HULOGIC2 Specification and Programmer's Reference

1 About this document

1.1 Summary

This document provides a specification and programmer's reference for the FPGA device designed for the Housekeeping Unit (IHU). Since the FPGA's functionality is tightly coupled to other parts of the IHU subsystem, it is important to read this document in conjunction with the relevant schematics.

1.2 Revision information

Rev	Date	Author	Description
0.0	21 Feb 2010	J Bromley	Initial release for discussion
1.0	23 Feb 2010	J Bromley	Updated for Issue 1A release to KIWISAT team

2 Overview

2.1 Hardware

The IHU circuit board is described in Drawing AMSATZL-2004031202 rev A00. The parts of that design that most strongly relate to the HULOGIC2 FPGA appear on sheet 3 (CPU), which shows the relationship between CPU and FPGA.

The CPU is an Intel TN80C188EB20 microcontroller, clocked by a 14.7456 MHz crystal connected to the CPU's integral oscillator. The CPU uses this clock to time all internal operations, and also puts out a clock at half this frequency (7.3728 MHz) on its CLKOUT pin. All CPU bus interface timing is specified with respect to CLKOUT.

The FPGA is a QuickLogic QL8x12B-1PL68M antifuse programmable FPGA.

2.2 FPGA functions

The FPGA has three major functions:

- Management of the IHU's triple-redundant static RAM array
- Interface to several MAX1202 A/D converters using an SPI bus
- Glue logic relating to the IHU's serial communications device and FEC block

These three functions are logically independent, but all are managed over the CPU's address/data bus and therefore they share a small amount of logic internally to the FPGA.

2.2.1 Triple redundant RAM

The error detecting and correcting triple-redundant RAM array (referred to as EDACRAM in the remainder of this document) consists of three identical blocks of memory, organised as two banks with independent chip-select lines (nECS0 and nECS1 on the schematic). On write cycles, the same CPU data is written to all three blocks simultaneously. On read, all three blocks are read and the three data bytes are compared in the FPGA. For each bit of the read data byte, a 3-way majority vote is used to determine the data presented to the CPU. In this way, single-bit soft errors in the RAM can be tolerated.

The majority voting logic also detects discrepancies and increments an error counter if there is dissent among the three RAM blocks. The error counter is available in the FPGA as a readable register for inspection by software.

2.2.2 A/D converter interface

The FPGA's SPI interface logic supports four separate groups of SPI peripherals. Each group is populated with two MAX1202 converter devices; a general-purpose I/O port, not implemented by the FPGA, is used to select between these two devices. Each MAX1202 device supports eight analogue input channels which it can sample one-by-one thanks to an internal 8-to-1 analogue multiplexer.

For each A/D converter to perform a conversion it is necessary to send an 8-bit command word, and then gather the 12 bits of conversion result, over the serial SPI bus. The FPGA facilitates this process by allowing software to write a conversion command and, later, to collect the conversion result from two 8-bit readable registers. A *busy* flag in one of these registers indicates whether the conversion result is valid.

2.2.3 Glue logic (1): Serial communications interface

The 85C30 serial communications device (SCC) used on the IHU has a simple SRAM-like interface with active-low chip select, read enable and write enable signals. However, owing to pin count limitations on the device, it has no hardware reset input. Instead, reset is accomplished by driving its write enable and read enable signals low simultaneously. The FPGA provides the necessary logic to do this, and also some conditioning logic to manage the timing of the SCC's enable signals.

2.2.4 Glue logic (2): General-purpose digital output

The FPGA supports two bits of general-purpose digital output, named MUX[0] and MUX[1]. A writeable register allows software to update these bits at any time. They are used by the IHU's FEC Encoder module.

3 Programmer's reference

3.1 EDACRAM

EDACRAM access through the FPGA is transparent; software does not need to be aware of the existence of the FPGA to access EDACRAM. The two 512Kx8 banks are enabled completely separately by the CPU's programmable chip select lines. One bank is enabled by the nLCS line, the other by nGCS[0]. The address range of both these chip selects can be freely chosen by software writing to the CPU's internal control registers, and does not concern us here.

3.2 SCC device

Although the SCC device is a separate chip and its functionality is not implemented in the FPGA, it is accessed through part of the FPGA's register bank. Consequently, its four read/write registers appear in the FPGA address map described below.

3.3 Register map

Apart from the EDACRAM, all other functionality supported by the FPGA is exposed through a 16-byte block of registers. These registers are enabled by CPU chip select line nGCS[3] and therefore the location of this block is determined by programming of the CPU's internal control registers. However, programmers should note that only the least significant four address lines are used to select the FPGA's registers. If the address range selected by nGCS[3] is larger than 16 bytes, the FPGA registers will appear multiple times in this address range; aliases will appear in each 16-byte block within the range selected by nGCS[3]. It is unimportant which of these aliases is accessed, but it is recommended that a convention be established to avoid possible confusion.

The address map of the FPGA-accessible registers is shown in the following table.

Address	Write	Read
0	SCC write register 0	SCC read register 0
1	SCC write register 1	SCC read register 1
2	SCC write register 2	SCC read register 2
3	SCC write register 3	SCC read register 3
4	FEC MUX	
5		
6		
7		
8	ADC command	
9	ADC clock control	EDAC error count
A		ADC result low byte and status
B		ADC result high byte
C		
D		EDAC error count with auto-clear
E		
F		

Important Note

Software should avoid making access to any unspecified location in this register map. To conserve FPGA resources, partial register decoding has been used. As a result, access to unspecified locations in the map can cause unpredictable behaviour because multiple registers may be accessed together.

3.4 Detailed description of registers

3.4.1 SCC registers

These registers are implemented by the 85C30 SCC device and are not discussed further here. Consult the device data sheet for details.

3.4.2 FEC MUX register (offset 0x4, write-only)

Writing to this register updates the MUX outputs used by the FEC logic. The register has no other effect, and can be written at any time. Its bit map is:

	MSB							LSB
bit	7	6	5	4	3	2	1	0
							MUX[1]	MUX[0]

The value of the register on reset is zero. The value written to the six unused bits has no effect, but it is strongly recommended to write them as zero for compatibility with possible future changes.

3.4.3 EDAC error count registers (offset 0x9 and 0xD, read-only)

Whenever a read access is made to EDACRAM, majority voting is used to determine the corrected value of each bit. If one of the three blocks yields a discrepant value in any bit position, an internal error counter is incremented at the end of the read operation. The EDAC error count registers give access to the current value of this count.

Only four bits of count are maintained; the upper four bits of the register read as zero. The count value is cleared to zero on system reset. When the counter reaches its maximum value of 15, further increments are suppressed; consequently, error counts in the range 0 to 14 correctly reflect the number of errors seen since the most recent reset, but a count of 15 should be interpreted as "15 or more".

The counter can be read through either of two register addresses. Reading through address 0x9 yields the current count value in a straightforward way, with no side effects. Reading through address 0xD, however, yields the current count value and then clears the counter as a side effect. This allows software to maintain an accurate cumulative count. There is no possibility of a further error occurring between the read and clear operations, since they are implemented atomically as part of the register read.

	MSB							LSB
bit	7	6	5	4	3	2	1	0
	0	0	0	0	4-bit error count value			

3.4.4 ADC clock control register (offset 0x9, write-only)

This register controls the divide ratio used to generate SPI clock from the CPU's 7.3728 MHz CLKOUT signal. The value written into the lower 4 bits of this register specifies the length of SPI clock high and low half-periods, as a number of CLKOUT cycles.

The register is cleared to zero on system reset. This zero value gives the longest possible SPI clock half-period. The relationship between register values and SPI clock period is shown in the table below. Note that writing 0 to the register is equivalent to writing 16.

Value written	Effective divisor	SPI clock half-period (us)	SPI clock frequency (MHz)
0 or 16	16	2.170	0.2304
1	**DO NOT USE**		
2	**DO NOT USE**		
3	3	0.407	1.2288
4	4	0.543	0.9216
5	5	0.678	0.7373
6	6	0.814	0.6144
7	7	0.949	0.5266
8	8	1.085	0.4608
9	9	1.221	0.4096
10	10	1.356	0.3686
11	11	1.492	0.3351
12	12	1.628	0.3072
13	13	1.763	0.2836
14	14	1.899	0.2633
15	15	2.035	0.2458

Important Note

Writing the value 1 or 2 to this register will cause improper operation of the SPI interface. Take care to write only values in the range 3 to 16, or zero.

This clock control register may be written at any time. In particular, it is acceptable to write to it whilst an SPI transaction is in progress, although this will cause the SPI clock frequency to change discontinuously during the transaction and should probably be avoided as it may degrade the A/D converter's analogue behaviour for that conversion.

The table above indicates "SPI clock frequency". However, it is important to be aware that the SPI clock does not run continuously. There is a burst of 20 high pulses on SPI clock during each A/D converter transaction. For each of those pulses, the SPI clock is high for the specified half-period and then low for the same time before the next high pulse. There can be an arbitrarily long interval between transactions, during which the SPI clock idles at a low level.

Because of the relatively slow output delays of the MAX1202 devices, it is strongly recommended that clock divisor values of 6 or greater should be used in practice.

3.4.5 ADC command register (offset 0x8, write-only)

	MSB							LSB
bit	7	6	5	4	3	2	1	0
				SPI channel		Converter input		

Writing to the ADC command register causes an SPI transaction to begin. Bits [4:3] of the register value determine which of the four SPI channels will be used for this transaction. Bits [2:0] of the register value are passed to the selected A/D converter device to determine which of its eight inputs will be sampled on this conversion. Bits [7:5] are unused, but should be written as zero to preserve compatibility with possible future changes.

The MAX1202 device uses a rather curious coding scheme to drive the select inputs of its 1-of-8 analogue multiplexer from its 3-bit converter selection value. The HULOGIC2 FPGA normalises this coding scheme. The 3-bit converter input value written to the ADC command register controls the analogue multiplexer according to the following table:

Command register bits [2:0]	Chosen input (as described in MAX1202 datasheet)	Channel number on ADC Module	
		SEL=1	SEL=0
0	0	1	9
1	1	2	10
2	2	3	11
3	3	4	12
4	4	5	13
5	5	6	14
6	6	7	15
7	7	8	16

When a value is written to this register, the FPGA composes the appropriate A/D command value, activates the SPI chip select line chosen by the SPI channel value, and starts to run the 21-bit SPI transaction. This transaction sends the A/D command, and then receives 12 bits of converter result into the result registers (described below).

If this register is written when an SPI transaction is in progress, the write has no effect. Software should check the "busy" status bit is clear in the ADC result and status register (see below) before attempting to write this register. In practice it is probably appropriate for software to access the ADCs under control of a timer interrupt. On each interrupt, software should first read the conversion result data from the previous activation. It can then write a new command to the ADC command register. The frequency of interrupts should be chosen so that there is enough time for the SPI transaction to complete (more than 42 half-periods of the SPI clock) between interrupts. It should then be unnecessary for software to poll the busy flag.

3.4.6 ADC result and status registers (offset 0xA and 0xB, read-only)

These registers give access to the most recent SPI transaction. The least-significant byte register (offset 0xA) includes the less significant 4 bits of the 12-bit A/D result, and some status bits. The more significant byte (offset 0xB) contains the most significant 8 bits of the 12-bit A/D result. If either of these registers is read whilst an SPI transaction is in progress, the A/D conversion result will be undefined. However, the status bits in register 0xA are valid at all times.

The status bits of register 0xA comprise two fields, CHANNEL NUMBER and BUSY. The BUSY flag is true throughout an SPI transaction. It goes true when the ADC command register (see above) is written, and goes false 21 SPI clocks later when the transaction is complete. If the BUSY flag is true, the A/D conversion results are invalid. Software should wait until BUSY goes false before reading A/D conversion results. Throughout the transaction, and at all times until the next ADC command is issued, the two-bit Channel Number field reliably reflects the SPI channel that was activated by the most recent transaction (it is a copy of the value written to bits [4:3] of the ADC command register).

A/D result high byte (offset 0xB)

	MSB							LSB
bit	7	6	5	4	3	2	1	0
	Most significant bits of ADC conversion result (bits [11:4])							

A/D result low byte (offset 0xA)

	MSB							LSB
bit	7	6	5	4	3	2	1	0
	Least significant bits of ADC conversion result (bits [3:0])				0	Channel Number		BUSY