AN02021: Using external memory with XCORE.AI

Publication Date: 2024/10/16
Document Number: XM-014230-AN v1.0.0

XMOS

IN THIS DOCUMENT

1 External memory overview

xcore.ai processors can be connected to external LPDDR-1 memory. This document describes

- How to set-up the XTC tools to recognise the external memory
- How to use the external memory from software
- What performance to expect
- How to connect the memory in your design schematic
- Recommended PCB layout

This application note also has an accompanying software source file containing an a simple example to illustrate how external memory access can be used.

xcore.ai processors have a built-in LPDDR-1 controller and PHY that, when activated, maps an external LPDDR-1 memory into the address space of one of the xcore-processors. You can connect an LPDDR-1 memory to the PHY, as long as it has a 16-bit data-bus, complies with JEDEC standard JESD209B, and is 128, 256, 512, or 1024 Mbits in size. LPDDR-1 is also known as Mobile-DDR and it uses 1.8V signalling levels. xcore.ai supports LPDDR1 operating at a clock frequency of up to 166 MHz.

In an xcore.ai design with external memory, you can configure the XTC tools to place functions and/or data in external memory. External memory is (much) slower than internal memory, but offers a way to work with large amounts of data and code.

2 Setting up the tools to recognise external memory

XTC Tools version 15.1 or later can automatically configure the memory for you. Memory is configured by default on the XK-EVK-XU316 evaluation board that comes with LPDDR-1 included.

2.1 Setting up the configuration file

For the tools to recognise the external memory, you need to change the XN configuration file to include the external memory. In its simplest form, you specify the size of the memory and the LPDDR clock frequency at which it should operate. The `sizeMbit` attribute defines the size of the external LPDDR in megabits. Values of 256/512/1024 are

allowed. By correctly supplying this value, the xcore will raise a HW exception if out-of-bounds memory access occurs. The clock frequency is specified in Hz or typically MHz:

```
<Node Id="0" InPackageId="0" Type="XS3-L16A-1024"
      Oscillator="24MHz" SystemFrequency="600MHz" ReferenceFrequency="100MHz">
  <Extmem sizeMbit="1024" Frequency="100MHz">
    <Padctrl clk="0x30" cke="0x30" cs_n="0x30" we_n="0x30"
             cas_n="0x30" ras_n="0x30" addr="0x30" ba="0x30"
             dq="0x31" dqs="0x31" dm="0x30"/>
    <Lpddr emr_opcode="0x20" protocol_engine_conf_0="0x2aa"/>
  </Extmem>
  <Tile Number="0" Reference="tile[0]"/>
  <Tile Number="1" Reference="tile[1]"/>
</Node>
```

In this simple case, the clock frequency must be a precise even division of the system clock, and should not exceed 166 MHz. If you try and use a frequency that is not an even division of the system clock, for example 166 MHz, the system will generate a warning.

An example **XN** file that uses the system PLL can be found in `src/system-pll.xn`. The `CMakeLists.txt` file has a line that can be commented in to use this file.

If you cannot generate the desired frequency from the system clock, for example you want to use a 166 MHz DDR clock and a 600 MHz system clock, you can use the secondary PLL to generate a clock for the memory. This is achieved by setting the secondary PLL up in the XN file, and stating that the external memory should use the secondary PLL with the specified divider:

```
<Node Id="0" InPackageId="0" Type="XS3-L16A-1024"
      Oscillator="24MHz" SystemFrequency="600MHz" ReferenceFrequency="100MHz"
      SecondaryPllInputDiv="1" SecondaryPllOutputDiv="3" SecondaryPllFeedbackDiv="83">
  <Extmem sizeMbit="1024" SourcePll="SecondaryPll" Divider="2">
    <Padctrl clk="0x30" cke="0x30" cs_n="0x30" we_n="0x30"
             cas_n="0x30" ras_n="0x30" addr="0x30" ba="0x30"
             dq="0x31" dqs="0x31" dm="0x30"/>
    <Lpddr emr_opcode="0x20" protocol_engine_conf_0="0x2aa"/>
  </Extmem>
  <Tile Number="0" Reference="tile[0]"/>
  <Tile Number="1" Reference="tile[1]"/>
</Node>
```

An example **XN** file that uses the secondary PLL can be found in `src/secondary-pll.xn`. The `CMakeLists.txt` file has a line that can be commented in to use this file.

For more details on how to set up and control the PLL, see the application note on xcore.ai Clock Frequency Control.

Two sub-elements are required inside the `<Extmem>` element: `<Padctrl>` and `<Lpddr>`.

`Padctrl` is used to explicitly change the settings of the pad and, for example, to increase the drive-strength of specific pads:

```
<Padctrl clk="0x30" cke="0x30" cs_n="0x30" we_n="0x30"
         cas_n="0x30" ras_n="0x30" addr="0x30" ba="0x30"
         dq="0x31" dqs="0x31" dm="0x30"/>
```

The values passed in each element of Padctrl are port-settings and are specified in the xcore.ai device datasheet.

The `<Lpddr>` element is used to specify values for the 18 LPDDR registers. More details on these registers are provided in the xcore.ai device datasheet. An obligatory element is the `emr_opcode`:

```
<Lpddr emr_opcode="0x20"/>
```

2.2 Compiler flags required

In order to use external memory, you must compile with the large memory model. This is achieved by adding `-mcmodel=large` to the compiler flags. The use of external memory requires the use of a much larger address space than that is normally used. The larger RAM and the presence of memory at differing base addresses means that the successful use of immediates cannot now be assumed by the compiler.

For completeness, if external memory is not used then there are two other memory models available:

- `-mcmodel=small` This is the default behaviour that minimizes code size and maximizes speed.
- `-mcmodel=medium` Uses small offsets for code and data in the same source file for objects that are not explicitly placed in a section.

`-mcmodel=large` Uses absolute addresses for all code and data. This option allows sparse placement of code/data across the memory space. In terms of code speed and code size the **small** model is the most efficient, and the **large** model the least efficient. It depends on the relative sizes of code and data as to which model is required.

3 Using external memory in software

The external memory is available in the core that uses it from address **0x1000 0000** to **0x1FFF FFFF**, and can be used just like ordinary internal memory, except that accesses to external memory will be slower. That range assumes a 128 Mbit memory, for smaller memories the top of the range should not be used. Only one of the two physical cores should be using external memory, otherwise an error will be raised by the XTC compilation/build tool.

The declaration below shows how to place data in external memory:

```
__attribute__((section(".ExtMem_data")))
unsigned int array_ext[MEMCPY_EXT_BLOCK_BYTES / sizeof(int) * BLOCKS] = {1,5,20};
```

The code below shows how to place code in external memory:

```
__attribute__((section(".ExtMem_code")))
unsigned int add_six(unsigned int a) {
    return a + 6;
}
```

In this snippet of code, the array `array_ext` and function `add_six` are placed in external memory. The attributes `__attribute__((section(".ExtMem_data")))` and `__attribute__((section(".ExtMem_code")))` affect the declaration immediately following the attribute.

4 What performance to expect

External memory is limited to the speed specified in the datasheet for the memory device you use, however xcore.ai devices do not support clock rates over 166 MHz.

This maybe sufficient for many applications. But there is a significant latency and a reduction in bandwidth when making accesses to LPDDR compared with accesses to internal memory. And these factors are particularly noticeable when using the vector unit. A vector load/store from external memory requires at least 10 clock cycles at 166 MHz, whereas internal memory can provide a vector every clock cycle on each tile. Internal memory has close to 80x more bandwidth than external memory.

External memory is routed through a very small buffer that enables the programmer to, for example, walk through an array one byte at a time without each byte being loaded separately. The behaviour is one of a cache with eight lines (each with eight words), using LRU replacement. A prefetch instruction is available that will initiate a fetch from memory into the buffer. This instruction is needed to get maximum performance.

Given the very small size, it is best to think of it as a buffer that can be used for one of the following:

- Execute code in one thread directly from external memory
- Walk through data in external memory sequentially
- Write data in external memory sequentially

Using data in a random order (for example by having two threads execute code from memory, or by writing random bytes), will result in lower performance.

For best performance, data should be accessed on 32-byte boundaries. An example function that copies data at maximum performance is written in assembly in `memcpy_ext.S` with an associated header file `memcpy_ext.h`. In order to attain full performance, a `prefetch` instruction is used to start loading the next line from LPDDR early.

The main program times copying data and shows data can be read at around 420 MByte/s using a 166 MHz LPDDR RAM on a 600 MHz processor.

5 Connecting memory in your design schematic

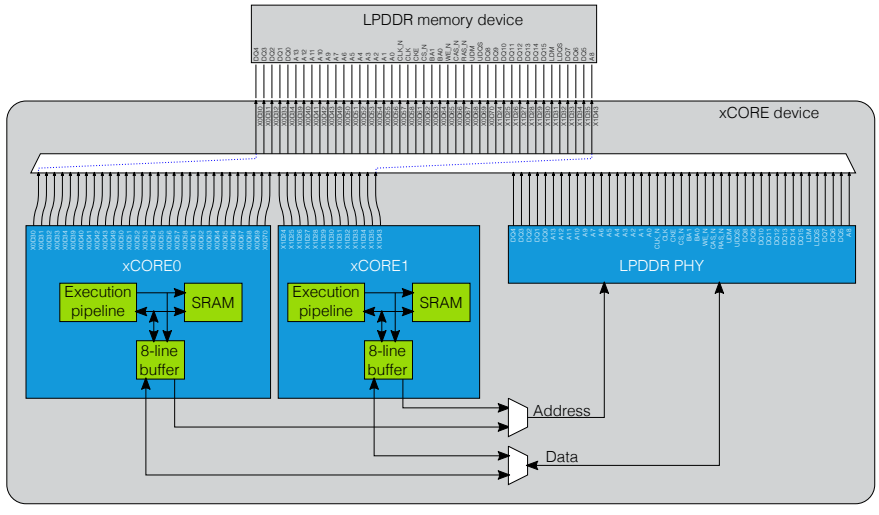

Fig. 1: LPDDR pin muxing

The LPDDR-1 controller is located in the VDDIOT voltage domain of the xcore.ai device, and is only pinned out in large packages. Check the xcore.ai device datasheet to verify that LPDDR is pinned out on a particular device. When using the LPDDR controller, you should supply 1.8V to VDDIOT, and ground the LV_T_N pin to signal that the top IO domain is “Low Voltage”.

The pin layout and the GPIO muxing are shown in *LPDDR pin muxing*. Inside the device, the LPDDR controller shown on the right has 43 pins (DQ0..15, A0..13, and 13 control pins). These are muxed by the mux with 43 IO pins (30 GPIO pins from core 0, 13 GPIO pins from core 1), and the muxed signals then connect to 43 package pins. These package pins can be used as GPIO (this is the default), or an LPDDR device can be wired up to it as shown. The pins are hard-coded, so if you want to use LPDDR you must connect pin X0D30 to DQ4, pin X0D31 to DQ3, etc.

The schematics are shown in *LPDDR schematics*.

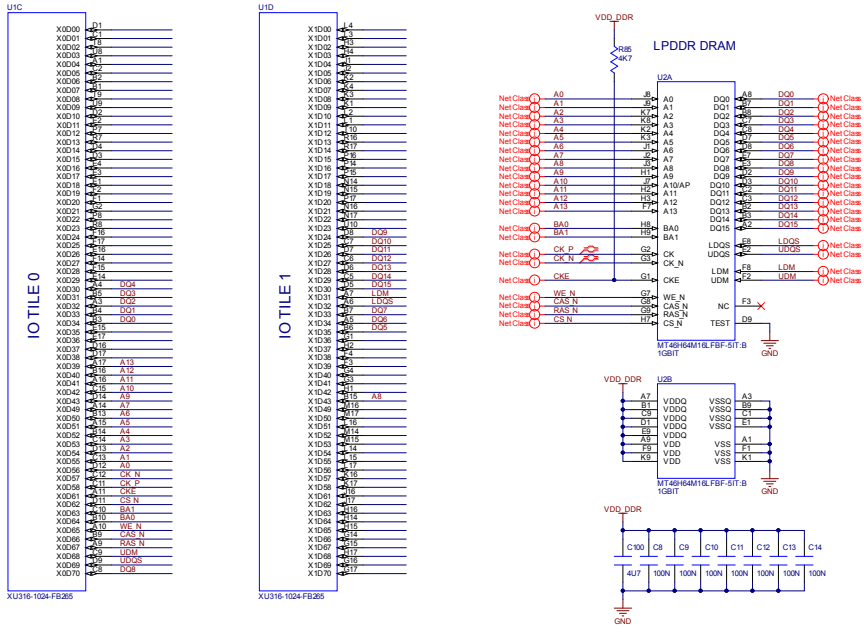

Fig. 2: LPDDR schematics

6 Recommended PCB layout

In order for the external memory to perform reliably, it is important to route the signal wires from xcore.ai to LPDDR very carefully and to put them right next to each other. We identify four key timing groups:

Data rate	Group end-points
Double data rate Data Write	DQ and DM to DQS
Double data rate Data Read	DQ to DQS
Single data rate	Address/Command to CK/CK#
Cross domain	CK/CK# to DQS

Traces should be a maximum of 2 inches in length. You can have the following variations:

- CK to CK# trace length: ±20 mil. (Matching the differential pair).
- CK/CK# trace lengths to DQS trace length: ±500 mil.
- Data group low (DQ0..7, LDM and LDQS): matched within ±50 mil of each other.
- Data group high (DQ8..15, UDM and UDQS): matched within ±50 mil of each other.
- Address/Control/CK group: matched within ±400 mil of each other.
- Data groups low and high: matched to within ±500 mil of each other.

As a guideline, the layout that XMOS uses is shown in *Routing of signals on bottom* to *Routing of signals on all layers*. This shows three of the layers on which signals are routed. The Gerber file for the XK-XU316-EVK board that uses LPDDR can be downloaded from the xmos.com website.

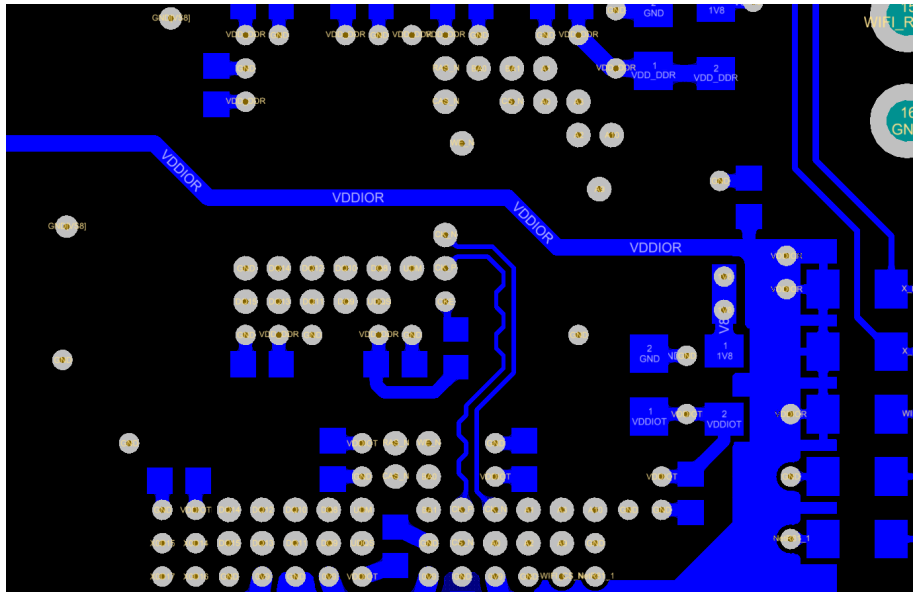

Fig. 3: Routing of signals on bottom

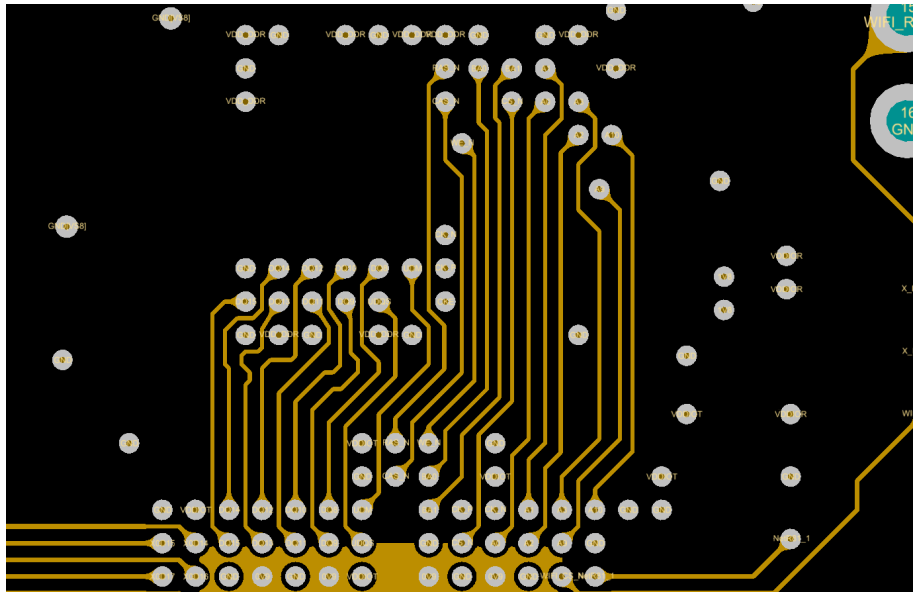

Fig. 4: Routing of signals on signal 2

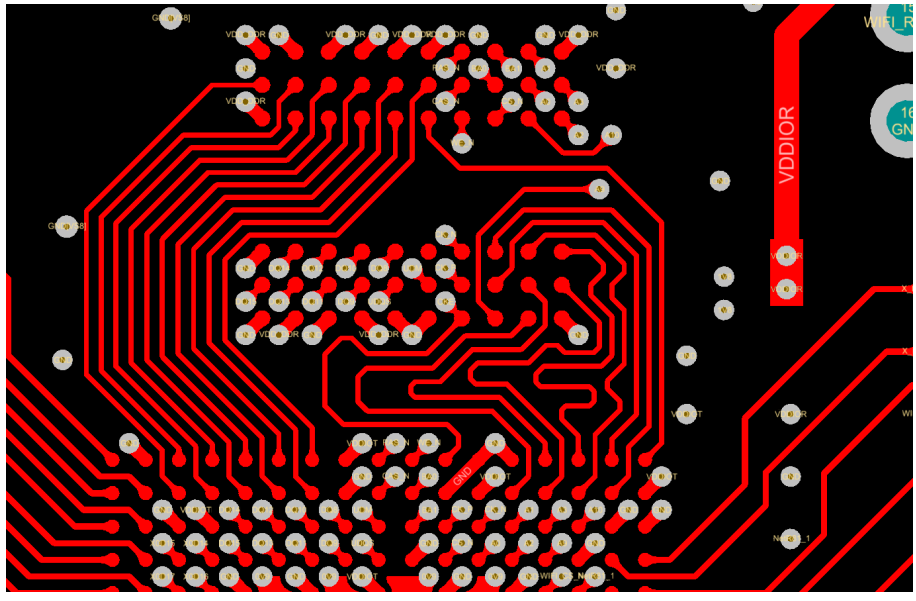

Fig. 5: Routing of signals on top

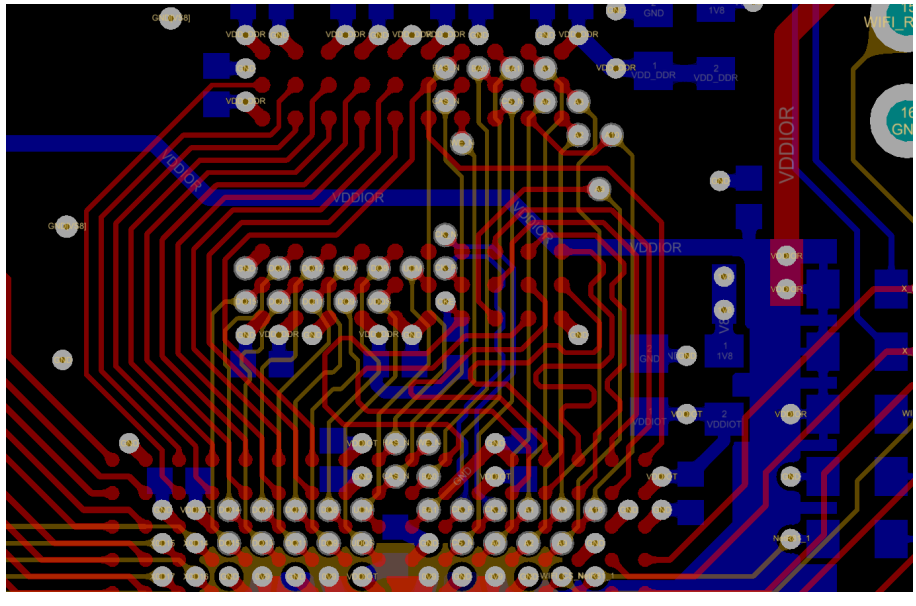

Fig. 6: Routing of signals on all layers

7 Errata

Some VPU instructions should not access external memory unless data is guaranteed to be present in the buffer: VLADD, VLSUB, VLMUL, VLADDSB, VLMACC, VLMACCR, VLMACCR1, and VLSAT. For correct operations, these instructions should operate from internal memory only.

8 Further information

- XTC Tools Guide
- The XS3 architecture manual
- XU316-1024-FB265 datasheet
- xcore.ai Clock Frequency Control
- Design and manufacturing data for XK-XU316-EVK

Note that only some packages expose the LPDDR controller GPIO.

9 Document History

Date	Release	Comment
2024-09-12	1.0.0	First release