### 1.9.3 Case study: Using the STM32F Flexible Memory Controller to access SDRAM

The Flexible memory controller (FMC) found in STM32 microcontrollers consists of the following main blocks:

1. the interface to the CPU's Advanced High-performance Bus (AHB) bus,
2. the NOR Flash/SRAM memory controller,
3. the SDRAM memory controller, and
4. NAND Flash controller.

The block diagram of the FMC is shown in Figure 1.32. The AHB interface allows the CPU (and other bus master peripherals) to access the external memories through the FMC controller. Two primary purposes of The Flexible Memory Controller (FMC) are to translate transactions on the high-speed CPU bus (namely AHB bus) into the appropriate external protocol and to meet the access time requirements of the external memory devices.

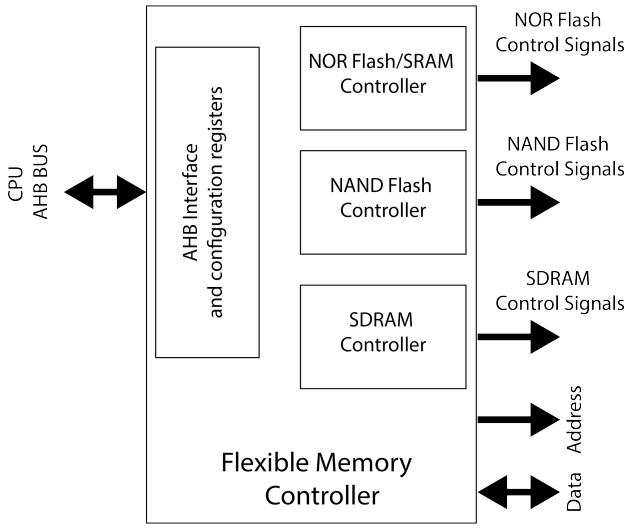


Fig. 1.32: FMC block diagram.

From the FMC (or microprocessor) point of view, the external memory is divided into fixed-size regions of 256 Mbytes each, called banks (Figure 1.33). Bank 1 is used to address NOR Flash memory devices. Bank 3 is used to address NAND Flash memory devices. Banks 4 and 5 are used to address two SDRAM devices (one device per bank). Let us focus only on the FMC SDRAM controller.

All external memories share the addresses, data and control signals with the controller, and each external device is accessed utilizing a unique chip-select signal.

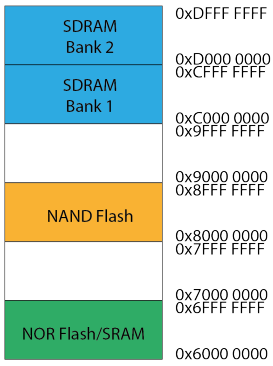


Fig. 1.33: Memory regions accesable from the FMC controller.

The FMC performs only one access at a time to an external device. Here, we will describe only the SDRAM controller and its use to interface a 128 Mbit SDRAM memory chip. All AHB transactions, in this case, translate into the SDRAM device protocol.

The FMC SDRAM controller supports SDRAM devices of up to 256 Mbytes. It can issue a 13-bit row address, an 11-bit column address, and a 2-bit bank address. The memory accesses can be 8-bit, 16-bit, and 32-bit. We will use Micron's 1 Meg x 32 x 4 banks MT48LC4M32B2 SDRAM chip, organized as 4096 rows x 256 columns x 32 bits per bank. Hence, the memory controller would issue a 12-bit row address, an 8-bit column address, and a 2-bit bank address.

The address bit 28 on the AHB bus (internal AHB address line 28) selects one of the two memory devices. For our particular case, where the FMC SDRAM controller is used to access the MT48LC4M32B2 SDRAM chip, the 32-bit memory address from the AHB bus is mapped into the SDRAM address as presented in Figure 1.35.

The SDRAM controller in Figure 1.34 accepts single and burst read and write requests and translates them into single memory accesses. In both cases, the SDRAM controller keeps track of the active row in each bank to be able to perform consecutive read and write accesses. The FMC SDRAM controller comprises a read FIFO (6 lines x 32 bits). It is used to read data in advance - the memory controller anticipates READ commands to the open row if the RBURST bit is set in the FMC_SDCRx register and stores data in the FIFO. Two bits RPIPE[1:0] in the FMC_SDCRx register defines how much data will be anticipated and stored into the FIFO during the read access. If we set both RPIPE[1:0] bits to zero, four data will be anticipated during a single read access. The first read data will be transmitted to the AHB bus, and the other three will be stored in the read FIFO buffer. The read FIFO buffer stores a 14-bit address tag for each line to identify its content: 11 bits for the column address, 2 bits for the internal bank in the active row, and 1 bit for the SDRAM device. Each time a read request occurs, the SDRAM controller checks if the address matches one of the address tags in the read FIFO buffer. In such a case, data are directly read

from the FIFO buffer. Otherwise, a new read command is issued to the SDRAM device, and new data is read to the FIFO buffer.

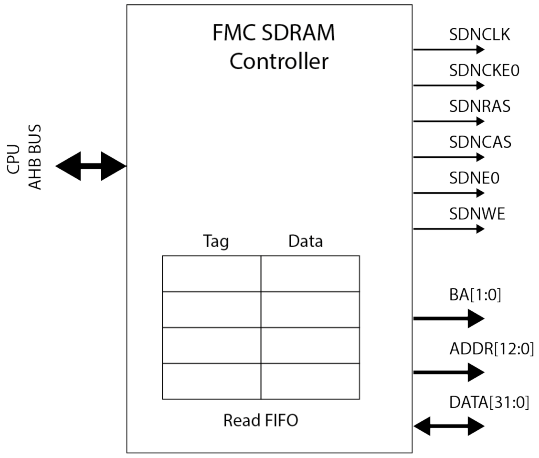


Fig. 1.34: FMC SDRAM Controller block diagram and signals.

The FMC SDRAM controller periodically issues auto-refresh commands to refresh the SDRAM. The programmer should initialize the internal counter value in the FMC_SDRTR. This value defines the number of memory clock cycles between two refresh cycles (refresh rate). When this counter reaches zero, the FMC SDRAM controller issues the auto-refresh command. If there is an ongoing memory access, the auto-refresh request is delayed until the memory access finishes; otherwise, the auto-refresh request takes precedence. If the memory access request occurs during an auto-refresh operation, the request is buffered and processed when the auto-refresh completes. Figure 1.35 illustrates how the 32-bit addresses issued by the CPU on the AHB bus map to the 26-bit addresses issued by the SDRAM controller to the SDRAM device.

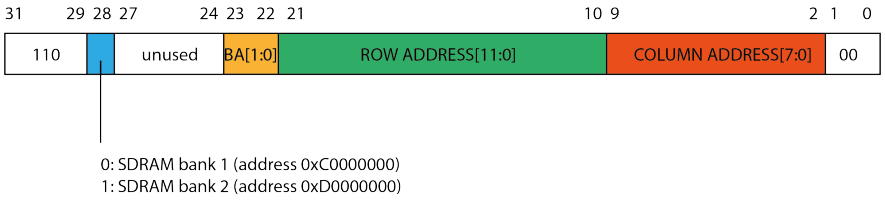


Fig. 1.35: Address maping for a 128-bit SDRAM (4096 rows x 256 columns x 4 banks x 32 bit).

In order to use the FMC SDRAM controller with an external SDRAM device residing in the SDRAM Bank 1, we should:

1. first, initialize the FMC SDRAM controller according to the used SDRAM device, and
2. secondly, initialize the SDRAM device.

The first step involves programming two FMC SDRAM controller configuration registers, SDRAM Control Register 1 (FMC_SDCR1) and SDRAM Timing Register 1 (FMC_SDTR1). The bits in FMC_SDCR1 (Figure 1.37) define the SDRAM clock period, CAS Latency, whether the FMC anticipates READ commands (burst read), data bus width and the internal organization of the SDRAM chip (rows, columns and banks).

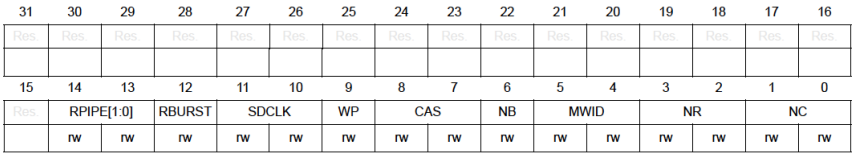


Fig. 1.36: Control register (FMC_SDCR).

The bits in FMC_SDTR1 define SDRAM timing parameters, e.g. RAS-to-CAS delay, row-precharge delay, etc. In order to set the bits in these two registers, we should consult the datasheet for a particular SDRAM chip.

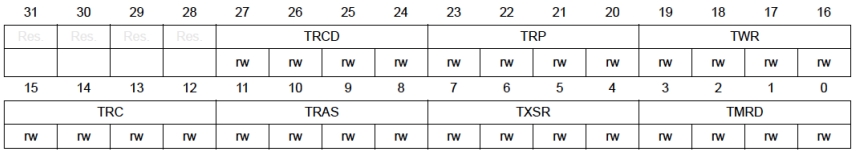


Fig. 1.37: Timing register (FMC_SDTR).

The second step initializes the SDRAM chip. During the SDRAM chip initialization, the FMC controller sends several predefined commands to the SDRAM chip. To send these commands, we should write them into the FMC SDRAM Command Mode Register (FMC_SDCMR). The required initialization steps are described in the datasheet for a particular SDRAM chip and involve the following:

1. providing stable CLOCK signal,
2. performing a PRECHARGE ALL command, which puts all rows in all banks into an idle state,

3. issuing several AUTO REFRESH commands
4. issuing several NOP commands before SDRAM is ready for access.

Instead of directly setting bits in the FMC SDRAM configuration registers, we will rather use the HAL library. The code Listing 1.1 shows the FMC SDRAM controller initialization.

```
uint8_t Init_SDRAM(void)
{
  static uint8_t sdramstatus = SDRAM_ERROR;
  /* SDRAM device configuration */
  sdramHand.Instance = FMC_SDRAM_DEVICE;

  /* Timing configuration for 100Mhz as SDRAM clock frequency
     (System clock is up to 200Mhz) */
  /* These parameters are from the MT48LC4M32B2 Data Sheet,
     Table 18 and Table 19 */
  sdramTiming.LoadToActiveDelay    = 2;   // t_MRD
  sdramTiming.ExitSelfRefreshDelay = 7;   // t_XSR
  sdramTiming.SelfRefreshTime      = 5;   // t_RAS
  sdramTiming.RowCycleDelay        = 7;   // t_RC
  sdramTiming.WriteRecoveryTime    = 2;   // t_WR
  sdramTiming.RPDelay              = 2;   // t_RP
  sdramTiming.RCDDelay             = 2;   // t_RCD


  sdramHand.Init.SDBank             = FMC_SDRAM_BANK1;
  sdramHand.Init.ColumnBitsNumber   = FMC_SDRAM_COLUMN_BITS_NUM_8;
  sdramHand.Init.RowBitsNumber      = FMC_SDRAM_ROW_BITS_NUM_12;
  sdramHand.Init.MemoryDataWidth    = FMC_SDRAM_MEM_BUS_WIDTH_32;
  sdramHand.Init.InternalBankNumber = FMC_SDRAM_INTERN_BANKS_NUM_4;
  sdramHand.Init.CASLatency         = FMC_SDRAM_CAS_LATENCY_3;
  sdramHand.Init.WriteProtection  = FMC_SDRAM_WRITE_PROTECTION_DISABLE;
  sdramHand.Init.SDClockPeriod      = FMC_SDRAM_CLOCK_PERIOD_2;
  sdramHand.Init.ReadBurst          = FMC_SDRAM_RBURST_ENABLE;
  sdramHand.Init.ReadPipeDelay      = FMC_SDRAM_RPIPE_DELAY_0;

  /* SDRAM controller initialization */

  if(HAL_SDRAM_Init(&sdramHand, &sdramTiming) != HAL_OK)
  {
    sdramstatus = SDRAM_ERROR;
  }
  else
  {
    sdramstatus = SDRAM_OK;
  }

  /* Once the FMC SDRAM Ctrl is initialized, we can access
     and initialize the SDRAM chip */
  /* SDRAM initialization sequence */
  SDRAM_Initialization_sequence(REFRESH_COUNT);

  return sdramstatus;
}
```

Listing 1.1: FMC SDRAM Controller initialization.

Firstly, we set the SDRAM timing parameters (in the FMC_SDTR1 register) considering the 100MHz SDRAM clock, and then we set the SDRAM configuration (in the FMC_SDCR1 register).

The code Listing 1.2 shows the FMC SDRAM chip initialization. The SDRAM initialization sequence is described in the SDRAM datasheet in detail. SDRAMs must be powered up and initialized in a predefined manner. Briefly, the initialization procedure contains four steps:

1. Enable the stable SDRAM clock.
2. Wait for at least 100us prior to issuing any command.
3. Perform a PRECHARGE ALL command.
4. Issue at least two AUTO REFRESH commands.
5. The SDRAM is now ready for mode register programming. Because the mode register will power up in an unknown state, it should be loaded with desired bit values prior to applying any operational command.

```
/**
  * @brief  Init the SDRAM device.
  * SDRAMs must be initialized in a predefined manner. Operational ↩
    procedures
  * other than those specified in the SDRAM Data Sheet may result in ↩
    undefined operation.
  * @param  RefreshCount: SDRAM refresh counter value
  * @retval None
  */
void SDRAM_Initialization_sequence(uint32_t RefreshCount)
{
  __IO uint32_t tmpmrd = 0;

  /* Step 1: Configure a clock configuration enable command */
  sdramCmd.CommandMode            = FMC_SDRAM_CMD_CLK_ENABLE;
  sdramCmd.CommandTarget          = FMC_SDRAM_CMD_TARGET_BANK1;
  sdramCmd.AutoRefreshNumber      = 1;
  sdramCmd.ModeRegisterDefinition = 0;


  /* Send the Clock Configuration Enable command to the target bank*/
  /* The command is sent as soon as the Command MODE field in the
     CMR is written */
  HAL_SDRAM_SendCommand(&sdramHand, &sdramCmd, SDRAM_TIMEOUT);

  /*
   * Once the clock is stable, the SDRAM requires a 100us delay
   * prior to issuing any command
   */

  /* Step 2: Insert 100 us minimum delay */
  /* Inserted delay is equal to 1 ms due to systick time base unit */
  HAL_Delay(1);


  /*
   * Once the 100us delay has been satisfied, a PRECHARGE command
   * should be applied. All banks must then be precharged,
   * thereby placing the device in the all banks idle state.
   */
  /* Step 3: Configure a PALL (precharge all) command */
  sdramCmd.CommandMode            = FMC_SDRAM_CMD_PALL;
  sdramCmd.CommandTarget          = FMC_SDRAM_CMD_TARGET_BANK1;
  sdramCmd.AutoRefreshNumber      = 1;
  sdramCmd.ModeRegisterDefinition = 0;

  /* Send the Precharge All command to the target bank */
```

```
46   /* The command is sent as soon as the Command MODE field
        in the CMR is written */
48   HAL_SDRAM_SendCommand(&sdramHand, &sdramCmd, SDRAM_TIMEOUT);

50   /*
      * Once in the idle state, at least two AUTO REFRESH cycles must
52    * be performed. If desired, more than two AUTO REFRESH
      * commands can be issued in the sequence.
54    */
     /* Step 4: Configure an Auto Refresh command */
56   sdramCmd.CommandMode            = FMC_SDRAM_CMD_AUTOREFRESH_MODE;
     sdramCmd.CommandTarget          = FMC_SDRAM_CMD_TARGET_BANK1;
58   sdramCmd.AutoRefreshNumber      = 8;
     sdramCmd.ModeRegisterDefinition = 0;
60
     /* Send the Auto-refresh commands to the target bank */
62   /* The command is sent as soon as the Command MODE
        field in the CMR is written */
64   HAL_SDRAM_SendCommand(&sdramHand, &sdramCmd, SDRAM_TIMEOUT);


66
     /*
68    * The SDRAM is now ready for mode register programming.
      * Because the mode register will power up in an unknown state,
70    * it should be loaded with desired bit values prior to
      * applying any operational command. Using the LMR command,
72    * program the mode register.
      */
74   /* Step 5: Program the external memory mode register */
     tmpmrd = (uint32_t)SDRAM_MODEREG_BURST_LENGTH_1          |\
76                      SDRAM_MODEREG_BURST_TYPE_SEQUENTIAL   |\
                        SDRAM_MODEREG_CAS_LATENCY_3           |\
78                      SDRAM_MODEREG_OPERATING_MODE_STANDARD |\
                        SDRAM_MODEREG_WRITEBURST_MODE_SINGLE;
80
     sdramCmd.CommandMode            = FMC_SDRAM_CMD_LOAD_MODE;
82   sdramCmd.CommandTarget          = FMC_SDRAM_CMD_TARGET_BANK1;
     sdramCmd.AutoRefreshNumber      = 1;
84   sdramCmd.ModeRegisterDefinition = tmpmrd;

86   /* Send the Load Mode Register command to the target bank */
     /* The command is sent as soon as the Command MODE field in
88      the CMR is written */
     HAL_SDRAM_SendCommand(&sdramHand, &sdramCmd, SDRAM_TIMEOUT);
90
     /*
92    * Wait for at least tMRD time. This is automatically performed by
      * the FMC SDRAM controller. At this point the DRAM is ready for
94    * any valid command.
      */
96
     /* Step 6: Set the refresh rate counter in Refresh Timer register */
98   /* This 13-bit field defines the refresh rate of the SDRAM device.
        It is expressed in number of memory clock cycles. */
100  HAL_SDRAM_ProgramRefreshRate(&sdramHand, RefreshCount);
}
```

Listing 1.2: SDRAM initialization sequence.

To enable the above procedure, the FMC SDRAM controller provides a special register called Command Mode register (FMC_SDCMR), illustrated in Figure 1.38. It contains four fields. The MODE field defines the command issued to the SDRAM chip. The possible commands are, for example, "CLK ENABLE", "PRECHARGE ALL", "AUTO REFRESH", and "LOAD MODE REGISTER". The

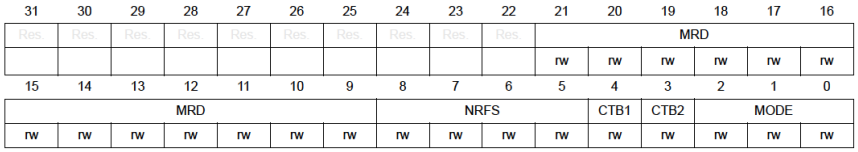


Fig. 1.38: Command Mode register (FMC_SDCMR).

CTB1 and CTB2 fields select the SDRAM chip to which the command is sent. As soon as the MODE field is written, the FMC SDRAM controller will issue the corresponding command to SDRAM chips selected by CTB1 and CTB2 command bits. The NRFS field defines how many consecutive Auto-refresh commands are issued in the fourth step of the initialization sequence, the MRD field contains the content that should be written to the SDRAM Mode Register. The mode register is a 12-bit special register inside the SDRAM chip and is used to define the specific mode of operation of the SDRAM. This definition includes the selection of a burst length (BL), a burst type, a CAS latency (CL), an operating mode and a write burst mode, as shown in Figure 1.39. The mode register is programmed from the FMC SDRAM controller via the "LOAD MODE REGISTER" command and retains the stored information until it is programmed again or the SDRAM device loses power.

At the end of the SDRAM chip initialization, we set the auto-refresh period in the FMC SDRAM controller. The AUTO REFRESH command is used during the regular operation of the SDRAM to refresh its content. This command is nonpersistent, so it must be issued each time a refresh is required. If memory access is in progress, the auto-refresh request is delayed. The refresh controller inside the SDRAM chip generates the address of the row that should be refreshed. For example, the 128Mb SDRAM requires 4096 AUTO REFRESH commands every 64ms. To ensure that each row is refreshed according to this requirement, the SDRAM controller must issue an AUTO REFRESH command every 15.625us. The FMC SDRAM controller provides the Refresh Timer register (FMC_SDRTR). This register holds the 13-bit refresh rate in number of SDRAM clock cycles. This 13-bit field should be set immediately after the initialization of SDRAM. The 13-bit refresh rate is calculated as follows. As the SDRAM clock runs at 100 Mhz (10 ns period), 15.625 us equals 1562 SDRAM clock periods. We should subtract at least 20 SDRAM clock periods from this value to obtain a safe margin if an auto-refresh request occurs when a read request has been accepted. Hence, the 13-bit refresh rate in the FMC_SDRTR register corresponds to 1542.

To demonstrate the different scenarios when using the FMC SDRAM controller, we copy a matrix of size 32 rows times 256 columns from the internal SRAM to the external SDRAM and then read it back. The elements of the matrix are 32-bit unsigned integers. In the first scenario (Listing 1.3), the matrix is accessed in row-major order, while in the second scenario (Listing 1.4), the matrix is accessed

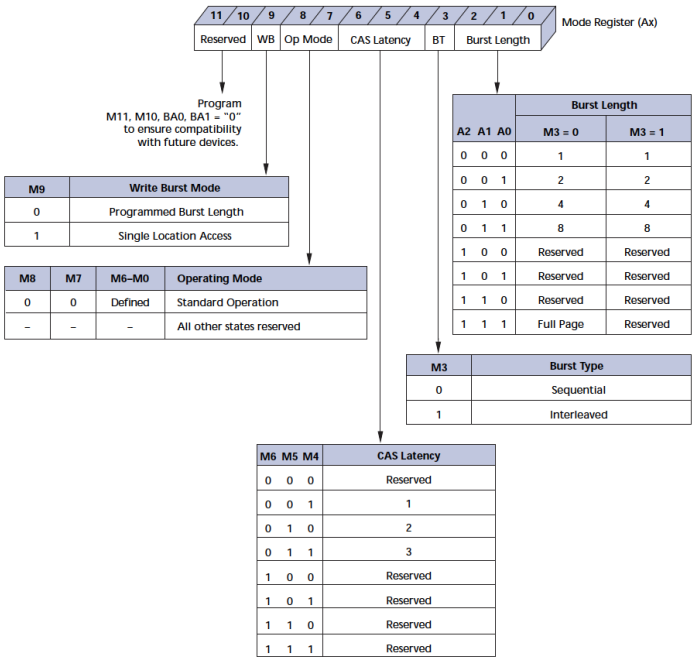


Fig. 1.39: SDRAM Mode Register.

in column-major order. The constants `SDRAM_DEVICE_ADDR` and `SDRAM_COLS` in Listings 1.3 and 1.4 equal 0xC0000000 and 256, respectively.

```
void SDRAM_mat_row_access_test(void){
  volatile uint32_t address;

  for (int i = 0; i<MAT_ROWS; i++) {
    for(int j=0; j<SDRAM_COLS; j++) {
      address = SDRAM_DEVICE_ADDR + ((i*SDRAM_COLS + j)<<2);
      matrixB[i][j] = *(uint32_t*)address;
    }
  }
}
```

Listing 1.3: Read matrix from SDRAM in row-major order.

```
void SDRAM_mat_col_access_test(void){
  volatile uint32_t address;

    for (int i = 0; i<SDRAM_COLS; i++) {
    for(int j=0; j<MAT_ROWS; j++) {
      address = SDRAM_DEVICE_ADDR + ((j*SDRAM_COLS + i)<<2);
      matrixB[j][i] = *(uint32_t*)address;
    }
  }
```

```
10 }
```

Listing 1.4: Read matrix from SDRAM in column-major order.

Figure 1.40 illustrates one read issued from the CPU for the first scenario (row-major order access). The FMC SDRAM controller does not support SDRAM burst reads or writes (the only allowable burst length is 1). Instead, it supports burst reads on the CPUs AHB bus by utilizing the internal FIFO. Hence, it anticipates four READ commands to fill in the internal FIFO. The FIFO content is then transferred to the CPU using the AHB burst read of length 4.

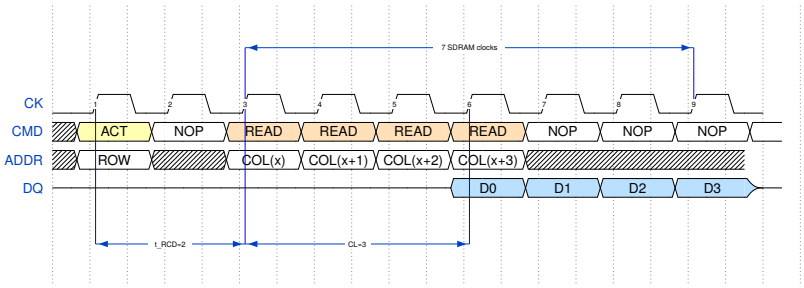


Fig. 1.40: Using row-major order to read a matrix, the SDRAM controller anticipates four consecutive READ command to the active SDRAM row for each read initiated from the CPU

In the second scenario, the matrix is accessed using column-major order. Figure 1.41 illustrates two consecutive reads issued from the CPU. As the CPU reads data from consecutive rows in each iteration, the CPU controller first reads four consecutive words from the active SRAM row and fills the internal FIFO, but it only returns one word to the CPU over the AHB bus. As the CPU starts another read from the next row, the SDRAM controller first precharges the active row. It then waits for two SDRAM clock periods (Row Precharge time) before activating the next row.

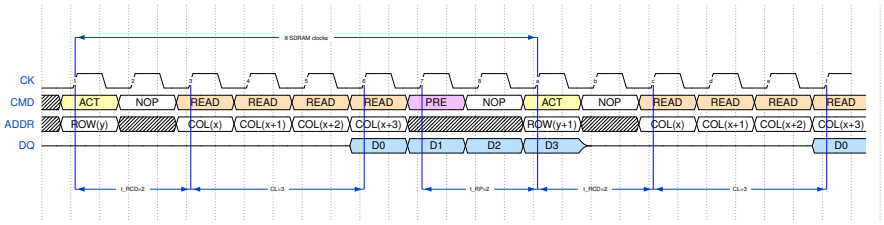


Fig. 1.41: Using column major order results in activating, reading and precharging an SDRAM row for every read issued from the CPU.

It is obvious that row-major order access is considerably faster than column-major order access. A rough estimate of the access time for row-major order access considering an already open row is seven (7) SDRAM clock periods per four words. On the other side, a rough estimate of the access time for column-major order access is eight (8) SDRAM clock periods per word. Recall that only one word is transferred to the CPU, although the SDRAM controller anticipates four consecutive reads from the active row.

To assess the performance (speed) of the row-major and column-major matrix reads, we use the code in Listing 1.5. For each test, the code first sets the PC8 pin and reads the timer TIM3 counter value (this is the start of the test). After the test, we reset the PC8 pin and read the timer TIM3 counter value (this is the start of the test). By setting and resetting the PC8 pin, we can measure the duration of each test using an oscilloscope. The timer TIM3 runs at 1MHz (1 us resolution). Hence, we can estimate the duration of each test simply by reading the timer counter before and after the test.

```
// Row-major order access:
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_SET);
timer_val_start = __HAL_TIM_GET_COUNTER(&TIM3Handle);
SDRAM_mat_row_access_test();
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_RESET);
timer_val_end = __HAL_TIM_GET_COUNTER(&TIM3Handle);
if (timer_val_end > timer_val_start)
  elapsed_rows = timer_val_end - timer_val_start;
else
  elapsed_rows = timer_val_end + (65536-timer_val_start);

// Column-major order access:
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_SET);
timer_val_start = __HAL_TIM_GET_COUNTER(&TIM3Handle);
SDRAM_mat_col_access_test();
timer_val_end = __HAL_TIM_GET_COUNTER(&TIM3Handle);
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_RESET);
if (timer_val_end > timer_val_start)
  elapsed_cols = timer_val_end - timer_val_start;
else
  elapsed_cols = timer_val_end + (65536-timer_val_start);
```

Listing 1.5: Code used to test the speed of row-major and column-major matrix read from the SDRAM.

Figure 1.42 shows the oscilloscope trace for the signal on the GPIOC pin. It shows that the row-major order read lasts for about 2.3 ms, while the column-major order read lasts for about 10 ms. Using the timer counter, we estimate the duration of the row-major order read to 2365 us and the duration of the column-major order read to 9816 us. Both measurements show that the row-major order read is about four times faster than the column-major order read, which is in accordance with the rough estimation from figures 1.40 and 1.41.

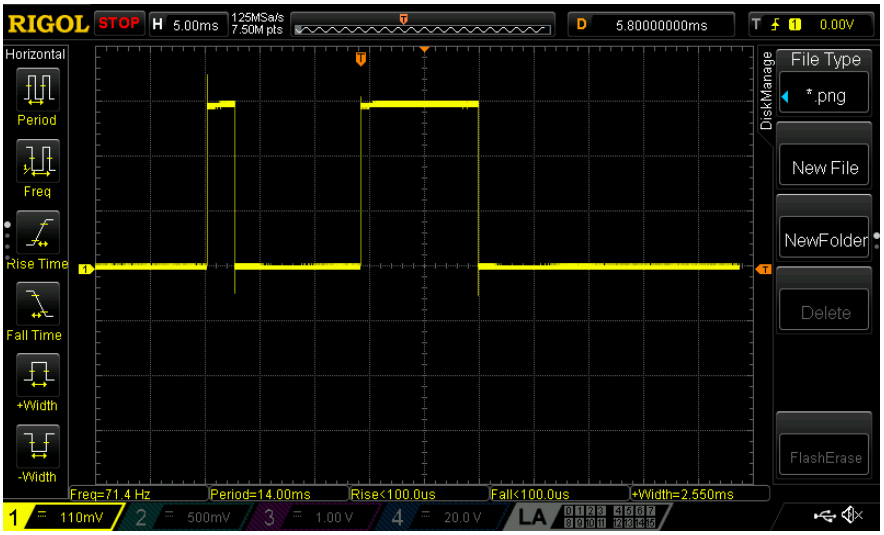


Fig. 1.42: Oscilloscope trace on the GPIOC pin 8. The row-major order matrix read lasts for about 2.5 ms while the column-major order matrix read lasts for more than 10 ms.