Reverse Engineering the DDR test of the SPEC-board PTS

Introduction

The SPEC board DDR is tested by executing the test07.py program. This SDRAM memory integrated circuit is the MT41J128M16HA-15E, a 2Gbit DDR3 which is externally (pin interface) configured as 128Mwords of 16bits in 8banks (3bank address lines and 14row/column shared address lines).
This Python test program makes use of the rr.py and gn2124.py modules to access the DDR.

Driver

The FPGA is accessed through the PCIe bus using the GN2124 integrated circuit. This interface is controlled by the application software making use of its corresponding kernel module (rawrabbit.ko).
The application software accesses the GN2124 driver by opening its device (/dev/rawrabbit) and making ioctl calls through the rrlib.so library.

Basic R/W Library Functions

To make read and write operations in the bus (host, FPGA (wishbone) or GN2124 registers), this library provides two functions: rr_iread and rr_iwrite. These functions are wrapped by the rr.py module functions: Gennum.iread and Gennum.iwrite respectively.
iread(self, bar, offset, width)
iwrite(self, bar, offset, width, datum)
where:
bar = 0, 2, 4 (or c for DMA buffer access). These values correspond to the addresses:
0x00000000 (GN4124 core registers: RR_BAR0)
0x20000000 ()
0x40000000 (GN4124 chip registers: RR_BAR4)
0xc0000000 (Host registers: RR_BAR12, for DMA items storage)
offset = address within bar
width = data size (1, 2, 4 or 8 bytes)
datum = value to be written

These functions are wrapped by the following functions of the gn2124.py module:
def **rd_reg(self, bar, addr)**: return self.bus.iread(bar, addr, 4)
def **wr_reg(self, bar, addr, value)**: self.bus.iwrite(bar, addr, 4, value)

DDR-Access Library Functions

The program uses the following functions to write and read from the DDR:

- gennum.**set_memory_page**(page_nb, pattern): Set the content of memory pages in the host (2^{10} = 1024pattern=4096bytes) by executing wr_reg(self.HOST_BAR=0xc, (page_nb<<12)+(i<<2), pattern). Each page has 2^{12}=4096 memory positions. Each pattern has 2^2=4bytes=32bits.
- gennum.**add_dma_item**(self, carrier_addr, host_addr, length, dma_dir, last_item): Add DMA item (first item is on the board, the following in the host memory)
  carrier_addr = carrier (DDR) address in bytes (to access concrete memory words (of 16bits), multiply word number by 2)
  host_addr = host memory page physical address in bytes
  dma_length = 0x4 (DMA length in bytes)
  dma_dir = 0 or 1 (0 -> carrier to PCIe, 1 -> PCIe to carrier)
  dma_last = 0 (0 -> last item in the transfer, 1 -> more item in the transfer)
  Only supports 32-bit host address
- gennum.**start_dma**():Start DMA transfer
- gennum.**wait_irq**():Wait for interrupt
- gennum.**get_memory_page**(page_nb): return the full content (1024words of 32bits) of a memory page in the host.

Initialization

Objects of the python modules (rr.py and gn2124.py) are created in order to access the board:

spec = **rr.Gennum**() # loads the library for accessing the device driver
gennum = **gn4124.CGN4124**(spec, GN4124_CSR)

Set the local bus frequency to 160MHz by means of a wr_reg function call in the RR_BAR_4:

gennum.**set_local_bus_freq**(160)

Get host memory pages physical address of the pages allocated to GN4124. These addresses comprise a list of pages for DMA access. The addresses of this list are shifted left by 12 bits to represent the physical addresses (32bits) of the allocated pages. This list is obtained by means of the rr_getplist(self.fd, plist) function of the rrlib.so library.

pages = gennum.**get_physical_addr**()

Memory operation

The SDRAM is internally arranged as 8banks of 2Mwords of 128bits (2Gbits in total). However, externally it is accessed in words of 16bits. To select a word address, the row address (2^{14}=16M) (and bank (2^3=8)) is specified (by means of the ACTIVATE command) and then the column (2^{10}=1M) (and bank) (READ or WRITE command). The definition of a complete address of a memory word depends on the DDR core (ddr3_ctrl) configuration. In the spec_ddr_test_top.vhd, the g_MEM_ADDR_ORDER parameter can be set to “BANK_ROW_COLUMN” or “ROW_BANK_COLUMN”. Currently it is set to "ROW_BANK_COLUMN", therefore, the address is formed as {ROW[13:0], BANK[2:0], COLUMN[9:0]}.

DDR-testing procedure

Set the content of the first three memory pages:
gennum.**set_memory_page**(0, 0x0)
gennum.**set_memory_page**(1, 0xDEADBABE)
gennum.**set_memory_page**(2, 0x0)

Testing of Address Lines

Since the same PCB tracks are used to set the row and column of a memory address, only the address corresponding to different rows needs to be checked (there are more rows than columns). Therefore, the total address lines to be checked should be 17 (14rows plus 3banks).
* In test07.py the total address lines checked are 18 (19-1) (possible bug?).
The procedure of testing address lines consists in checking that writing a pattern in a concrete memory position can be correctly read back and does not interfere with patterns written in other memory positions. The addresses containing only one bit set to 1 and the addresses containing only one bit set to 0 are checked independently.

Firstly, test07 checks if some pin is tied to GND:

- For each address_bit_i line in [1...19-1]:
  - Clear all the memory positions where the address has only one bit set to 1:
    For each address_bit_j in [1...19-1]:
    - DDR[$2^{address\_bit\_j}$] = 0

    Cleared carrier addresses (DDR):

Addr. in wishbone bus (32bits words)		Addr. for DDR inter. (16bits words)	
Dec. (bytes)	Bin. (bytes)	Dec. (words)	Bin. (words)
2^1=2	00…00010	2^0=1	00…00001
2^2=4	00…00100	2^1=2	00…00010
2^3=8	00…01000	2^2=4	00…00100
…			
2^{19-1}=256M ¿not 2^{17}?	10…00000	2^{17}=128M ¿not 2^{16}?	01…00000

* In order to check all the PCB address tracks, the bus addresses corresponding to the row and banks should be checked. Since these are the most significant bits, the first bit to be checked should be bit 10. Therefore, the addresses in the table above should be multiplied by 2^{10}=1K. Thus, the above and following memory access lines should be DDR[$2^{address\_bit+10}$] (possible bug?).

  - DDR[$2^{address\_bit\_i}$] = 0xDEADBABE
  - Read in different pages all the written memory positions:
    For each address_bit_j in [1...19-1]:
    - Page[3+address_bit_j] = DDR[$2^{address\_bit\_j}$]
  - Check that only the memory position $2^{address\_bit\_i}$ contains 0xDEADBABE:
    For each address_bit_j in [1...19-1]:
    - If address_bit_j = address_bit_i Then:
      - If Page[3+address_bit_j][0] <> 0xDEADBABE Then Error
      Else

- If Page[3+address_bit_j][0] <> 0 Then Error

Secondly, test07 checks if some pin is tied to Vcc:

- For each address_bit_i line in [1...19-1]:
  - Clear all the memory positions where the address has only one bit set to 0:
    For each address_bit_j in [1...19-1]:
    - DDR[NOT $2^{address\_bit\_j}$] = 0
      * Apart from being multiplied by 2^{10}, should the DDR address be masked with 2^{28}-1 (27bits of the DDR interface address (words) plus 1bits due to specifying this address in the bus (bytes)): DDR[(NOT $2^{address\_bit\_j}$) AND 2^{28}-1]?
  - DDR[NOT $2^{address\_bit\_i}$] = 0xDEADBABE
  - Read in different pages all the written memory positions:
    For each address_bit_j in [1...19-1]:
    - Page[3+address_bit_j] = DDR[NOT $2^{address\_bit\_j}$]
  - Check that only the memory position (NOT $2^{address\_bit\_i}$) contains 0xDEADBABE:
    For each address_bit_j in [1...19-1]:
    - If address_bit_j = address_bit_i Then:
      - If Page[3+address_bit_j][0] <> 0xDEADBABE Then Error
      Else
      - If Page[3+address_bit_j][0] <> 0 Then Error

Testing of Data Lines

Since the DDR content is accessed in 16bit words through the I.C. pin interface, only 16 PCB data tracks need to be checked.
The procedure of testing data lines consists in checking that writing a pattern which contains any of its bits to 1 does not interfere with the rest of data bits (set to 0).

- For each data_bit_i line in [0...16-1]:
  - DDR[0]=$2^{data\_bit\_i}$
  - If DDR[0] <> $2^{data\_bit\_i}$ Then Error