TMS320 DSP
DESIGNER'S
NOTEBOOK

Preliminary

TEXAS
INSTRUMENTS

Wait-states on TMS320C6201 CPU Data accesses versus various memory types.

Contributed by Eric Biscondi

Design Problem How many cycles is CPU stalled when performing CPU data access to the external memory?

Solution Depending on the memory type, the time required to complete an external memory access varies. The pipeline may stall the CPU during a various amount of cycle (wait-states) to wait for the external memory. (Cf. CPU and Instruction Set section 4.3.3.1) as shown in Figure 1. When performing CPU data access, wait-states are inserted due to the pipelined nature of the Data Memory Controller's interface to the External Memory Interface (EMIF).

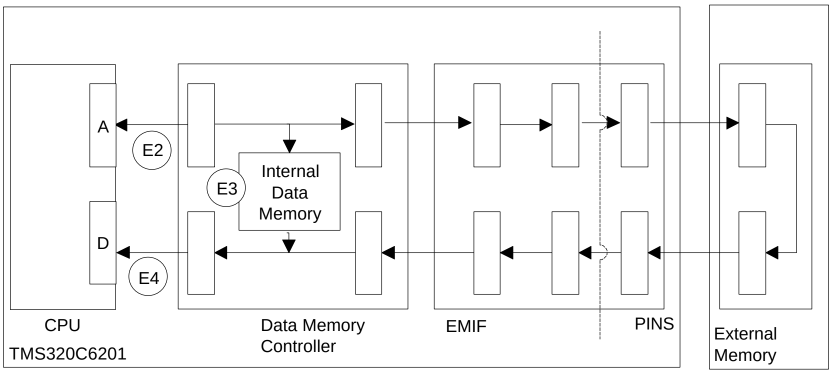

Figure 1: Example of path for a load access.

Accessing external memory, the architecture implies CPU to be stalled for a variable amount of cycles depending on the external memory type. This document describes the number of CPU wait-states required while accessing the external memory from the CPU to data access. Notice that it affects only CPU accesses (that means LD and ST instructions) and not CPU program fetches or DMA accesses from external memory.

Number of CPU wait-states when accessing SBSRAM memory:

The SBSRAM interface can run either at CLKOUT1, or at ½ CLKOUT1.

☐ Single load from SBSRAM:

13 CPU wait-states when SSCLK = CLKOUT1
17 CPU wait-states when SSCLK = ½ CLKOUT1

☐ Single store to SBSRAM:

6 CPU wait-states when SSCLK=CLKOUT1

8 CPU wait-states when SSCLK= ½ CLKOUT1

All CPU data accesses appear as single access externally. There are no additional CPU wait-states when performing two consecutive loads or stores from/to SBSRAMs. Parallel load/store requests are processed consecutively by the DMEMC without any penalty cycles. For example, two parallel loads operation will result in 28 CPU wait-states. For the same reason, load instruction followed by store instruction results in the sum of the wait-states needed for the loads and the wait-states needed for the store.

Number of CPU wait-states when accessing Asynchronous memory:

The number of wait-state when accessing an asynchronous memory depends on the setup time, strobe time, hold time defined for the CE space. The values for the setup, strobe and hold fields are defined to match the asynchronous memory timing requirements. Please refer to [5] for any details about the setting of those fields. The protocol used between CPU, DMEMC and EMIF implies a minimum number of CPU wait-states. To describe the number of wait-states when performing CPU data access to the external asynchronous memory, lets define x, y and z as follow:

```
If ( $setup \le 2$ )     $x = 2$
Else                     $x = setup$

if( $strobe \le 1$ )     $y = 1$
Else                     $y = strobe$

if( $hold \le 1$ )       $z = 1$
Else                     $z = hold$
```

Then, the number of wait-states is described by the following equations:

- Single load from Asynchronous Memory:

  wait-sates for a single load = $x + y + 8$ *cycles*

- Single store to Asynchronous Memory:

```
if ( $(x + y + z) > 10$ )
    wait-states for a single store = $x + y + z - 3$ cycles.
else
    wait-states for a single store = 6 cycles.
```

- Back-to-back stores and store followed by a load:

  Performing an asynchronous access immediately after an asynchronous store (as consecutive stores, parallel stores or store followed by a load), results into some penalty cycles included after the store. The number of wait-states inserted is described by the following equations:

```
if ((x + y + z) >= 10)
additional wait-states = 7 cycles.
else
       if ( y + z >= 6)
          additional wait-states = 6 cycles.
else
  additional wait-states = y + z cycles.
```

- Back-to-back loads and load followed by a store:

There is no penalty cycle inserted after an asynchronous load.

Example 1: How many cycles are required to execute a store, followed by a load, followed by a store? (With setup$_{load}$=0, strobe$_{load}$=0, hold$_{load}$=0, setup$_{store}$=3,strobe $_{store}$=4, hold$_{store}$=2).

STW ⇨ *1 cycle + 6 wait-states + 6 penalty cycles.*
LDW ⇨ *1 cycle + 11 wait-states + 0 penalty cycles.*
STW ⇨ *1 cycle + 6 wait-states.*

Example 2: How many cycles does it take to perform 3 consecutive loads from asynchronous memory with setup$_{load}$=3, strobe$_{load}$=0 and hold$_{load}$=0?

LDW ⇨ *1 cycle + 12 wait-states + 0 penalty cycles.*
LDW ⇨ *1 cycle + 12 wait-states + 0 penalty cycles.*
LDW ⇨ *1 cycle + 12 wait-states.*

Number of CPU wait-states when accessing SDRAM memory:

Because SDRAM is a paged memory type, the External Memory Interface controls SDRAM using multiplexed address information and monitors the active row of SDRAM. During an SDRAM access, the selected bank is activated with the row address by the ACTV command. The row stays opened until the EMIF sends a deactivate command, DCAB, to the SDRAM. This allows suppressing the activate-deactivate overhead when performing burst accesses (i.e. using the DMA controller or fetching an instruction). As all CPU data accesses appears as single accesses, even to complete multiple consecutive accesses, the EMIF will activate the row, and then will access the SDRAM. Performing a single access to an inactive row results in some additional wait-states due to the deactivation of the active row as shown below:

- Load from an active row:

  17-18 CPU wait-states. (Including the ACTV command)

- Load from an inactive row:

  41-42 CPU wait-states. (Including DCAB and ACTV)

- ☐ Store to an active row:

  7-8 CPU wait-states (Including the ACTV command).
- ☐ Store from an inactive row:

  32-33 CPU wait-states. (Including DCAB and ACTV)

There are no additional CPU wait-states when performing two consecutive loads or stores from/to SDRAM. There is no difference when performing two consecutive accesses to different banks or to the same bank. In both cases, a DCAB command needs to be issued by the EMIF.

Number of CPU wait-states when accessing the peripheral bus:

To control on-chip peripherals, the CPU is accessing on-chip control registers through the peripheral bus. CPU accesses through the peripheral bus controller causes 6 wait-states.

Conclusion :

CPU data access to external memory implies a certain amount of wait-states depending on the memory type. These wait-states are due to the number of the registers between the CPU and the external memories. On the other hand, the Program Memory Controller (PMEMC) or through the DMA Controller are accessing the external memory in burst mode, which allows to accessing up to one data per cycles. Depending on the type of memory, the number of wait-states on CPU data accesses may be important therefore it is more efficient to perform all data movements using one of the DMA channel than transferring data using the CPU.

References

[1]. TMS320C6201 Data Sheet, Texas Instruments 1997.
[2]. TMS320C62xx Peripherals, Reference Guide, Texas Instruments 1997.
[3]. TMS62812A –10ns Data Sheet, Texas Instruments 1997.
[4]. Interfacing the TMS320C62xx to External SDRAM, Application Note.
[5]. Interfacing the TMS320C62xx to External Asynchronous SRAM, Application Note.
[6]. Interfacing the TMS320C62xx to External SBSRAM, Application Note.