Light52 -- free, open source MCS51 compatible CPU core

Revision 2 - September 27, 2012 Core Datasheet

OVERVIEW

Light52 is a free, small open-source CPU core compatible to the Intel MCS51 architecture.

This core has been designed with an emphasis on area reduction. While it is within the performance envelope of other free MCS51 cores, the implementation trades area for speed. This core is smaller than most free and commercial MCS51 cores and its speed is comparable to that of a 5-clocker.

The core implements the full MCS51 instruction set with the possible exception of the BCD opcodes (DA and XCHG) which are optional.

This core is in a very early stage of development; it is not fully tested and not documented at all except for the code comments and this file.

All the information presented in this datasheet should be considered preliminary.

FEATURES

- 100% binary compatible to MCS51 (except possibly for optional BCD instructions).
- Speed comparable to a 5-clocker.
- Configurable through VHDL generics.
- Smaller than most other cores.
- Clock rate not much worse than other commercial cores.
- Includes 16-bit timer, UART and I/O ports.
- Additional peripherals and SFRs can be added easily.
- Fully synthesizable, static synchronous design with positive edge clocking and no internal tri-states.

Light52 lacks some features usually present in other MCS51 cores and has some important limitations:

SHORTCOMINGS

- No access to off-chip memory.
- Strictly Harvard: XDATA and XCODE spaces can't be merged.
- From 2 to 8 clocks per instruction.
- Far slower than most commercial cores: performance/area ratio is worse even though area is much smaller.

Pinout

Table 1: ***Core Signal Pinout***

Signal	Direction	Description
clk	input	Clock, active on rising edge.
reset	input	Active high synchronous reset.
rxd	input	RxD input for on-board UART.
txd	output	TxD output from on-board UART.
external_irq[7..0]	input	High-level-sensitive interrupt inputs
p0_out[7..0]	output	Port P0 8-bit output.
p1_out[7..0]	output	Port P1 8-bit output.
p2_in[7..0]	input	Port P2 8-bit input.
p3_in[7..0]	input	Port P3 8-bit input.

Functional Description

Since the MCS51 architecture is already well documented elsewhere, this datasheet will only deal with those aspects of the core which depart from the original.

In this version of the core, there is no support for shared XCODE/XDATA memory spaces (the core performs simultaneous accesses to XCODE and XDATA and there is no wait state or access arbitrage logic yet). The MCU memory model is therefore strictly Harvard.

The peripherals included in the MCU core are generally not compatible to the MS51 peripherals and are somewhat less flexible -- the core trades programmability in run-time for configurability in synthesis time. See the peripherals section below for a detailed description of available peripherals.

Interrupt operation is identical to the original, except for SFR register IP: interrupt priprities are fixed to their default values and the IP SFR is unimplemented. Interrups can be tailored to any specific application by customizing the MCU VHDL.

Programming Model

Table 2 lists the SFRs implemented in the current version of the core.

Table 2: *Light52 Special Function Registers*

Symbol	Description	Direct Address	Bit Address and Symbol								Reset Value
ACC	Accumulator	E0H	E7	E6	E5	E4	E3	E2	E1	E0	00H
B	B register	F0H	F7	F6	F5	F4	F3	F2	F1	F0	00H
DPH	DPTR high	83H									00H
DPL	DPTR low	82H									00H
IE	IQR Enable	A8H	AF	AE	AD	AC	AB	AA	A9	A8	00H
			EA	-	-	ES	-	-	ET0	-	
PSW	Program	D0H	D7	D6	D5	D4	D3	D2	D1	D0	00H
	Status Word		CY	AC	F0	RS1	RS0	OV	-	P	
SP	Stack Pointer	81H									07H
P0	Port 0 outp.	80H	87	86	85	84	83	82	81	80	00H
P1	Port 1 outp.	90H	97	96	95	94	93	92	91	90	00H
P2	Port 2 inp.	A0H	A7	A6	A5	A4	A3	A2	A1	A0	
P3	Port 3 inp.	B0H	B7	B6	B5	B4	B3	B2	B1	B0	
TCON	Timer 0	88H	8F	8E	8D	8C	8B	8A	89	88	00H
	Control		-	-	CEN	ARL	-	-	-	T0IRQ	
TH0		8CH									
TL0		8AH									
SCON	UART	98H	9F	9E	9D	9C	9B	9A	99	98	00H
	Control		-	-	RxRdy	TxRdy	-	-	RxIrq	TxIrq	
SBUF	Data Buffer	99H									
SBPL	Baud Rate L	9AH									(*1)
SBPH	Baud Rate H	9BH									(*1)
EXTINT	External	C0H	C7	C6	C5	C4	C3	C2	C1	C0	00H
	IRQ Flags		EIRQ7	EIRQ6	EIRQ5	EIRQ4	EIRQ3	EIRQ2	EIRQ1	EIRQ0	

Notes

1 Only if generic UART_HARDWIRED is false, and then write only.

The core implements a subset of the MCS51 peripherals and SFRs, and the peripherals implemented are incompatible with the original. For this reason, existing MCS51 programs will generally NOT work unmodified on this core -- code needs to be ported like it needs to be in any other MCS51 derivative.

Note too that SFR registers PCON and IP are not implemented.

Object Code Initialization

The object code for the MCU application is contained within the MCU module. The XCODE ROM is initialized at synthesis time with the contents of generic **OBJ_CODE**, which is expected to be defined in package **obj_code_pkg**.

The constant type (**t_obj_code**) is defined in package **light52_pkg**, which is part of the core sources.

This project has adopted the convention that the package **obj_code_pkg** must be defined in a vhdl file placed within the MCS51 program directories -- for this purpose, the vhdl package can be considered as just another object code format.

This way, the object code for different projects using this core (or for the different code samples within this project) can be neatly separated from the core sources.

The project includes a Python script (directory **/tools/build_rom**) which can be used to produce a suitable **obj_code_pkg** package file from an Intel-HEX object file. The code samples in directory **/test** contain usage examples for this script (BAT files **build.bat**).

While the method chosen for object code initialization is clean and vendor-independent, it has a major drawback: The object code must be available at synthesis time, and every time the code changes the synthesis has to be re-run. This may be a big handicap in certain applications.

Subsequent versions of the core may provide the option to use memory initialization files so that the XCODE memory can be initialized post-synthesis.

Note that the build BAT scripts and the assembler source format is tailored for ASEM-51, a free MCS51 assembler which is available in Windows and Linux versions.

Configuration Generics

Some of the core features can be configured through VHDL generics:

Table 3: ***Core Configuration Generics***

Generic	Default	Description
CODE_ROM_SIZE	1024	Size of XCODE ROM in bytes. Can't be zero.
XDATA_RAM_SIZE	512	Size of XDATA RAM in bytes. Can't be zero.
OBJ_CODE	(dummy)	Object code to be placed on ROM. See previous section.
USE_BRAM_FOR_XRAM (*1)	false	Use extra BRAM as XDATA RAM.
IMPLEMENT_BCD_INSTRUCTIONS (*1)	false	Tyue to implement DA and XCHG, false to execute them as NOPs.
SEQUENTIAL_MULTIPLIER (*1)	false	Use sequential multiplier instead of combinational.
UART_HARDWIRED	true	True to hardwire UART baud rate, false to make it configurable at run time.
UART_BAUD_RATE	19200	Default baud rate for UART.
UART_CLOCK_RATE	50MHz	Clock rate assumed by UART initialization constants.

Notes

1 Unimplemented, will cause a synthesis assertion failure if given a non-default value.

At this early stage of development some of these generics do not work, and others are not checked against bounds.

TBD: This datasheet should explain each of the configuration generics.

Peripheral Modules

The MCU core includes a number of peripheral modules. These peripherals have been designed hastily in order to provide a working environment for the CPU -- they do not have their own separate test bench, for example.

The current version of the MCU ships with a simple, hardwired UART, a 16-bit timer and four 8-bit input/output ports.

Light52 UART

The light52 UART is a simplified version of the original MCS51 serial port.

Some of the operational parameters of the UART are hardwired and non-configurable in the current version, not even at synthesis time:

1. Number of stop bits hardwired to 1.
2. Parity hardwired to NONE.
3. Number of bits per character hardwired to 8.

Besides, the 9-bit mode of the original MCS51, whith its applications in inter-MCU communication, is unimplemented yet.

Serial port interrupts work the same as in the original serial port (same vector and same interrupt enable flag IE.ES).

The UART core has some limited capability to recover from errors, described in the source file and very similar to that of the original UART. A follow-up version of this MCS will include flags for detected errors (bad start and stop bits and bit sampling errors), as well as TX and RX overruns.

Since all operational parameters are hardwired except possibly the baud rate, the UART setup is easy: set the baud rate by writing to registers SBPL and SBPH and enable interrupts by setting flag IE.ES -- the UART can be operated in polling mode too if desired.

Register SCON

This register reflects the status of the serial port:

	7	6	5	4	3	2	1	0
SCON	0	0	RxRdy	TxRdy	0	0	RxIrq	TxIrq
	h	h	r	r	h	h	w1c	w1c

Bits marked 'h' are hardwired and can't be modified.

Bits marked 'r' are read only; they are set and reset by the core.

Bits marked 'w1c' (write 1 to clear) are set by the core when an interrupt has been triggered and must be cleared by the software by writing a 1.

TxRdy	Ready to transmit. High when there's no transmission in progress. It is cleared when data is written to SBUF and is raised at the same time a TX interrupt is triggered.
RxRdy	Received data ready. High when there's data in the RX buffer. Raised at the same time the RX interrupt is triggered, cleared when SBUF is read.
RxIrq	RX interrupt pending service. Raised when the RX interrupt is triggered, cleared when 1 is written to it.
TxIrq	TX interrupt pending service. Raised when the TX interrupt is triggered, cleared when 1 is written to it.

When writing to the status/control registers, only flags **TxIrq** and **RxIrq** are affected, and only when writing a '1' as explained above. All other flags are read-only.

Note that register SCON implements control flags only. Some future version of the core will implement a SMOD register with configuration flags for the parameters that now are hardwired.

Register SBUF

This is the read/write buffer of the serial port. It gives the software access to the 1-byte-deep receive and transmit buffers. These buffers work like the original MCS51 serial port.

	7	6	5	4	3	2	1	0
SBUF	UART Tx/Rx buffer register							
reset	x	x	x	x	x	x	x	x
	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits marked 'r/w' can be read and written to by the CPU and can be updated by the core.

SBUF	Writing to this register will trigger a serial port transmission unless SCONTxRdy=0. Reading this register will give the last byte received by the UART, if any.

Writing to this register will trigger a transmission unless there is already a transmission going on (flag SCON.TxRdy=0). In which case the last write access will be ignored. there is no overrun flag to signal this event; the user must prevent it from happening.

When a byte is received, the core raises flag SCON.RxRdy=1. If a new byte is received before the last one has been read (i.e. with flag SCON.RxRdy=1), the receive buffer register will be overwritten with the new data. Again, there is no indication that this has happened; the user must make sure to prevent these overruns.

Reading from this register when flag SCON.RxRdy=1 will clear the flag and return the last received byte.

Reading from this register when flag SCON.RxRdy=0 will return undefined data (usually the last received byte but this may change in later versions).

Registers SBPH and SBPL

If generic UART_HARDWIRED is set to false, then the UART implements these two write-only registers.

These registers should be loaded with the baud period measured in clock cycles -- no prescaling involved:

BIT_PERIOD = UART_CLOCK_RATE / UART_BAUD_RATE

The bit period register is the 13-bit wide combination of SBPH and SBPL, with the 3 higher bits of SBPH being ignored.

Note that these registers are write only: reading from their addresses will return an indetermnate value (actually, the value of the SCON register). This saves logic and is hardly an inconvenience for the programmer, which will seldom have to read these registers.

These registers are loaded at reset with their default value, defined by generics UART_BAUD_RATE and UART_CLOCK_RATE, according to the same formula above.

When the generic UART_HARDWIRED is set to true, these registers are hardwired to their default value and writing to them has no effect.

Timer 0

Basic timer, not directly compatible to any of the original MCS51 timers. This timer is totally independent of the UART.

This is essentially a reloadable up-counter that optionally triggers an interrupt every time the count reaches a certain value.

Timer Registers

The core includes 3 registers:

1. A configurable prescaler register of up to 31 bits.
2. A 16-bit compare register accessible through TCL and TCH.
3. A 16-bit counter register accessible through TL and TH.

Reading TL or TH will give the value of the timer register. If the registers are read while the count is enabled, the software has to deal with a possibly inconsistent (TL,TH) pair and should apply the usual tricks -- majority vote, etc.

The prescaler is reset to 0 when TCON.CEN=0. When TCON.CEN=1 it counts up to PRESCALER_VALUE-1, then rolls over to 0 and the timer register is incremented.

PRESCALER_VALUE is a VHDL generic configurable at synthesis time.

The compare register is write-only, in order to save logic. Reading TCH or TCL will give the value of TCON.

Timer Operation

The counter register is reset to 0 when TCON.CEN=0. When flag TCON.CEN is set to 1, the counter starts counting up at a rate of one count every PRESCALER_VALUE clock cycles.

When counter register = reload register, the following will happen:

- If flag ARL is 0 the core will clear flag CEN and and raise flag Irq, triggering an interrupt. The counter will overflow to 0000h and stop.
- If flag ARL is 1 then flag CEN will remain high and flag Irq will be raised, triggering an interrupt. The counter will overflow to 0000h and continue counting.

Register TSTAT

This register reflects the status of the timer:

	7	6	5	4	3	2	1	0
TSTAT	0	0	CEN	ARL	0	0	0	TIRQ
reset	0	0	0	0	0	0	0	0
	h	h	r/w	r/w	h	h	h	w1c

Bits marked 'h' are hardwired and can't be modified.

Bits marked 'r' are read only; they are set and reset by the core.

Bits marked 'r/w' can be read and written to by the CPU and can be reset by the core.

Bits marked 'w1c' (write 1 to clear) are set by the core when an interrupt has been triggered and must be cleared by the software by writing a 1.

CEN	Count ENable. Must be set to 1 by the CPU to start the counter. When CEN=0 the prescaler and the counter register are reset to 0. Writing a 1 to CEN will start the count up. The counter will increment until it matches the compare register value (if ARL=1) or until it overflows (if ARL=0), at which moment the counter register will roll back to zero.
ARL	Auto ReLoad. Set to 1 to enable compare/autoreload mode.
T0IRQ	Timer interrupt pending service. Raised when the timer interrupt is triggered, cleared by writing 1 to it.

Registers TL,TH

	7	6	5	4	3	2	1	0
TL	Counter register value, bits 7..0							
reset	0	0	0	0	0	0	0	0
	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

	7	6	5	4	3	2	1	0
TH	Counter register value, bits 15..8							
reset	0	0	0	0	0	0	0	0
	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits marked 'r/w' can be read and written to by the CPU and can be reset by the core.

TH:TL This is the current value of the counter register. Will be reset to zero when TSTAT.CEN=0.

Registers TCL,TCH

	7	6	5	4	3	2	1	0
TCL	Compare register value, bits 7..0							
reset	1	1	1	1	1	1	1	1
	w	w	w	w	w	w	w	w

	7	6	5	4	3	2	1	0
TCH	Compare register value, bits 15..8							
reset	1	1	1	1	1	1	1	1
	w	w	w	w	w	w	w	w

Bits marked 'w' can be written to by the CPU but reading them will yield an undefined value.

TCH:TCL This is the current value of the reload register.

Input/Output Ports

The MCU includes 4 8-bit I/O ports. In order to save logic, the ports are hardwired to be either input or output, and are not configurable even at synthesis time -- it is simpler and cheaper to just add or modify whatever port setup is needed in each particular application than trying to provide for all possibilities in advance.

The port SFR addresses are the same as the original P0..P3 port addresses. However, since the ports are strictly input or strictly output, the behavior of the ports is different in a very important way:

The '**Read-Modify-Write**' feature of the MCS51 is **not implemented**:

- All instructions reading an input port read the pin regardless of addressing mode.
- All instructions reading an output port read the register regardless of addressing mode.

In short, writing to an output port will not have any effect. Reading an output port will access the port output registers and NOT the pins, as stated above.

The input ports are NOT registered or otherwise proof against metastability. A read instruction will read the instantaneous value of the input pin; then the value *will* be registered to internal register T before reaching its destination within the CPU, but there is some amount of logic between the port input and the T register, which means the port inputs have some non-negligible setup time, and not necessarily the same for all lines of the same port.

Subsequent versions of the core might register the input ports to minimize and equalize setup times.

Note too that the current version of the core does not have any external interrupt capability.

External Interrupt Inputs

The MCU has 8 external interrupt inputs which, in the current version of the ccre, are meant mostly for debugging.

The inputs are registered and level sensitive. As long as input external_irq[i] is high, flag EXTINT[i] will be high and the interrupt request line 0 of the CPU will be asserted.

Subsequent versions will use edge triggering and add a mask register.

Register EXTINT

This register contains the external interrupt pending flags:

	7	6	5	4	3	2	1	0
EXTINT	EIRQ7	EIRQ6	EIRQ5	EIRQ4	EIRQ3	EIRQ2	EIRQ1	EIRQ0
reset	0	0	0	0	0	0	0	0
	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c

Bits marked 'w1c' (write 1 to clear) are set by the core when an interrupt has been triggered and must be cleared by the software by writing a 1.

EIRQ<i>	External interrupt <i> pending service. Raised when the core input external_irq[i] is high, cleared by writing 1 to it as long as the input has been cleared too.