EduSoC v1.0 Documentation

Minimalistic RISC-V-oriented SoC for educational purposes

Alexander Kharitonov
University of Stuttgart, ITI

9th July 2023

Overview

EduSoC (Educational System on Chip) is an FPGA-based, RISC-V-oriented system-on-chip infrastructure. It is designed for educational purposes, offering minimal complexity and consisting of a small set of essential modules and peripherals:

- Clock and reset signal generation
- Boot ROM (size and speed configurable)
- Block RAM (size and speed configurable)
- SoC controller with interrupt engine supporting up to 32 interrupts
- GPIO module with change notification interrupt, supporting up to 512 GPIO pins
- Timer module with up to 16 independent timers with configurable periods and interrupts
- PWM module with up to 16 independent pulse width modulated outputs
- VGA controller with 8-bit color framebuffer (size and speed configurable)
- Serial UART bridge for external memory programming and SoC/peripheral control
- Crossbar bus interconnect

In its default configuration, EduSoC is intended for use with a Digilent Arty S7 board and offers a Verilog-compatible module for use with this board (`edusoc_basic`, see Appendix A). However, EduSoC can be configured for any hardware platform, and more peripherals can be added to EduSoC with relative ease.

EduSoC does not target or contain any specific RISC-V core. A simple core interface is provided.

Contents

List of Figures

List of Tables

1. System Structure

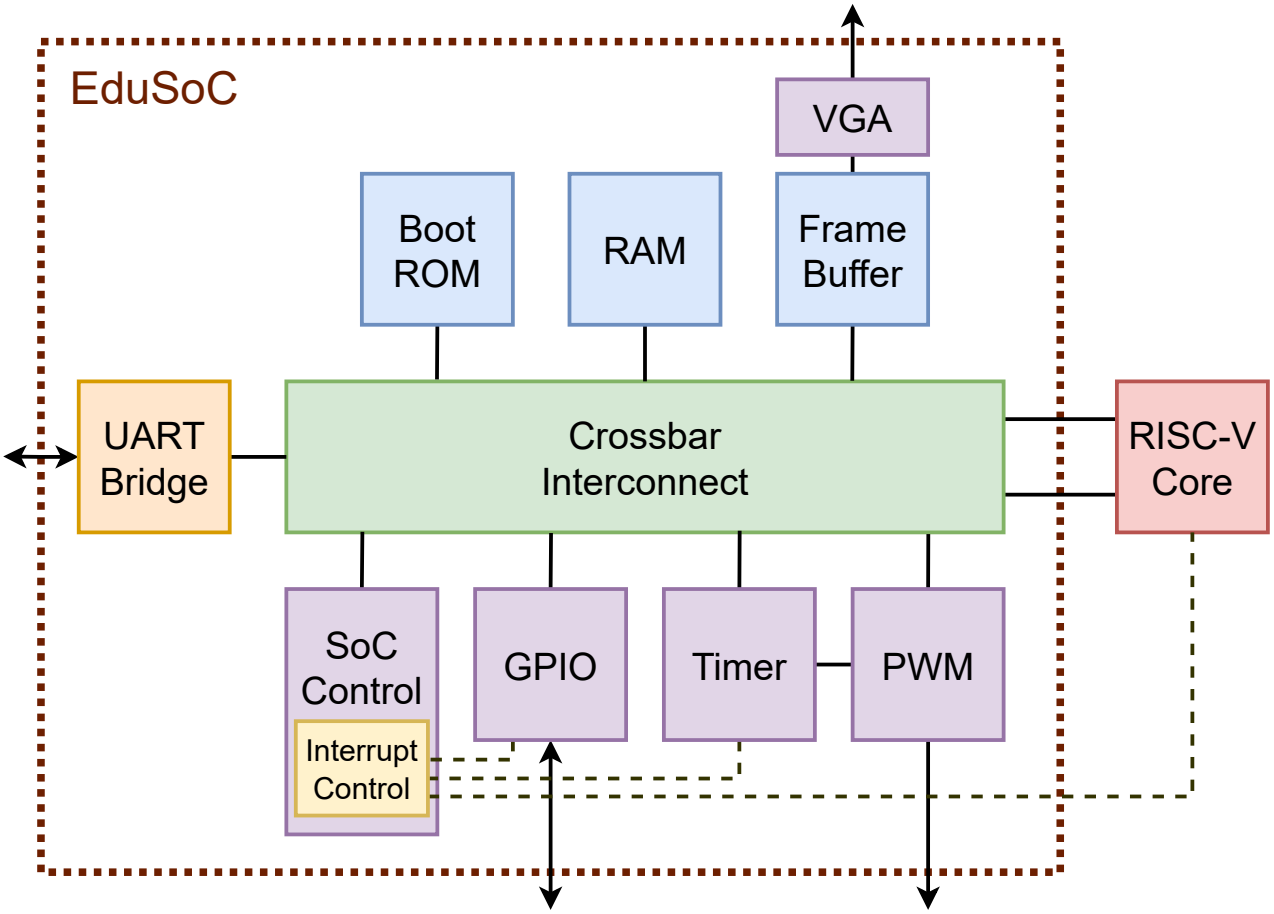

Figure 1: System Block Diagram

EduSoC provides all components within the dotted frame.

At its center, the crossbar interconnect serves as a bus connection between all system components. The interconnect arbitrates three bus masters (the UART bridge and the core's data and instruction buses), where the UART bridge has priority over the other two. The remaining SoC modules are bus slaves.

2. Memory Map

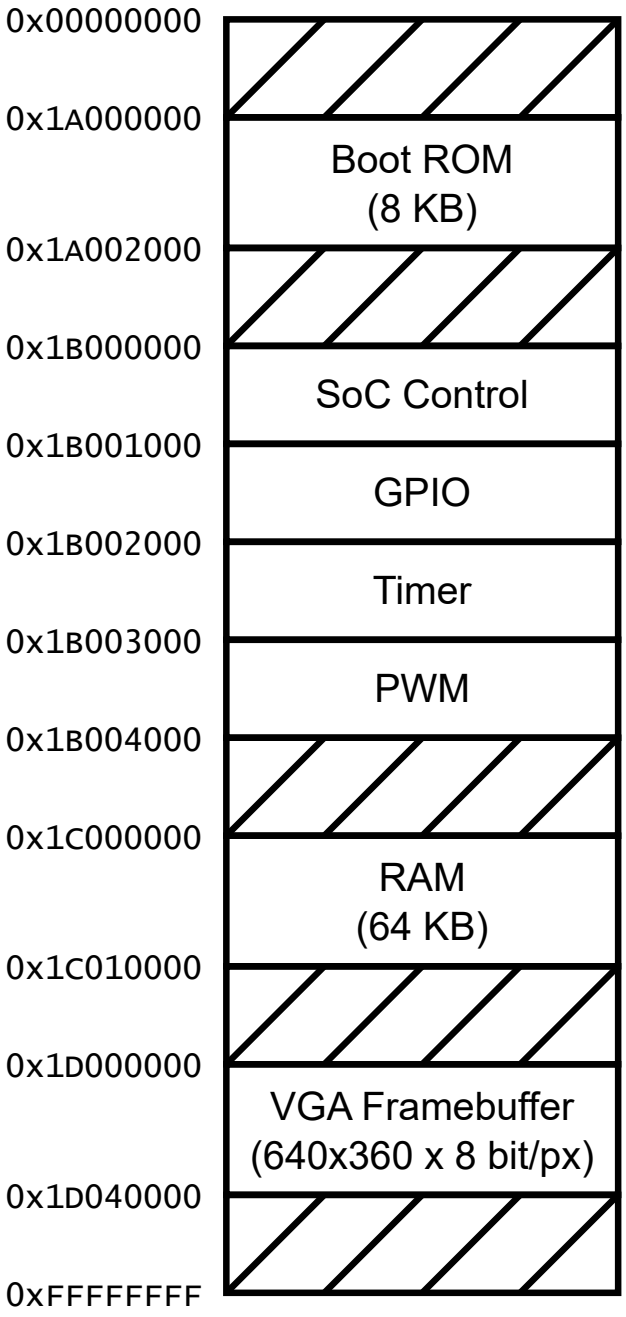

Figure 2: Default Memory Map

The above memory map represents the default configuration of EduSoC, and is fully configurable (see Section 8).

Hatched areas are unused. Writing to them has no effect, and reading from them produces undefined values.

3. Interfaces

3.1. Memory Bus

The EduSoC memory bus has the following signals:

Name	Width (bits)	Direction
`addr`	32	Master ⇒ Slave
`read_data`	32	Slave ⇒ Master
`write_data`	32	Master ⇒ Slave
`write_en`	1	Master ⇒ Slave
`byte_en`	4	Master ⇒ Slave
`req`	1	Master ⇒ Slave
`valid`	1	Slave ⇒ Master

Table 1: Memory Bus Signals

- `addr`: Requested byte address in the memory space. On the SoC level, only 4-byte-aligned requests are supported, so the lowest two address bits are ignored.
- `read_data`: Data read from the requested address. Only valid when `valid` = 1 (see below). Value is not defined for write requests.
- `write_data`: Data to be written to the requested address. Ignored for read requests.
- `write_en`: Whether the current request is a read request (0) or a write request (1).
- `byte_en`: Which of the four bytes of `write_data` should actually be written. Each bit corresponds to one of the four bytes (1 = write, 0 = keep unchanged), with the lowest bit corresponding to the lowest byte. Ignored for read requests.
- `req`: Set high (1) by the master to request bus access. Once set, all master signals should be held at a constant value until the `valid` signal is 1 (see below).
- `valid`: Set high (1) by the slave once the master's request has been completed. For read requests, this means valid read data is available on `read_data`, for write requests, this means the data from `write_data` has been written to the bus.

Once `valid` is asserted high (1), the request is considered complete and potential read data stays available until the request is terminated, either by resetting `req` to 0, or by changing `addr` or `write_en` (which constitutes a new request).

This also means that multiple consecutive requests may be made by keeping `req` = 1 and changing `addr` and/or `write_en` once the previous request is complete.

The following are example waveforms for read and write requests to the memory bus.

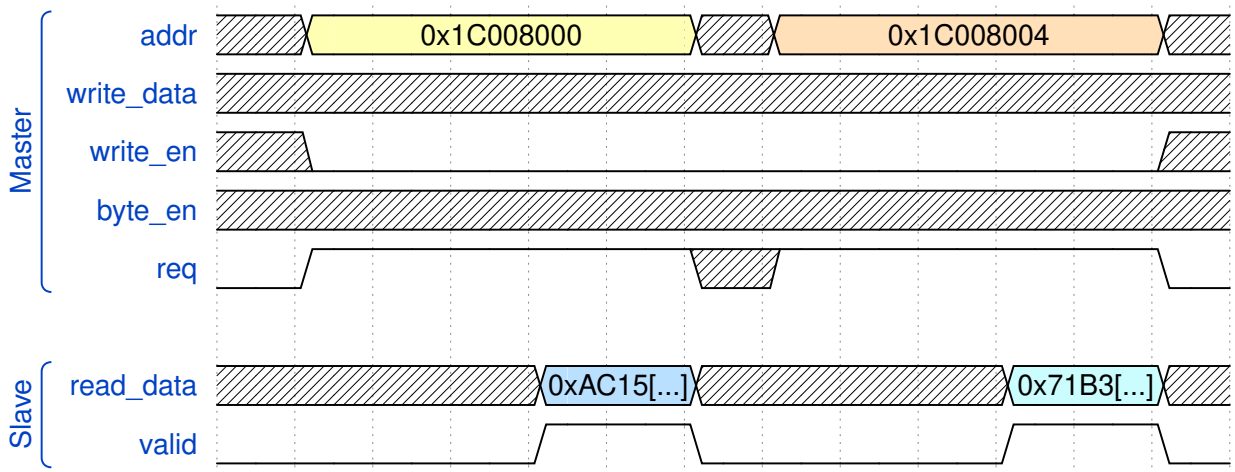

Figure 3: Example Read Waveform

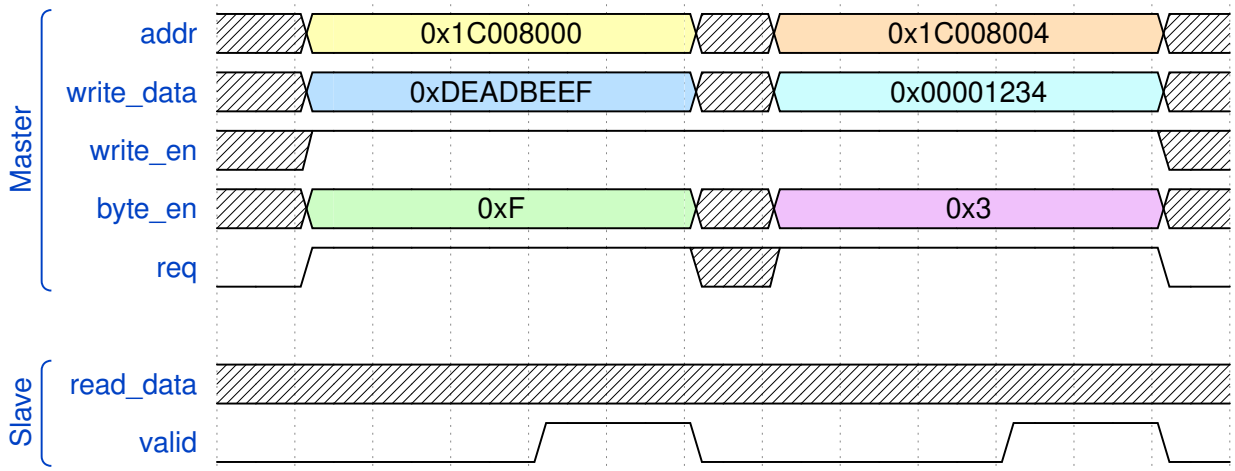

Figure 4: Example Write Waveform

3.2. Interrupt Bus

The EduSoC interrupt bus has the following signals:

Name	Width (bits)	Direction
`irq`	1	Generator ⇒ Handler
`irq_id`	5	Generator ⇒ Handler
`irq_ack`	1	Handler ⇒ Generator
`irq_ack_id`	5	Handler ⇒ Generator

Table 2: Interrupt Bus Signals

- `irq`: Set high (1) by the generator when an interrupt has occurred. Held until the interrupt is is acknowledged by the handler.

- `irq_id`: When `irq` = 1, conveys the ID of the interrupt to be handled (see Section 4).
- `irq_ack`: Set high (1) by the handler to signal that the currently asserted interrupt has been acknowledged and is being handled.
- `irq_ack_id`: When `irq_ack` = 1, conveys the ID of the interrupt that is being acknowledged.

3.3. SoC Interface

The SoC interface, consisting of the I/O interface and the CPU core interface, has the following signals and buses:

Name	Type or width (bits)	Direction
`ext_clk`	1	SoC input
`ext_resn`	1	SoC input
`vga_hsync`	1	SoC output
`vga_vsync`	1	SoC output
`vga_r`	4	SoC output
`vga_g`	4	SoC output
`vga_b`	4	SoC output
`uart_rx`	1	SoC input
`uart_tx`	1	SoC output
`gpio_in`	N	SoC input
`gpio_out`	N	SoC output
`gpio_drive`	N	SoC output
`pwm`	M	SoC output
`core_clk`	1	SoC output
`core_res`	1	SoC output
`control_flags`	16	SoC output
`instr_bus`	Memory Bus	SoC is slave
`data_bus`	Memory Bus	SoC is slave
`core_int_triggers`	16	SoC input
`int_bus`	Interrupt Bus	SoC is generator

Table 3: SoC Interface Signals

The following are central control signals for the system and must be connected. In a simulation environment, they may be left open, as they are driven automatically.

- `ext_clk`: Externally supplied clock signal. Must be driven by a 100 MHz clock.
- `ext_resn`: External reset (active low). Must be held high (1) and may optionally be asserted low (0) to reset the core and SoC.

Next is the VGA video output. For more information about these signals, see Section 6.

- `vga_hsync`: VGA horizontal sync signal.
- `vga_vsync`: VGA vertical sync signal.

- `vga_r`: VGA red color signal (16 levels).
- `vga_g`: VGA green color signal (16 levels).
- `vga_b`: VGA blue color signal (16 levels).

The following is the UART interface for external control and programming of the SoC. See Section 5 for more information.

- `uart_rx`: UART receive line (to SoC).
- `uart_tx`: UART transmit line (from SoC).

The following is the GPIO interface, with bit width $N = 32 \cdot \#\text{ports} \leq 512$. The three signals are designed to be combined into a single N-bit bidirectional I/O port. For more information, see Section 7.2.

- `gpio_in`: GPIO input signals.
- `gpio_out`: GPIO output signals.
- `gpio_drive`: GPIO driver controls. If a bit is 0, that pin of the port is an input ("high impedance"), reading from the corresponding `gpio_in` bit. If a bit is 1, that pin of the port is an output, writing to the corresponding `gpio_out` bit.

The following is the output for the PWM modules, containing M pulse width modulated signals, where $M \leq 16$ is the number of PWM modules in the SoC. See Section 7.4 for more information.

- `pwm`: PWM output signals.

Finally, the following is the CPU core interface. See Sections 3.1 and 3.2 for the memory and interrupt bus descriptions, respectively.

- `core_clk`: CPU core clock.
- `core_res`: CPU core reset, active high. Intended as a synchronous reset, so the core should be reset on the rising edge of `core_clk` if this signal is high (1).
- `control_flags`: User definable control signals for the core. See Section 7.1 for more information.
- `instr_bus`: CPU instruction memory bus. Intended to be used by the core to load program instructions from memory.
- `data_bus`: CPU data memory bus. Intended to be used by the core to read and write memory or peripheral register values.
- `core_int_triggers`: Core interrupt triggers. If asserted high, a corresponding interrupt will be triggered. See Section 4 for more information.
- `int_bus`: CPU interrupt bus.

4. Interrupts

EduSoC contains a simple interrupt controller to notify the core of asynchronous events. It supports up to 32 distinct interrupts, designated by interrupt IDs (0-31).

The priority order of the interrupts is fixed, where interrupts with a lower ID are forwarded to the core/handler first.

16 of these interrupts are SoC interrupts, generated by SoC modules and peripherals. The remaining 16 interrupts are core interrupts, which may be triggered by any events internal to the core. To trigger such an interrupt, the corresponding bit in the `core_int_triggers` signal (see Section 3.3) must be set high (1) for at least one clock cycle. The interrupt will stay asserted until it is acknowledged by the core (i. e. typically handled in software).

The interrupt indices are assigned as follows:

ID	Source	ID	Source	ID	Source	ID	Source
0	Core Interrupt 0	8	Reserved	16	Reserved	24	Core Interrupt 8
1	Core Interrupt 1	9	Reserved	17	Reserved	25	Core Interrupt 9
2	Core Interrupt 2	10	Reserved	18	Reserved	26	Core Interrupt 10
3	Core Interrupt 3	11	Timer	19	Reserved	27	Core Interrupt 11
4	Core Interrupt 4	12	Reserved	20	Reserved	28	Core Interrupt 12
5	Core Interrupt 5	13	Reserved	21	Reserved	29	Core Interrupt 13
6	Core Interrupt 6	14	Reserved	22	Reserved	30	Core Interrupt 14
7	Core Interrupt 7	15	GPIO	23	Reserved	31	Core Interrupt 15

Table 4: Interrupt Mapping

The core should always acknowledge any asserted interrupt as soon as possible, even if that interrupt is not supported or not handled by the core. Otherwise, other interrupts may be missed, especially lower-priority ones.

Interrupts are only asserted to the core if they are enabled first, and they may be cleared by a special register write. See Section 7.1 for more information on interrupt control.

5. UART Bridge

The UART bridge allows EduSoC to be programmed and controlled externally through a UART interface. The bridge functions as a memory bus master which is controlled by a UART protocol.

The UART data rate is 500000 Baud by default, but may be configured, see Section 8.

The UART bridge only supports writing to the memory bus in units of 32 bit words. Each 32 bit (4 byte) word is transmitted in little endian byte order. The UART protocol is as follows:

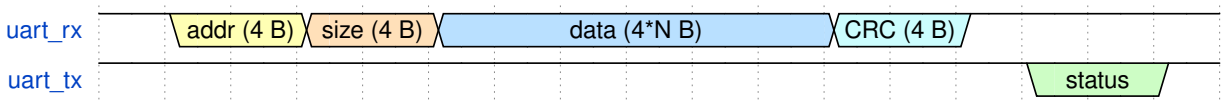

Figure 5: UART Bridge Protocol

First, the starting memory address is sent (i. e. the address of the first word to be written). As described in Section 3.1, only 4-byte-aligned addresses are supported. This is followed by 4 bytes containing the size N of the data to be written, **in units of 32 bit (4 byte) words** (*not* bytes).

The next N 32 bit words, i. e. $4 \cdot N$ bytes, are the actual data to be written, starting at the given address, with further words being written to the consecutively next memory addresses. Finally, the last 32 bit word is a CRC sum of the preceding data words (not including the address and size). The CRC algorithm used is CRC-32C (polynomial `0x1EDC6F41`, input and output reversed, initial value = final XOR = `0xFFFFFFFF`).

Once the bridge has received all of this data, it responds with a single byte indicating the status, which may be one of the following:

Value	Meaning
0x59	Success
0x23	CRC Mismatch
0xE0	Error
Other	Unknown (error)

Table 5: UART Status Codes

Once a success or CRC mismatch status has been sent by the bridge, it is ready for a new transmission. In the last two cases (error), the state of the UART bridge and the destination memory area is undefined, and the bridge may not respond to further UART transmissions. An external system reset is required to return to a known valid state.

In case of a CRC mismatch, data corruption during the transmission is possible. Note that this possibly corrupted data has been written to the destination memory area at this point, so it is recommended to retry the transmission to overwrite the corrupted data.

The EduSoC repository includes a Python script (`uart-programming/uart_upload.py`) implementing this UART interface for the Digilent Arty S7 board. See the header comment in the script itself for more information.

6. VGA Controller

EduSoC features a VGA video interface, requiring only an external DAC module for the analog color signals, for example the Digilent PmodVGA. By default, it is configured for a 640x480 video output at 60 Hz refresh rate, where only the central 640x360 area is backed by actual pixel data.

The VGA controller is connected to the framebuffer memory, reading one byte (8 bits) of framebuffer data for each pixel, in typical scanline order, and generating all VGA interface signals from this data and its given pixel clock. The pixel data format is RGB-332, so for each byte (= pixel), the upper three bits are the red intensity, the next three bits are the green intensity, and the lowest two bits are the blue intensity.

The framebuffer can be read and written to using the memory bus, like any other memory section, and changes to it will be visible on a connected screen immediately as soon as the changed pixels are drawn in the next frame. Note that, like any other EduSoC memory, the framebuffer contents are not affected by a system reset.

7. Peripherals

EduSoC contains multiple memory-mapped peripheral modules. They are accessible using the main memory bus and have registers to control their function, which are described in the following sections.

All registers in the following peripherals have four separate addresses each (differentiated by the last hexadecimal digit of the address), implementing functions for easy bit manipulations:

- `0x???????0`: Main register. Allows reading the register value and writing a value directly.
- `0x???????4`: SET register. Reads as 0. When written: For every high (1) bit in the written value, the corresponding register bit is set high (1). Low (0) bits in the written value leave the corresponding register bits unchanged.
- `0x???????8`: CLEAR register. Reads as 0. When written: For every high (1) bit in the written value, the corresponding register bit is cleared (set to 0). Low (0) bits in the written value leave the corresponding register bits unchanged.
- `0x???????C`: INVERT register. Reads as 0. When written: For every high (1) bit in the written value, the corresponding register bit is inverted ($0 \leftrightarrow 1$). Low (0) bits in the written value leave the corresponding register bits unchanged.

In the following register descriptions, only the main register address is given, but the other addresses still function as described above, by using the corresponding last address digit. The only exceptions are register bits which are read-only or may only be written in certain ways - the SET, CLEAR, and INVERT variants of those registers respect these restrictions, and will not perform any unsupported operations.

The register addresses given in the following register descriptions assume the default memory map as described in Section 2.

7.1. SoC Control

The SoC control peripheral implements functions for controlling the SoC and system as a whole, as well as controlling system interrupts.

It allows software- or UART-initiated resets of the core and SoC, allows the core to be halted, and allows detailed control over interrupts (enabling/disabling interrupts globally, enabling/disabling individual interrupts, reading interrupt status, clearing individual interrupts).

Additionally, a 16 bit core control flag signal is provided (see `control_flags` in Section 3.3). These flags may have a user-defined meaning and can also be read or written using the memory bus. Unlike most registers, these flags are *not* affected by system resets, retaining their state.
For the bootloader code and UART programming utilities provided with EduSoC, only one of these flags (flag 0) has a meaning by default - it serves as an indicator that a program has been loaded into RAM.

This peripheral has the following registers:

Address	Name	Reset Value
0x1B000000	SOCCON_CONTROL	0x00000008[1]
0x1B000010	SOCCON_INT_EN	0x00000000
0x1B000020	SOCCON_INT_FLAGS	0x00000000

Table 6: SoC Control Registers

7.1.1. `SOCCON_CONTROL` Register

Address: 0x1B000000
Reset value: 0x00000008[1]

Bit Range	Bits (highest to lowest)							
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	control_flags[15:8]							
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	control_flags[7:0]							
15:8	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
7:0	U-0	U-0	U-0	U-0	R/W-1	R/W-0	R/W-0	R/W-0
	—	—	—	—	INTGEN	SOCRES	CORERES	COREHLT

Legend:	R = Readable	W = Writable	U = Unimplemented, read as 0
-n = Initial Value	1 = Set	0 = Cleared	x = Unknown

Bit 31-16 `control_flags`: Core control flags, see description above. Reset retains values.

Bit 15-4 Unimplemented: Read as 0.

[1] Upper 16 bits are initially 0x0000, but retain their values during further resets.

Bit 3 `INTGEN`: Interrupt global enable.
0 = No interrupts will be asserted to the core, irrespective of individual interrupt enable settings.
1 = Individually enabled interrupts will be asserted to the core.

Bit 2 `SOCRES`: SoC software reset.
0 = No effect.
1 = SoC will be reset on the next clock cycle (including this register/bit).

Bit 1 `CORERES`: CPU core reset.
0 = No effect.
1 = CPU core reset line is asserted high (1), leading to a reset on the next CPU core clock cycle.

Bit 0 `COREHLT`: CPU core halt.
0 = CPU core clock is running.
1 = CPU core clock is halted. Note that this also prevents synchronous core resets from occurring.

7.1.2. `SOCCON_INT_EN` Register

Address: `0x1B000010`
Reset value: `0x00000000`

Bit Range	Bits (highest to lowest)							
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	`INT_ENABLE`[31:24]							
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	`INT_ENABLE`[23:16]							
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	`INT_ENABLE`[15:8]							
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	`INT_ENABLE`[7:0]							

Legend:	R = Readable	W = Writable	U = Unimplemented, read as 0
-n = Initial Value	1 = Set	0 = Cleared	x = Unknown

Bit 31-0 `INT_ENABLE`: Individual interrupt enable.
0 = Interrupt with the corresponding index will not be asserted to the core.
1 = Interrupt with the corresponding index will be asserted to the core whenever it occurs.

Each bit corresponds to one interrupt ID, e. g. bit 0 corresponds to interrupt ID 0.

When enabling an interrupt, if it has occurred in the past (i. e. its flag is 1, see `SOCCON_INT_FLAGS`), it will immediately be asserted to the core.

7.1.3. `SOCCON_INT_FLAGS` Register

Address: `0x1B000020`
Reset value: `0x00000000`

Bit Range	Bits (highest to lowest)							
31:24	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0
	`INT_FLAGS[31:24]`							
23:16	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0
	`INT_FLAGS[23:16]`							
15:8	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0
	`INT_FLAGS[15:8]`							
7:0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0
	`INT_FLAGS[7:0]`							

Legend:	R = Readable	W = Writable	U = Unimplemented, read as 0
C = Clearable	-n = Initial Value	1 = Set 0 = Cleared	x = Unknown

Bit 31-0 `INT_FLAGS`: Interrupt occurred flags.
0 = Interrupt with the corresponding index has not occurred.
1 = Interrupt with the corresponding index has occurred and will be asserted to the core if it is enabled.

Each bit corresponds to one interrupt ID, e. g. bit 0 corresponds to interrupt ID 0.

Flags/bits in this register may be cleared (set to 0) using memory writes, which will clear the corresponding interrupts and stop them from being asserted until they reoccur. Attempting to set bits to 1 using memory writes will have no effect.

7.2. GPIO

7.3. Timer

7.4. PWM

8. Configuration

A. `edusoc_basic`: Verilog-Compatible Interface for Arty S7

B. Revision History