/bin
make

cd isa

source ../../elf2hex.sh

mv *.dump ../../tests/dump
mv *.hex ../../tests/hex

cd ..

make clean
```

elf2hex.sh:

```
riscv64-unknown-elf-objcopy -O ihex rv32mi-p-breakpoint rv32mi-p-breakpoint.hex
riscv64-unknown-elf-objcopy -O ihex rv32mi-p-csr rv32mi-p-csr.hex
...
riscv64-unknown-elf-objcopy -O ihex rv64um-v-remw rv64um-v-remw.hex
```

type:

```
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

spike rv32mi-p-breakpoint
spike rv32mi-p-csr
...
spike rv64um-v-remw
```

4.2.2. MSP430 Bare Metal

type:

```
rm -rf hello_c.elf
rm -rf hello_c.hex

export PATH=/opt/riscv-elf-gcc/bin:${PATH}

riscv64-unknown-elf-gcc -o hello_c.elf hello_c.c
riscv64-unknown-elf-objcopy -O ihex hello_c.elf hello_c.hex
```

C Language:

```c
#include <stdio.h>

int main() {
  printf("Hello QueenField!\n");
  return 0;
}
```

type:

```
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

spike pk hello_c.elf
```

type:

```
rm -rf hello_cpp.elf
rm -rf hello_cpp.hex

export PATH=/opt/riscv-elf-gcc/bin:${PATH}

riscv64-unknown-elf-g++ -o hello_cpp.elf hello_cpp.cpp
riscv64-unknown-elf-objcopy -O ihex hello_cpp.elf hello_cpp.hex
```

C++ Language:

```cpp
#include <iostream>

int main() {
  std::cout << "Hello QueenField!\n";
```

```
  return 0;
}
```

type:

```
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

spike pk hello_cpp.elf
```

type:

```
rm -rf hello_go.elf
rm -rf hello_go.hex

export PATH=/opt/riscv-elf-gcc/bin:${PATH}
export PATH=/opt/riscv-go/bin:${PATH}

GOOS=linux GOARCH=riscv64 go build -o hello_go.elf hello_go.go
riscv64-unknown-elf-objcopy -O ihex hello_go.elf hello_go.hex
```

Go Language:

```go
package main

import "fmt"
func main() {
  fmt.Println("Hello QueenField!")
}
```

4.2.3. MSP430 Operating System

4.2.3.1. GNU Linux

Building BusyBox

type:

```
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

git clone --recursive https://git.busybox.net/busybox

cd busybox
make CROSS_COMPILE=riscv64-unknown-linux-gnu- defconfig
make CROSS_COMPILE=riscv64-unknown-linux-gnu-
```

Building Linux

type:

```
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

git clone --recursive https://github.com/torvalds/linux

cd linux
make ARCH=riscv CROSS_COMPILE=riscv64-unknown-linux-gnu- defconfig
make ARCH=riscv CROSS_COMPILE=riscv64-unknown-linux-gnu-
```

Running Linux

type:

```
export PATH=/opt/riscv-elf-gcc/bin:${PATH}

qemu-system-riscv64 -nographic -machine virt \
-kernel Image -append "root=/dev/vda ro console=ttyS0" \
-drive file=busybox,format=raw,id=hd0 \
-device virtio-blk-device,drive=hd0
```

4.2.3.2. GNU Hurd

4.2.4. MSP430 Distribution

4.2.4.1. GNU Debian

4.2.4.2. GNU Fedora