The following document contains information on Cypress products. The document has the series name, product name, and ordering part numbering with the prefix “MB”. However, Cypress will offer these products to new and existing customers with the series name, product name, and ordering part number with the prefix “CY”.

How to Check the Ordering Part Number

1. Go to www.cypress.com/pcn.
2. Enter the keyword (for example, ordering part number) in the **SEARCH PCNS** field and click **Apply**.
3. Click the corresponding title from the search results.
4. Download the Affected Parts List file, which has details of all changes

For More Information

Please contact your local sales office for additional information about Cypress products and solutions.

About Cypress

Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, smart home appliances, consumer electronics and medical products. Cypress' microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first. Cypress is committed to providing customers with the best support and development resources on the planet enabling them to disrupt markets by creating new product categories in record time. To learn more, go to www.cypress.com.



THIS SPEC IS OBSOLETE

Spec No: 002-05207

Spec Title: AN205207 - FR FAMILY MB91265 SERIES 16-BIT MAC UNIT

Replaced by: None

AN205207

FR Family MB91265 Series 16-Bit MAC Unit

This application note will explain the configuration and usage of the 16-bit MAC unit, and give some short application examples as well as some programming helps.

Contents

1 Introduction 1
2 Features 1
2.1 Features 1
2.2 Block diagram 2
2.3 MAC unit registers description 3
2.4 Instructions of the MAC unit 4
3 Operation of the MAC unit 6
4 Usage of the MAC unit 7
4.1 FIR filter introduction 7
4.2 Initialization 9
4.3 Calculation performance of the MAC unit 10
4.4 Using the MAC unit for PID control loops 13
A Appendix 15
A.1 Register list and memory map of the MAC unit 15
A.2 Related Documents 15
A.3 Glossary 16
5 Document History 17

1 Introduction

The Cypress MB91265 Series is a Flash microcontroller especially for motor control and other control applications. It features a Multi-Function Timer incl. Waveform generator to generate the output voltages needed for three-phase motors, and other dedicated peripherals like the 16-bit MAC (*Multiply-and-ACcumulate*) unit for fast processing.

The MAC unit has an own instruction and data RAM and works completely independently of the CPU as soon as the calculation is started. It can execute 16 bits * 16 bits + 40 bits calculation in one machine cycle (31.25ns), and shift values arbitrarily in Y-RAM. Additional jump (even as conditional branch) and store commands, which can also issue an interrupt request upon completion, make the MB91265 Series MAC unit ideal for digital filtering of any kind, like FIR (*Finite Impulse Response*) or IIR (*Infinite Impulse Response* or *recursive*) filters. Once the filter is set up in the MAC unit, the CPU only has to update new values, start the MAC unit and can work on other tasks while waiting for the result. Therefore, it is easily possible e.g. to process a signal sampled by the A/D converter with a 64-tap filter at high sampling rates and low CPU usage. Some examples will be shown later in this document.

This application note will explain the configuration and usage of the 16-bit MAC unit, and give some short application examples as well as some programming helps.

2 Features

This section describes the features, the register set and the block diagram of the 16-bit MAC.

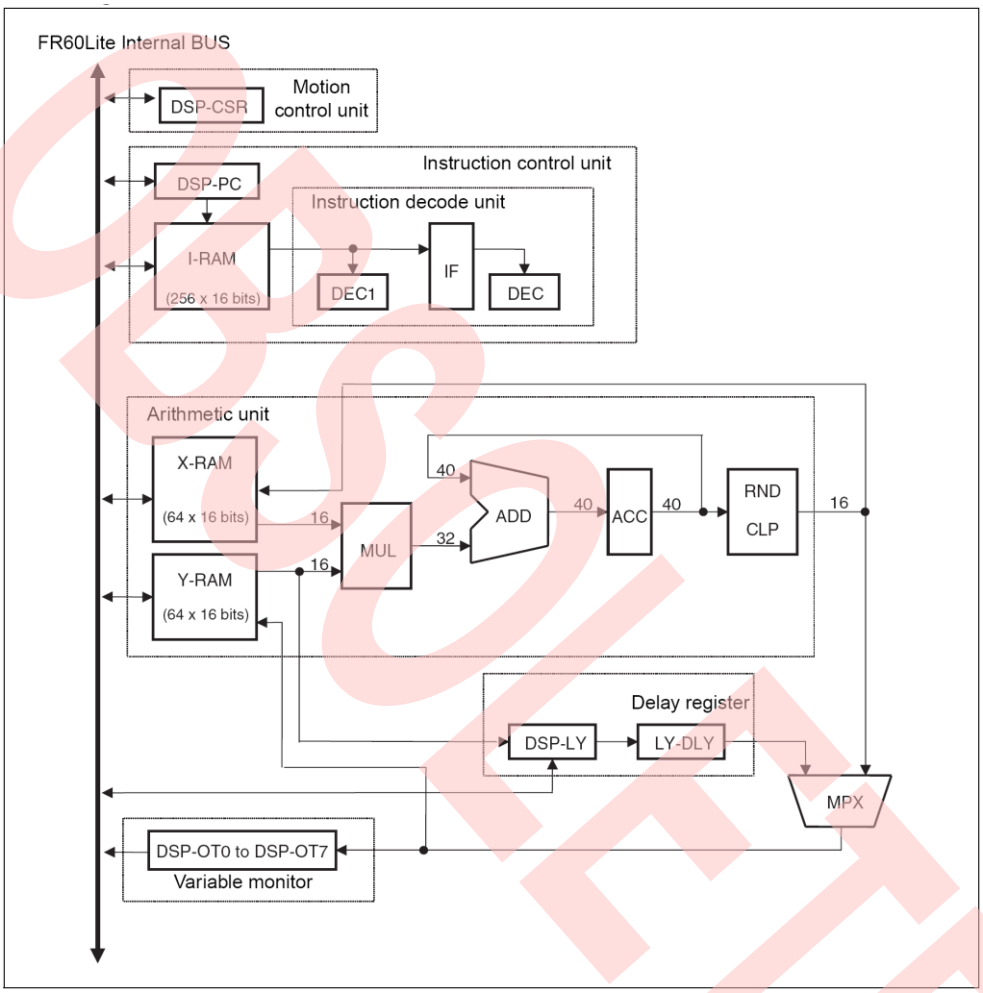
2.1 Features

- High-speed multiply-accumulate (in one system clock cycle)
- Own program flow, independent of CPU
- Data format: 16-bit fixed-point (16 * 16 + 40 bits), Q8-Q15 format selectable for output
- Instruction RAM (IRAM): 256 * 16 bits
- Data RAM: 64 * 16 bits * 2 pairs (X-RAM: Coefficients, Y-RAM: Variables)
- Rounding and saturation of output value available
- Instruction set: MAC instruction, STR instruction, JMP instruction (also conditional)

OBSOLETE

- Delay handling: Variables can be shifted arbitrarily in Y-RAM
- Variable monitor: Up to eight Y-RAM values can be monitored during operation without stopping the uDSP program

2.2 Block diagram



OBSOLETE

2.3 MAC unit registers description

2.3.1 DSP Control/Status Register (DSP-CSR)

This 8-bit register controls the operation state of the MAC unit and also holds the status flags (IRQ, SatDSP and RunDSP).

Bit 7: SatDSP: This flag is set when saturation occurs while clipping is enabled in the STR command (CLP=1). This bit is automatically cleared at the beginning of operation.

Bits 6…4: USR2-0: These bits can be used to control the program flow of the MAC unit. When the JMP command is used with COND=1, the condition coded in the JMP command is compared with these bits, and the jump is only performed on a match. Since the DSP-CSR register can be written and read also during MAC operation, this can be used together with the variable monitor function to react on intermediate results, e.g. for loop programming.

Bit 3: IrqDSP: This is the interrupt request flag for the MAC unit. If the MAC IRQ is enabled (IeDSP=1), this bit signals an IRQ to the CPU if the SIRQ bit in a STR or JMP instruction is set.

Bit 2: IeDSP: Interrupt request enable bit for the MAC unit

Bit 1: HltDSP: This write-only bit can be used to stop an ongoing calculation, e.g. for access to the MAC units memory. The RunDSP bit is thereby cleared, and the PC points at the next instruction, so operation can be resumed by setting the GoDSP bit.

Bit 0: GoDSP (write) / RunDSP (Read): Setting this bit starts the MAC unit program execution, if the MAC unit is not already running. Reading the RunDSP bit returns ‘1’ as long as the calculations are ongoing, and ‘0’ when the operation is halted (by HLT=1 in a STR or JMP command or by writing 1 to the HltDSP bit).

2.3.2 DSP Program Counter (DSP-PC)

The program counter is 8-bit long. It points to the memory address (in I-RAM) holding the next instruction to be executed by the MAC unit. As in other controllers, the program counter is automatically incremented after a command was executed, and can be overwritten by a JMP command. Note that the PC counts 16-bit words, so that it is incremented by 1 after every execution, so also odd numbers occur.

The program counter initially has to point to the first instruction before the MAC unit is started by setting the GoDSP bit.

2.3.3 DSP Delay Register (DSP-LY)

If the LDLY bit in a MAC instruction is set, first the content of the DSP-LY register is transferred to the LY-DLY register. Then, the Y-RAM data of the actual MAC instruction is copied to the DSP-LY register. If the STLY bit of the MAC instruction is set, the content of the LY-DLY register is written to the Y-RAM address of the actual instruction after its execution. Note that this write operation takes another clock cycle. All together, this mechanism allows very effective programming of digital filters, since the input data can be automatically shifted through the filters memory by the MAC unit. In this case, every filter stage only needs two clock cycles for processing.

Also refer to the MAC command description later in this document for details.

The DSP-LY register can only be accessed by the CPU if the MAC unit is halted.

2.3.4 DSP Variable Monitor Register (DSP-OT0 to DSP-OT7)

The content of the first eight Y-RAM words are permanently mirrored to the DSP-OT0 to DSP-OT7 registers. Since the MAC units memory is not accessible for the CPU during MAC operation, these registers can be used to monitor Y-RAM data by the CPU, e.g. for intermediate results or conditional branches. These registers are read-only and their read value is indeterminate after a reset.

2.3.5 X-RAM

The X-RAM (64*16 bits) stores the first factor (i.e. the coefficients) for the MAC operations.

This RAM area cannot be accessed by the CPU while the MAC unit is operating (RunDSP = 1). Initialize used X-RAM areas before starting the MAC operations.

Obsolete

2.3.6 Y-RAM

The Y-RAM (64*16 bits) stores the second factor (i.e. the variables) for the MAC operations.

This RAM area cannot be accessed by the CPU while the MAC unit is operating (RunDSP = 1). Initialize used Y-RAM areas before starting the MAC operations.

2.3.7 I-RAM

The I-RAM (256*16 bits) is the instruction (program) RAM of the MAC unit. The instructions stored in this area are sequentially executed while the DSP-PC is incremented, and also a JMP command exists. This RAM area cannot be accessed by the CPU while the MAC unit is operating (RunDSP = 1). Transfer the MAC program to I-RAM before starting the MAC operations.

2.4 Instructions of the MAC unit

The MAC unit implements three main types of commands:

MAC: Multiply-and-Accumulate;

STR: Store

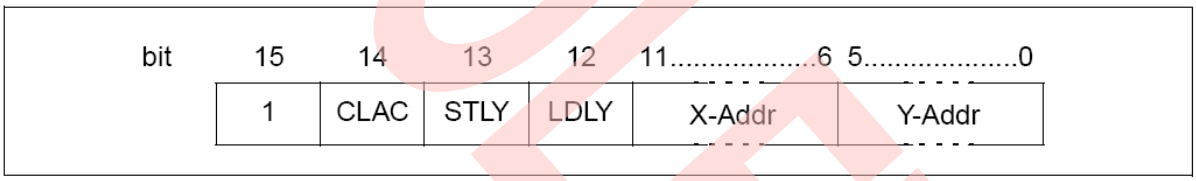
JMP: Jump

Each of these commands has several option bits, which are described in the following. The instructions are stored in the I-RAM of the MAC unit, from where they are executed independently of the CPU. Usage examples are given later in this document.

2.4.1 MAC Instruction

The MAC instruction is the core functionality of the MAC unit. Depending on the setting of the CLAC bit, either a MAC (Multiply-and-Accumulate) or a MUL (Multiply) instruction is executed.

Operation code:



[bit14] **CLAC** (Clear ACC)

Setting this bit causes the instruction to act as a multiplication instruction.

"0": ACC = ACC + data@X-Addr x data@Y-Addr (multiply-and-accumulate instruction)

"1": ACC = 0 + data@X-Addr x data@Y-Addr (multiplication instruction)

[bit13] **STLY** (STore LY)

When this bit is set, the value in the LY-DLY register is written to the Y-RAM address defined by Y-Addr after the instruction was executed. The execution time increases by one cycle if this bit is set.

[bit12] **LDLY** (LOad LY)

When this bit is set, first, the actual content of the DSP-LY register is moved to the LY-DLY register. Then, the content of the Y-RAM defined by Y-Addr is loaded to the DSP-LY register. These transfers have no influence on execution time of the MAC command.

[bit11 to bit6] **X-Addr** (X-RAM Address)

These bits specify the X-RAM address for the actual MAC command.

[bit5 to bit0] **Y-Addr** (Y-RAM Address)

These bits specify the Y-RAM address for the actual MAC command.

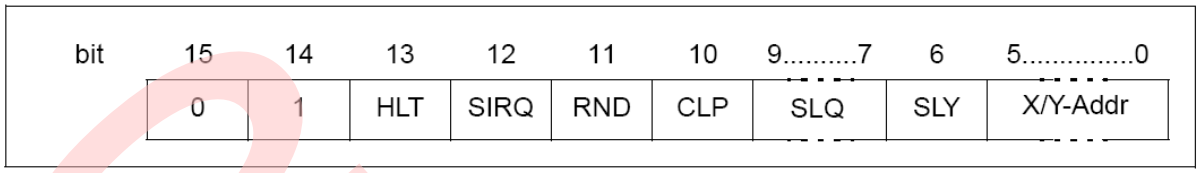
Note: X-RAM, Y-RAM and I-RAM addresses are word addresses relative to the beginning of the corresponding memory area; e.g. X-RAM address 0 = 0x00C000, X-RAM address 1 = 0x00C002, Y-RAM address 4 = 0x00C088 etc.

OBSOLETE

2.4.2 STR Instruction (Transfer Instruction)

The Store command converts the 40-bit accumulator value to a 16-bit value in accordance with the RND, CLP, and SLQ flags, and stores the result in the data RAM specified by the SLY flag and X/Y-Addr.

Operation code:



[bit13] **HLT** (HLT instruction flag)

Setting this bit causes the MAC unit to halt program execution after the store instruction was executed. This also clears the RunDSP flag in the DSP-CSR register and enables access of the CPU to the MAC unit RAM areas.

[bit12] **SIRQ** (INT instruction flag)

If this bit is set, the IrqDSP flag in the DSP-CSR register is set after the store command was executed, and an interrupt request is issued to the CPU.

[bit11] **RND** (Rounding)

This bit specifies whether to perform rounding for 16-bit data specified by the SLQ bits. Rounding rounds the 16-bit data based on the bit immediately below the LSB (add 1 if lower bit is 1, no action if 0).

[bit10] **CLP** (Clipping)

If this bit is set, the result value is either 0x7FFF (positive) or 0x8000 (negative) if the result is not correctly represented by the current output format selection due to an overflow of the 16-bit value. This is done by comparing the MSB (bit39) of the accumulator with the MSB of the selected output word selected by the SLQ bits. If rounding is enabled, the result after rounding is used.

[bit9 to bit7] **SLQ**

These bits specify the output format of the 40-to-16 bit truncation, and thereby the position of the decimal point. Q8 to Q15 formats can be selected. For many filtering applications, the output format will be defined by the sum of filter coefficients. For example, for an 8-tap moving average filter, X-RAM 0…7 could be set to 32. To achieve unity DC gain, Q8 output format has to be chosen in this case (8*32 = 256 -> result = ACC / 256 -> use ACC bits 23 to 8.

[bit6] **SLY**

This bit specifies the transfer destination RAM area:

0: X-RAM

1: Y-RAM

[bit5 to bit0] **X/Y Addr** (RAM Address)

These bits specify the address in X- or Y-RAM (0…63) to store the 16-bit calculation result. From there, the CPU can read the result after operation is halted. If one of the first eight Y-RAM addresses are used for the result, the CPU can also read the value using the variable monitor registers, and the calculation can continue directly after the store instruction (if HLT=0).

OBSOLETE

2.4.3 JMP Instruction (Branch Instruction)

The JMP instruction loads the address specified by the J-Addr8 bits to the MAC unit program counter (DSP-PC). If the COND bit is set, the branch is only performed if UBP2…0 bits match the USR2…0 bits in the DSP-CSR register.

Operation code:

bit	15	14	13	12	11	10.....8	7.....0
	0	0	HLT	SIRQ	COND	UBP2 to UBP0	J-Addr8

[bit13] **HLT** (HLT instruction flag)

Setting this bit causes the MAC unit to halt program execution after the branch instruction was executed. This also clears the RunDSP flag in the DSP-CSR register and enables access of the CPU to the MAC unit RAM areas.

[bit12] **SIRQ** (INT instruction flag)

If this bit is set, the IrqDSP flag in the DSP-CSR register is set after the branch command was executed, and an interrupt request is issued to the CPU.

[bit11] **COND** (CONDition)

0: Always branch (Jump)

1: Branch to J-Addr8 if [UBP2…0] = [DSP_CSR_USR2…0]

[bit10 to bit8] **UBP2 to UBP0** (condition specification)

These bits set the condition to use for the conditional branch. The condition is established if these bits match the value of the USR2, USR1, and USR0 bits in the DSP-CSR register. These bits must be set to "000" if an unconditional branch is specified. (when COND=0)

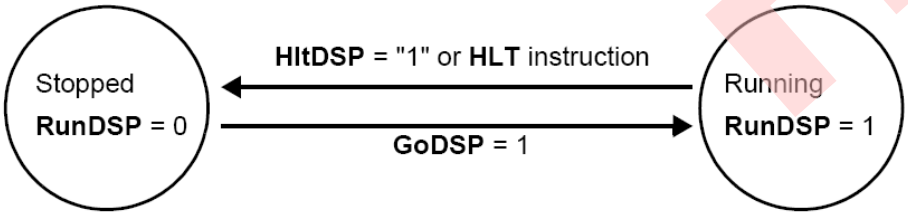
[bit7 to bit0] **J-Addr8** (Jump Address)

These bits specify the branch target address in I-RAM. This value is written to the DSP-PC when the JMP is executed.

3 Operation of the MAC unit

This section explains the operation of the MAC unit.

The operation of the MAC unit is mainly controlled by the DSP-CSR register. It starts program execution when '1' is written to the GoDSP bit, and stops when a HLT command is executed or '1' is written to the HltDSP bit. Note that some of the registers and memory areas of the MAC unit are only accessible for the CPU when the MAC unit is stopped (RunDSP = 0).



- Stopped: The multiply-and-accumulate macro is stopped. The CPU can access instruction RAM (IRAM), data RAM (X-RAM, Y-RAM), and all registers of the MAC unit.

OBSOLETE

- Running: The multiply-and-accumulate unit is operating. When ‘1’ is written to the GoDSP bit while the MAC unit is stopped, it enters this state and starts program execution from the current DSP-PC (program counter) address. While in this state, not all registers and memory areas of the MAC unit can be accessed. Trying to read from a non-accessible area will return indeterminate results, while writing to it has no effect. Writing ‘1’ to the HltDSP bit or a Store or Jump command with set HLT bit cause the MAC unit to halt and return to the ‘stopped’ state.

The general initialization of the MAC unit consists of the following steps:

1. Stop the MAC unit (write ‘1’ to the HltDSP bit)
2. Transfer coefficients and data to X- and Y-RAM according to application
3. Transfer MAC unit instructions to I-RAM
4. Set the DSP-PC to the first instruction to be executed
5. Start the MAC unit (write ‘1’ to the GoDSP bit)

Steps 4 and 5 can be combined to a single 16-bit access.

The MAC unit starts calculation and continues program execution until one of the following conditions occurs:

- ‘1’ is written to the HltDSP bit
- A JMP command with HLT = 1 is executed
- A STR command with HLT = 1 is executed
- A reset occurs

Note that the 40-bit accumulator of the MAC unit is not initialized automatically. Therefore, the first MAC command of the program should clear the accumulator (CLAC=1), except when this behavior is explicitly desired.

Since the MAC unit operates independently of the rest of the MCU, the CPU can process other tasks during calculation. Before accessing the MAC result, either bit0 (RunDSP) of the DSP-CSR register can be polled to check for MAC program end, or the MAC interrupt can be used. Both methods have their advantages; the polling loop will have less latency between the MAC calculation end and the usage of the result, while the ISR might be better suited if other tasks have to be scheduled. It also is often possible to give the CPU another task which has at least the same execution time as the MAC program (e.g. UART communication), and when the CPU completes this task, the MAC unit also has already completed calculation and the result can be used immediately.

4 Usage of the MAC unit

This chapter explains how to setup and use the MAC unit with some examples

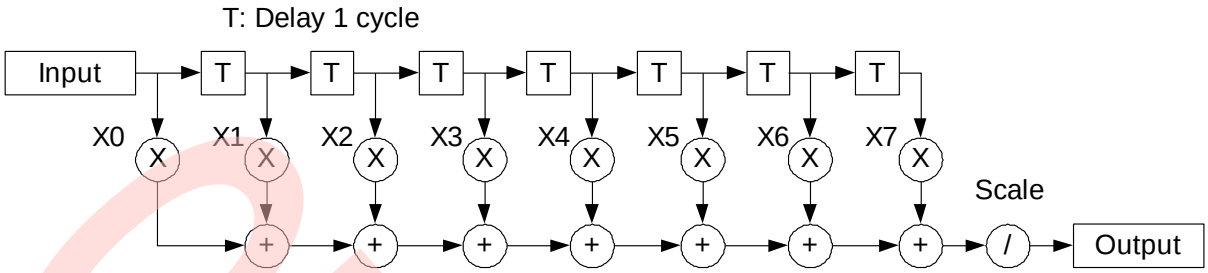
4.1 FIR filter introduction

Multiplication-and-accumulate operations are typical for digital filters. Therefore, the functionality of the MAC unit enables high-speed filtering and other processing typical for DSP applications. Since the MAC unit operates completely independently of the CPU, it can process data separately and thereby reduce CPU load.

The typical structure of a digital FIR filter is shown below:

OBSOLETE

Figure 1. FIR filter



For every sampling point in time, the filter's output value is the sum of the *n* last input values; each one weighted with a coefficient X0…X7.

For example, in case of a moving average filter with equally weighted coefficients, the values X0…X7 are identical. The moving average filter is often used as simple low-pass filter, e.g. to reduce noise in analog measurements. Another common low-pass filter is the binomial filter, whose coefficients can be calculated with Pascal's triangle (e.g. 1, 7, 21, 35, 35, 21, 7, 1). The gain at 0 Hz (DC gain) is given by the sum of all coefficients, and since this is mostly set to unity for low-pass filters, the scaling stage divides the output by this value. For digital systems, usually a power of two is selected, so that the division can be performed by a simple bit shift. Every 'T' block in the above filter holds the delayed input signal from the last clock cycle, so the input values are 'daisy-chained' through the filter. The pulse response of a FIR filter of length N is max. N+1 samples long. The resulting output of the filter to an M-sample input signal therefore is M+N+1 samples long.

The MAC unit of the MB91265 Series can perform the hand-over of the input value from one filter stage to the next without CPU interaction. This is done by the DSP-LY and LY-DLY registers, when the LDLY and STLY bits in the MAC commands are set accordingly. Therefore, the complete processing of every input value, consisting of *(n+1)* multiplications, *n* additions, *n* register transfers and one division (or bit shift) is handled solely by the MAC unit. The CPU transfers the new input value to the MAC unit and starts it. 2n cycles later the result is ready-to-use; meanwhile other tasks can be performed.

The MAC unit also automatically performs the output scaling without additional cycles, simply by setting the appropriate format in the STR command. In the example filter above, X0…X7 would be set to 32, and Q8 format would be used to right-shift the result by 8*32=256).

OBSOLETE

4.2 Initialization

The following example code segment initializes X-, Y- and I-RAM and performs a calculation using the MAC unit (average of four values):

```
#define XRAM_START ((short int *)0x00C000)
#define YRAM_START ((short int *)0x00C080)
#define IRAM_START ((short int *)0x00C100)

short int xram_data[]={64, 64, 64, 64};        // coefficient data
short int yram_data[]={100, 200, 300, 400};    // variable data
short int iram_data[]={
    0xC000,                          // Clear Accu, MUL XRAM addr. 0 with YRAM addr. 0
    0x8041,                          // Add (XRAM addr. 1 x YRAM addr. 1) to Accu
    0x8082,                          // Add (XRAM addr. 2 x YRAM addr. 2) to Accu
    0x80C3,                          // Add (XRAM addr. 3 x YRAM addr. 3) to Accu
    0x6E40                           // Convert to Q8 format with clip and round, store to
    };                               // Y-Addr. 0, set HLT bit to stop the uDSP

DSP_PC = 0;                          // reset DSP Programm Counter

ptr = XRAM_START;                    // set pointer to beginning of X-RAM
for (i=0;i<4;i++) {
  *(ptr++) = xram_data[i];           // fill X-RAM with coefficients
}

ptr = YRAM_START;                    // set pointer to beginning of Y-RAM
for (i=0;i<4;i++) {
*(ptr++) = yram_data[i];             // fill Y-RAM with variables
}

ptr = IRAM_START;                    // set pointer to beginning of I-RAM
for (i=0;i<5;i++) {
  *(ptr++) = iram_data[i];           // fill I-RAM with instructions
}

DSP_CSR = 0x01;                      // start uDSP

while (DSP_CSR_RunDSP) __wait_nop(); // wait for result

result = *YRAM_START;                  // read result
```

In such a short calculation, of course the overhead for data transfer and initialization is quite big. But as typical applications, especially filters with constant coefficients, are initialized once and then are started over and over again, the overhead becomes less important.

The MAC unit program in the example above is written directly in the byte code as explained in the hardware manual. As this is not very intuitive, a small header file was implemented, allowing easier programming using predefined macros. The corresponding code now looks something like this:

```
short int iram_data[]={
  DSP_MUL(NO_OPT,0,0),
  DSP_MAC(NO_OPT,1,1),
  DSP_MAC(NO_OPT,2,2),
  DSP_MAC(NO_OPT,3,3),
  DSP_STR(HLT|CLP|RND,Q8,YRAM,0)
};
```

This now looks more like a program and is much easier to handle. The mentioned header file is included in the software example accompanying this document. Please refer to the comments in the source code for further explanations.

OBSOLETE

The contents of X-RAM, Y-RAM and IRAM after initialization are as follows:

Table 1. Content of X-RAM and Y-RAM after initialization

X-RAM		Y-RAM	
Memory address	content	Memory address	content
0xC000	64	0xC080	100
0xC002	64	0xC082	200
0xC004	64	0xC084	300
0xC006	64	0xC086	400

Table 2. Content of I-RAM after initialization

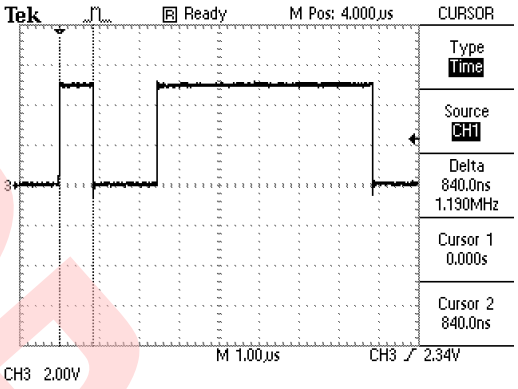
I-RAM		Meaning of instruction
Memory address	content	
0xC100	0xC000	Accu = 0 + value @ 0xC000 * value @ 0xC080
0xC102	0x8041	Accu = Accu + value @ 0xC002 * value @ 0xC082
0xC104	0x8082	Accu = Accu + value @ 0xC004 * value @ 0xC084
0xC106	0x80C3	Accu = Accu + value @ 0xC006 * value @ 0xC086
0xC108	0x6E40	Accu is converted to 16-bit value and stored @ 0xC080; HLT-bit is set and MAC-operation is stopped

4.3 Calculation performance of the MAC unit

To compare the calculation time with and without the MAC unit, two short example programs were implemented. They also demonstrate the usage of the mentioned header file for easier MAC programming. The first example calculates the average of eight values, and the second calculates either the step- or the pulse response of an 8th order binomial filter by feeding it either with a step or a pulse sequence at the input. In both cases, the calculations are performed twice, using the MAC unit and 'pure' C code without optimizations.

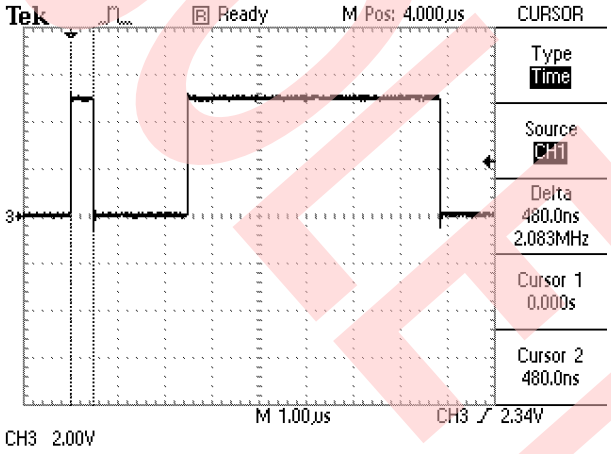
OBSOLETE

Figure 2. Example 1 calculation times with and without MAC



The first example already shows the runtime benefit when using the MAC unit quite clearly: The calculation of the average of eight values takes 5400 ns when it is calculated by the CPU itself. When using the MAC unit, a value of 880 ns is measured. This time includes not only the calculation time itself, but also the set and clear of the status pin. Also, the measurement is strongly dependent of the frequency the software polls the RunDSP flag. In a 'real' application, also other tasks can be served while the MAC is operating, or an adequate count of NOP commands can be inserted. Using 10 NOP instructions instead of the polling loop already reduces the measured time by more than 40%:

Figure 3. Example 1 with optimized MAC timing



Now, only 480 ns are measured. This time still contains the set and clear of the status pin.

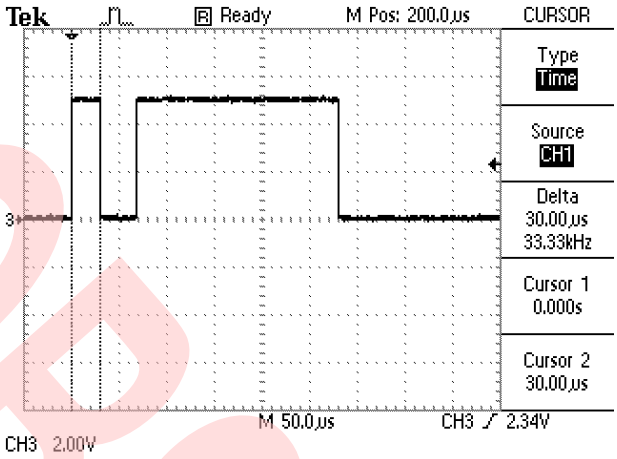
Since these values add a constant time to the MAC calculation time, the overall measured performance increases with the filter length, because the amount of the overhead becomes smaller. For example, the total runtime for a 64-tap moving average filter is about 4.6 µs.

In the second example, a 10-sample input sequence is sent through an 8th degree binomial filter. This corresponds to a convolution, so for every output value, one entire filter pass must be calculated. The filter output sequence (step response sequence) looks like this:

```
MAC results:
1  9  37  93  163  219  247  255  256  256  255  247  219  163  93  37  9  1  0  0
C-code results:
1  9  37  93  163  219  247  255  256  256  255  247  219  163  93  37  9  1  0  0
```

OBSOLETE

Figure 4. Example 2 calculation times with and without MAC

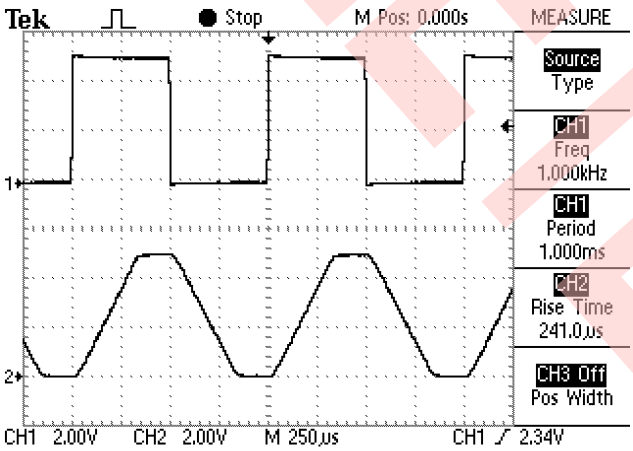


The 10-sample input sequence is zero-padded to a length of 20 samples. Then, this sequence is put through the filter, so 20 filter passes are calculated. The software filter takes about 212 µs for the processing of the entire filter response. Using the MAC unit, this time is shortened to 30 µs for the processing of all samples together. Again, this time also includes the memory transfers and the overhead of the outer loop, which passes the input samples into the filter. The variable shifting in the filter memory (the delay in every filter step) is done by the MAC unit. Therefore, the LDLY bit of the first MUL command is set, as well as the LDLY and STLY bits of all subsequent MAC commands.

In many applications, the input data will not be a fixed array, but a stream e.g. from the AD-converter. Then, the ADC ISR could be used to place the new sample into the filter and start a filter pass.

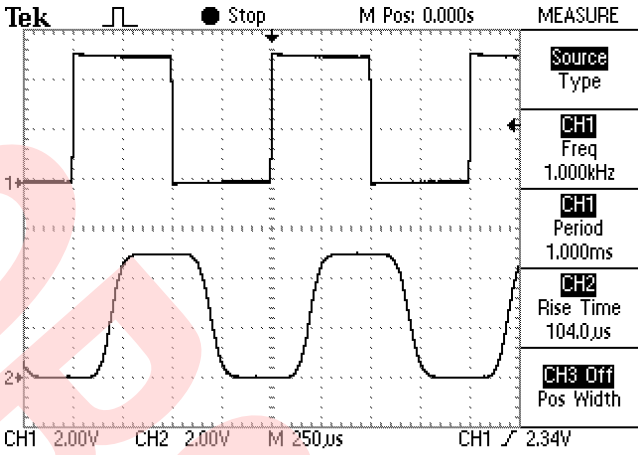
The following two scope figures show a 1 kHz rectangular signal, which is sampled by the ADC with 50 kHz, then passed through a 15-tap moving average filter (first figure) or a 15th degree binomial filter (second figure). The filtered values are output by the PPG and filtered by a simple RC filter (and an additional averaging in the scope). The processing time for each sample is about 1.2 µs.

Figure 5. Point moving average filter (ch1: input, ch2: output)



OBSOLETE

Figure 6. 15-degree binomial filter (ch1: input, ch2: output)



4.4 Using the MAC unit for PID control loops

PID control loops are widely used in many applications. Especially in motor control, often fast response times are needed which demand fast processing and high sampling frequency. The MAC unit of the MB91265 series is well suited also for this task.

Without deepening in control theory, this chapter will only demonstrate how to implement a simple PID control algorithm on the MAC unit. Lots of information about control theory can be found in literature and on the web.

One standard equation for a (discrete) PID control loop is:

```
Control_output = Kp * error + Ki * error_sum + Kd * (error - old_error)
```

Kp is the proportional gain, Ki is the gain of the integral term, and Kd is the gain of the derivative term of the control loop.

After some transformations, the control loop can be described by another equation:

```
Control_output = old_output + C0 * error(n) + C1 * error(n-1) + C2 * error(n-2)
               With: C0 = Kp + Ki + Kd,  C1 = -Kp -2Kd,  C2 = Kd
```

This form can easily be implemented using the MAC unit:

OBSOLETE

```
short int * ptr_speed_error;

void init_mac(void){
  short int * ptr;
  ptr_speed_error = (short int *) 0xC082;   // yram addr. 1 used as error variable

  DSP_CSR = STOP_DSP;                  // stop DSP, set HLT bit
  DSP_PC = 0;                          // reset DSP Programm Counter

  ptr = XRAM_START;
  *(ptr++) = 32767;                    // constant scaling for old reg output memory
  *(ptr++) = C0;                       // PID coefficients: C0=Kp+Ki*Ta+Kd/Ta
  *(ptr++) = C1;                       //                   C1=-Kp-2Kd/Ta
  *(ptr++) = C2;                       //                   C2=Kd/Ta

  ptr = YRAM_START;                    // initalize YRAM
  *(ptr++) = 0;                        // used for last control output memory
  *(ptr++) = 0;                        // current error
  *(ptr++) = 0;                        // error 1 step ago
  *(ptr++) = 0;                        // error 2 steps ago

  ptr = IRAM_START;                    // initialize uDSP programm memory
  *(ptr++) = DSP_MUL(NO_OPT,0,0);      // scale last control output value
  *(ptr++) = DSP_MAC(LDLY,1,1);        // + C0 * error(n); load error(n) to LY
  *(ptr++) = DSP_MAC(LDLY|STLY,2,2);   // + C1 * error(n-1); load error(n-1) to LY;
                                       //      overwrite error(n-1) with error(n);
  *(ptr++) = DSP_MAC(STLY,3,3);        // + C2 * error(n-2); overwrite error(n-2)
                                      // with error(n-1);
  *(ptr++) = DSP_STR(CLP|RND,Q15,YRAM,0); // store downscaled output value to YRAM addr. 0;
                                       // it is re-used in every control loop cycle
  *(ptr++) = DSP_JMP(HLT,0,0);    // jump back to first instruction (set DSP_PC to 0) and halt
}
```

The control loop itself now looks like this:

```
unsigned short int reg_speed(unsigned short int des_rpm, unsigned short int act_rpm){

//  while (DSP_CSR_RunDSP) __wait_nop();        // ensure that no MAC calculation is ongoing
                                                // (will normally not be necessary here)
  *ptr_speed_error=(des_rpm-act_rpm);           // calculate actual speed error
  DSP_CSR = START_DSP;                          // start uDSP

  __wait_nop(); __wait_nop(); __wait_nop(); __wait_nop(); // wait for uDSP completion
  _wait_nop(); __wait_nop(); __wait_nop(); __wait_nop(); // also other operations could be
                                                         // done here
  if (*YRAM_START < 0) *YRAM_START = 0;                  // keep output in PWM duty range
  else if (*YRAM_START > PWM_MAX_DUTY) *YRAM_START = PWM_MAX_DUTY;
  return (unsigned short int) *YRAM_START;
}
```

In this example, the variables (control output and speed error) are placed directly in the Y-RAM of the MAC unit, so that no additional transfers are needed. But as mentioned earlier, this means that access only is possible when the MAC unit is stopped. Since the calculation time will usually be short compared to the sampling time of the control loop, this is no big restriction. For the same reason, it often will not be necessary to check if the MAC unit is stopped before starting a new calculation.

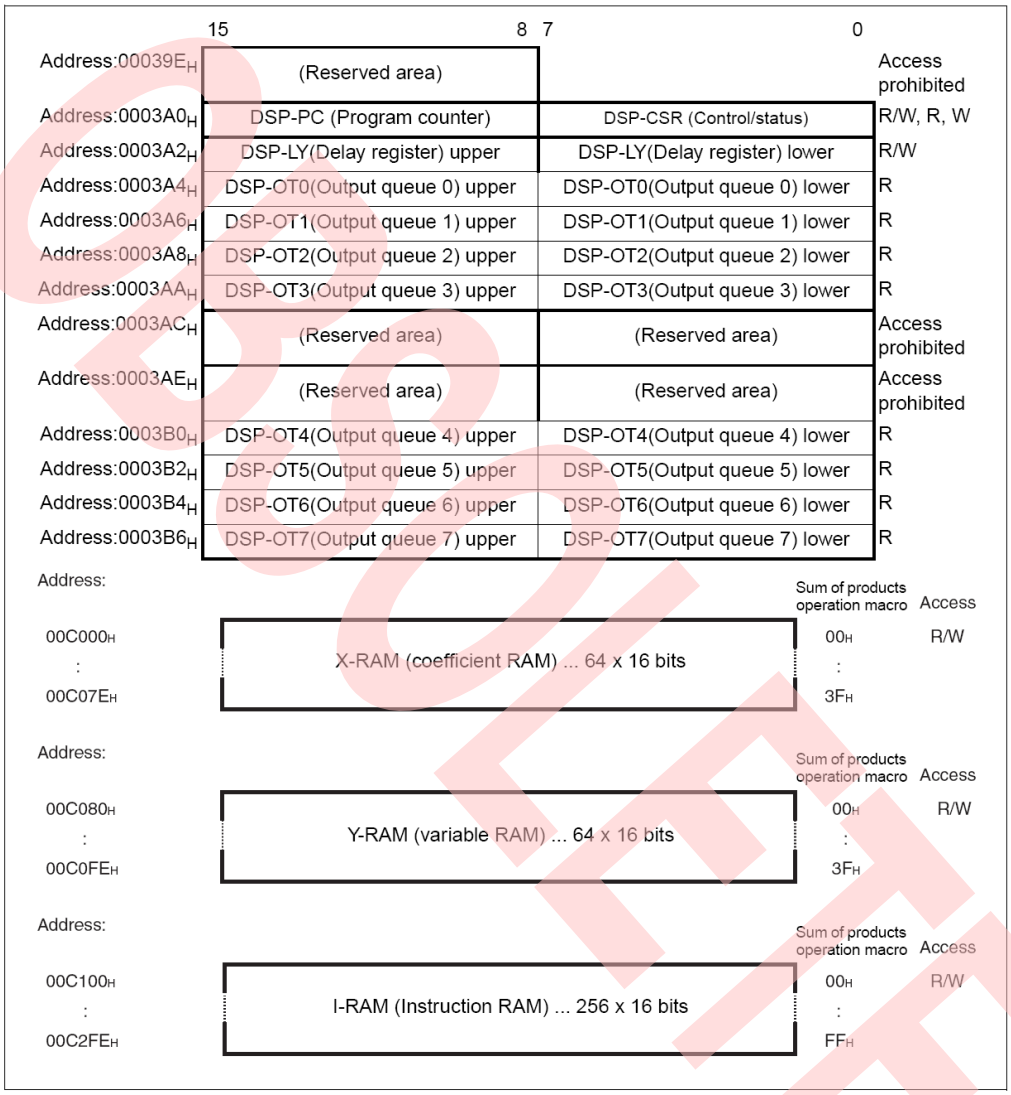
The runtime of the above control loop is below 1.4us, including overhead like the operations for the status pin. During the eight cycles which the MAC unit needs for the calculation, other tasks could be performed as well instead of the NOPs. As an example, some sensor information could be updated. But also without additional tweaking, the entire calculation time for the PID controller is reduced by more than 25% compared to pure C code with the same functionality.

OBSOLETE

A Appendix

A.1 Register list and memory map of the MAC unit

Figure 7. Register list and memory map



A.2 Related Documents

- hm91265-cm71-10130-2e.pdf

OBSOLETE

A.3 Glossary

CPU	Central Processing Unit
DSP	Digital Signal Processor, Digital Signal Processing
FIR	Finite Impulse Response (Filter)
IIR	Infinite Impulse Response (Filter)
IRQ	Interrupt ReQuest
ISR	Interrupt Service Routine
MAC	Multiply-and-ACcumulate
MCU	Micro-Controller Unit
PID	Proportional-Integral-Derivative (Control loop)

OBSOLETE

5 Document History

Document Title: AN205207 - FR Family MB91265 Series 16-Bit MAC Unit

Document Number: 002-05207

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	NOFL	08/09/2006	Initial release
			05/08/2007	Complete revision, filtering applications added, PID example added
			05/10/2007	Typo + index corrected
*A	5123375	NOFL	03/22/2016	Migrated Spansion Application Note from MCU-AN-300030-E-V12 to Cypress format
*B	5843042	AESATP12	08/03/2017	Updated logo and copyright.
*C	6061730	SSAS	02/07/2018	Obsolete

OBSOLETE

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2006-2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.