REFERENCES


---

**DSP Techniques for Real-Time Applications**

Patrick Heath
Motorola
Austin, Texas

Patrick Heath is Motorola's Advanced MCU technical marketing manager. Among other tasks, he writes software demonstrating how to take advantage of new MCU features. He is also responsible for Motorola's strategy of low cost evaluation boards. After receiving separate Bachelor's degrees in computer science, mathematics, and business administration from Graceland College, and a Master's degree in computer science from the University of Missouri, Rolla, Heath worked for IBM in Lexington, Kent, before joining Motorola in Austin, Texas.
DSP Techniques in Real-Time Applications using the Motorola MC68HC16

by

Patrick N. Heath
Advanced MCU Technical Marketing Mgr.
Motorola, Inc.

CPU16 DSP Applications

- Math Intensive Control
  - Servo Control: disk drives, motor control, robotics
  - Process Control
  - Medical Instruments
  - Automotive: emissions control, cruise control
- Filtering
  - Noise Filtering
  - Motion Detection
  - Low-end audio filters
- Signal Detection
  - Extracting periodic signals from noise
  - Demodulation
- Linearization of Sensor Readings
- Not possible:
  - High-end filters, modems
  - Speech processing, image processing
  - Graphics
DSP Techniques in Real-Time Applications using the Motorola MC68HC16

by
Patrick N. Heath
Advanced MCU Technical Marketing Mgr.
Motorola, Inc.

CPU16 DSP Applications

- Math Intensive Control
  - Servo Control: disk drives, motor control, robotics
  - Process Control
  - Medical Instruments
  - Automotive: emissions control, cruise control

- Filtering
  - Noise Filtering
  - Motion Detection
  - Low-end audio filters

- Signal Detection
  - Extracting periodic signals from noise
  - Demodulation

- Linearization of Sensor Readings

- Not possible:
  - High-end filters, modems
  - Speech processing, image processing
  - Graphics
**Benefits of Digital Signal Processing**

- Stability - no drift with temperature or age
- Reduced system component count - reduced cost
- Improved filter precision, especially for higher-order filters
- Parameters are easily modified
- No calibration needed
- Non-linear function generation and linear-phase filtering possible
- Complexity of the signal processing becomes independent of hardware
- Lower system power consumption

**MC68HC16 = MCU + DSP**

Since the analog signals are already digitized by the Analog to Digital Converter, simply use the on-chip DSP to improve control.

**Disadvantages of Digital Signal Processing**

- Limited resolution of data - determined by A/D Converter (10 bits)
- Limited bandwidth - determined by CPU16 throughput
- Limited dynamic range - determined by MAC hardware (16x16+32)
- Quantization noise - caused by A/D conversion process
- Round-off errors - caused by D/A conversion process

These can be improved by increasing resolution, bandwidth and range.

High resolution, bandwidth and dynamic range = High Cost

**What is DSP?**

DSP is converting real-time analog signal waveforms into digital representation and then processing them in software using a microprocessor. Processed signal samples may then be used to reconstruct continuous output waveforms.

[Diagram showing signal processing flow: Analog In - Sample & Hold - ADC - DSP - DAC - Analog Out]
Benefits of Digital Signal Processing

- Stability - no drift with temperature or age
- Reduced system component count - reduced cost
- Improved filter precision, especially for higher-order filters
- Parameters are easily modified
- No calibration needed
- Non-linear function generation and linear-phase filtering possible
- Complexity of the signal processing becomes independent of hardware
- Lower system power consumption

MC68HC16 = MCU + DSP

Since the analog signals are already digitized by the Analog to Digital Converter, simply use the on-chip DSP to improve control.

Disadvantages of Digital Signal Processing

- Limited resolution of data - determined by A/D Converter (10 bits)
- Limited bandwidth - determined by CPU16 throughput
- Limited dynamic range - determined by MAC hardware (16x16+32)
- Quantization noise - caused by A/D conversion process
- Round-off errors - caused by D/A conversion process

These can be improved by increasing resolution, bandwidth and range.

High resolution, bandwidth and dynamic range = High Cost

What is DSP?

DSP is converting real-time analog signal waveforms into digital representation and then processing them in software using a microprocessor. Processed signal samples may then be used to reconstruct continuous output waveforms.
An Introduction to DSP

Analog Signal Processing

\[ v(t) = u(t) \left[ 1 + \frac{1}{2} e^{-\frac{t}{RC}} \right] \]

Digital Signal Processing

\[ y(n) = \frac{1}{2} x(n) + b y(n-1) \]

Basic DSP Operation Flowchart

1. Clear Accum
2. Load H and I
3. Multiply current A/D sample in H by coefficient in I
   Add product to Accum and store back in Accum
   Advance pointers IX and IY
4. Goto Step 2

Accum M = Current Filter Output Signal

This sequence is functionally equivalent to the CLRM, LDHI and MAC instructions.

Delay Time of the filter is determined by the sampling rate of the ADC.

CPU16 DSP Support Features

- MAC (multiply and accumulate) and RMAC (repealing MAC) instructions; the RMAC instruction is interruptible - basic DSP operation support.
- All instructions operate using fixed point, 16-bit signed fractional data sufficient magnitude for control-oriented DSP ranges and digital signals support transcendental functions.
- Modular addressing (programmable pointer masks) to support circular buffers - improves throughput.
- Data pointers for the H register and I register are automatically incremented and/or decremented.
- 32-bit accumulator with 4 bits of overflow - improves throughput by postponing the need to scale results on every iteration.
- 2 types of overflow detection - Saturation support option - simulates performance of an analog system.
- Result rounding or truncation option - controls noise or error in calculations.
- Result scaling for transferring to a 16-bit accumulator or output device.

DSP Data Types

- Multiplicand and multiplier are 16-bit signed fractions.
- Multiplication product is a 32-bit signed fraction.
- Samples and coefficients are normalized by shifting into the range between -1 to 1-2EE-15 before DSP operations and denormalized afterwards.
- The MAC accumulator uses 36 bits.
- There is an implied radix point between bits 31 and 30. The range of values is from -1 to 1-2EE-30 ($7FFFFFFF$).

DSP Techniques

Heath

340
**An Introduction to DSP**

**Analog Signal Processing**

\[ v(t) = u(t) \left( \frac{1}{2} e^{-\frac{t}{RC}} \right) \]

**Digital Signal Processing**

\[ y[n] = \frac{1}{2} x[n] + bx[n-1] \]

---

**Basic DSP Operation Flowchart**

1. Clear AccM
2. Load H and I
3. Multiply current A/D sample in H by coefficient in I
   - Add product to AccM and store back in AccM
   - Advance pointers IX and IY
4. Goto Step 2

AccM = Current Filter Output Signal

This sequence is functionally equivalent to the CLRM, LDHI and MAC instructions.

- Delay Time of the filter is determined by the sampling rate of the ADC

---

**CPU16 DSP Support Features**

- MAC (multiply and accumulate) and RMAC (repetitive MAC) instructions: the RMAC instruction is interruptible - basic DSP operation support.
- All instructions operate using fixed point, 16-bit signed fractional data sufficient magnitude for control-oriented DSP ranges and signed numbers support transcendental functions.
- Data pointers for the H and I registers are automatically incremented and/or decremented.
- 36-bit accumulator with 4 bits of overflow improves throughput by postponing the need to scale results on every iteration.
- 2 types of overflow detection.
- Saturation support option - simulates performance of an analog system.
- Result rounding or truncation option - controls noise or error in calculations.
- Result scaling for transferring to a 16-bit accumulator or output device.

---

**DSP Data Types**

- Multiplicand and multiplier are 16-bit signed fractions.

---

**Heath DSP Techniques 341**

---
**MAC Instruction Format Example**

**MAC 2,2**

Prior to Execution

<table>
<thead>
<tr>
<th>X-reg</th>
<th>Y-reg</th>
<th>IX-reg</th>
<th>Y-reg</th>
<th>IX-reg</th>
<th>Y-reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>12001</td>
<td>14001</td>
<td>10001</td>
<td>15001</td>
<td>10001</td>
<td>15001</td>
</tr>
<tr>
<td>H-reg</td>
<td>I-reg</td>
<td>E-reg</td>
<td>D-reg</td>
<td>L-reg</td>
<td>H-reg</td>
</tr>
<tr>
<td>10001</td>
<td>10001</td>
<td>10001</td>
<td>10001</td>
<td>10001</td>
<td>10001</td>
</tr>
</tbody>
</table>

After Execution

<table>
<thead>
<tr>
<th>X-reg</th>
<th>Y-reg</th>
<th>IX-reg</th>
<th>Y-reg</th>
<th>IX-reg</th>
<th>Y-reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>13001</td>
<td>15001</td>
<td>10001</td>
<td>16001</td>
<td>10001</td>
<td>16001</td>
</tr>
<tr>
<td>H-reg</td>
<td>I-reg</td>
<td>E-reg</td>
<td>D-reg</td>
<td>L-reg</td>
<td>H-reg</td>
</tr>
<tr>
<td>16001</td>
<td>16001</td>
<td>16001</td>
<td>16001</td>
<td>16001</td>
<td>16001</td>
</tr>
</tbody>
</table>

**Note:** RMAC flow slightly different

---

**Modulo Addressing To Implement Circular Buffers**

- Useful for FIR (Finite Impulse Response) filters.
- The Modulo Addressing Mode is used only with the MAC and RMAC (repeating MAC) instructions. It is implemented by two hidden registers called the X-mask and the Y-mask. These registers hold 8-bit signed values.
- Once the end of the buffer has been reached, the address pointer will automatically wrap around to point to the beginning of the buffer. Because this wraparound mode removes the need for address boundary tests, filter speed is greatly enhanced.
- Buffer sizes range from 2 to 256 bytes in power-of-two increments.
- I.e. X-mask=00000000 disables modulo addressing for IX
- X-mask=000000001 implements a 2 byte buffer for IX
- X-mask=0000000011 implements a byte buffer for IX
- Modulo Addressing is not supported across bank boundaries, and IX and IV must have a base address on a byte block boundary corresponding to the size of the buffer.
**Detail of MAC and RMAC Instruction**

MAC:  \[ \text{AccE:AccD} \leftarrow (\text{H-Reg} \times \text{(!-Reg)} \right] \\
\text{AceM} \leftarrow (\text{AceM}) + (\text{AccE:AccD}) \\
\text{IX} \leftarrow (\text{IJQ} + \text{X-offset}, \text{qualified bv X mask}) \\
\text{IV} \leftarrow (\text{IY}) + \text{V-offHt}, \text{qualified bv V mask} \\
\text{IZ} \leftarrow (\text{H-Reg}) \\
\text{H·Reg} \leftarrow (\text{IX}) \\
\text{!-Reg} \leftarrow (\text{IV})

RMAC:  \[ \text{AccM} \leftarrow (\text{H-Reg} \times (\text{!-Reg}) + \text{Acc:M}) \\
\text{IX} \leftarrow (\text{IX}) + \text{X-offut}, \text{qualified bv X mask} \\
\text{IV} \leftarrow (\text{IY}) + \text{V-offut}, \text{qualified bv V mask} \\
\text{IZ} \leftarrow (\text{H-Reg}) \\
\text{H·Reg} \leftarrow (\text{IX}) \\
\text{!-Reg} \leftarrow (\text{IV})

**MAC Instruction Format Example**

Prior to Execution

<table>
<thead>
<tr>
<th>Address</th>
<th>H1</th>
<th>H0</th>
<th>I1</th>
<th>I0</th>
<th>E1</th>
<th>E0</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Register</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>Y-Register</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>IX</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>IV</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>X-mask</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
</tr>
<tr>
<td>Y-mask</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
</tr>
</tbody>
</table>

After Execution

<table>
<thead>
<tr>
<th>Address</th>
<th>H1</th>
<th>H0</th>
<th>I1</th>
<th>I0</th>
<th>E1</th>
<th>E0</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Register</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>Y-Register</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>IX</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
</tr>
<tr>
<td>IV</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
</tr>
<tr>
<td>X-mask</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
</tr>
<tr>
<td>Y-mask</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
</tr>
</tbody>
</table>

**Note:** RMAC flow slightly different

**Modulo Addressing To Implement Circular Buffers**

- Useful for FIR (Finite Impulse Response) filters.
- The Modulo Addressing Mode is used only with the MAC and RMAC (repeating MAC) instructions. It is implemented by two hidden registers called the X-mask and the Y-mask. These registers hold 8-bit signed values.
- Once the end of the buffer has been reached, the address pointer will automatically wrap around to point to the beginning of the buffer. Because this wraparound mode removes the need for address boundary tests, filter speed is greatly enhanced.
- Buffer sizes range from 2 to 256 bytes in power-of-two increments.
  - i.e. X-mask=%00000000 disables modulo addressing for IX
  - X-mask=%00000001 implements a 2 byte buffer for IX
  - Y-mask=%11111111 implements a 256 byte buffer for IV
- Modulo Addressing is not supported across bank boundaries, and IX and IV must have a base address on a byte block boundary corresponding to the size of the buffer.

**DSP Instruction Set**

- **Transfer/Initialization**
  - TEM: Transfer Accumulator E to Accumulator M[1:16]
  - TEDM: Transfer AccE and AccD to Accumulator M
  - TMER: Transfer Accumulator E to Accumulator E and Round
  - TMET: Transfer Accumulator M to Accumulator E and Truncate
  - TMMTR: Transfer Accumulator M to H-Mask and Y-Mask
  - CLRM: Clear Accumulator M
  - LDH: Load MAC Registers H and I

- **AccM Scaling**
  - ASM: Arithmetic Shift Left of Accumulator M
  - ASRM: Arithmetic Shift Right of Accumulator M

- **Multiply and Accumulate**
  - MAC: Multiply and Accumulate
  - RMAC: Repeating MAC instruction (interceptable)
  - ACED: Adds AccE:AccD to Accumulator M
  - ACE: Adds AccE to Accumulator M

- **Miscellaneous**
  - LBVM: Branch if MV bit is set
  - PSXM: Loads H, I, M[1:16], M[1:32], IX and IV masks
  - PUVM: Pushes H, I, M[1:16], M[1:32], IX and IV masks
  - FMULS: Fractional Signed Multiply
Accumulator M Overflow Provisions

- Accumulator M provides 4 bits (out of 36) to hold data overflow overflow. Two types of overflow are detected:
  - EV Overflow:
    - Recoverable, transient overflow into AccM bit [31:34]
    - The EV flag is set in the CCR
    - The EV flag is updated (set or cleared) with every MAC instruction (or RMAC iteration)
  - MV Overflow:
    - Non-recoverable, catastrophic overflow into AccM bit [35]
    - MV overflow bit is set in the CCR
    - If AccM < 16 or AccM > 15.999969482, then MV = 1 (and EV will also be set to 1)
    - Once set, MV can only be cleared by software

<table>
<thead>
<tr>
<th>Accumulator Magnitude</th>
<th>M35</th>
<th>M34</th>
<th>M31</th>
<th>M28</th>
<th>EV bit set in CCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 AccM = 80000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes (overflow)</td>
</tr>
<tr>
<td>06 AccM = -80000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No (no overflow)</td>
</tr>
<tr>
<td>-16 AccM = -80000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Yes (overflow)</td>
</tr>
</tbody>
</table>

S and E Bit Values in Accumulator M

Convergent Rounding

After all MAC instructions have been completed, AccM must be transferred to AccE to access the result. Convergent rounding minimizes the error.

- TMET: no rounding occurs; least significant bits are truncated
- TMER: convergent rounding of AccM; transfers result to AccE after adding $8000000000

AccM[15:0] AccE[15:0] Operation
- $8000 Rounds-up if remainder = 0.5
- $80000 Round-off if remainder > 0.5
- $80000 Round-off to even number
- $80000 Round-off to odd number

- EV and MV are updated during TMER operation (as well as N and Z, if applicable)
- AccM is not modified by the TMET or TMER instructions

Data Saturation

- Data Saturation simulates the effect of saturation in analog systems.
- Saturation Mode is user-specified by setting the SM bit in the Condition Code Register.
- During a TMER or TMET instruction, a corrected saturation value will be loaded into AccE only if the SM bit in the CCR is set (requesting data saturation mode) and the EV and/or MV bit in the CCR is set (indicating overflow has occurred in AccM).

Relative DSP Performance

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CPU/16 (16.67MHz)</th>
<th>TMS320C10 (16.67MHz)</th>
<th>TMS320C10 (25MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC: 16 x 16 36 bit Accumulator</td>
<td>2 bytes, 720 nSec</td>
<td>4 bytes, 480 nSec</td>
<td>320 nSec</td>
</tr>
<tr>
<td>12th order IIR Filter</td>
<td>58 bytes, 17.3 KHz(1)</td>
<td>156 bytes, 25.7kHz</td>
<td>40.1 KHz</td>
</tr>
<tr>
<td>65 tap FIR Filter</td>
<td>26 bytes, 10.3 KHz</td>
<td>266 bytes, 15.7 KHz</td>
<td>23.5 KHz</td>
</tr>
<tr>
<td>PID (2)</td>
<td>24 bytes, 4.4 uSec/loop</td>
<td>32 bytes, 3.1 uSec/loop</td>
<td></td>
</tr>
</tbody>
</table>

(1) 6 x 2nd order subroutine calls
(2) Based on u(n) = u(n-1) + K1 * x(n) + K2 * x(n-1) + K3 * x(n-2)

Data Saturation Table

<table>
<thead>
<tr>
<th>CCR Bits</th>
<th>EV</th>
<th>MV</th>
<th>Overflow Bits</th>
<th>Saturation Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0</td>
<td>0</td>
<td>0</td>
<td>0001 $E$: 1111</td>
<td>7FFF Maximum Positive Number</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 1</td>
<td>0</td>
<td>1</td>
<td>1110 $E$: 0000</td>
<td>8000 Maximum Negative Number</td>
<td></td>
</tr>
<tr>
<td>0 1 1 1 1 1</td>
<td>1</td>
<td>1</td>
<td>$E$: 1110</td>
<td>7FFF Maximum Positive Number</td>
<td></td>
</tr>
<tr>
<td>1 1 1 1 1 1</td>
<td>1</td>
<td>1</td>
<td>$E$: 0000</td>
<td>8000 Maximum Negative Number</td>
<td></td>
</tr>
</tbody>
</table>

All other values do not cause saturation.

SL: A hidden register (Sign latch) ensures correct saturation when AccM has catastrophically overflowed into AccM[35] by saving the value of the AccM[35] bit prior to overflow.
Accumulator M Overflow Provisions

- Accumulator M provides 4 bits (out of 36) to hold data overflow. Two types of overflow are detected:
  - EV Overflow:
    - Recoverable, transient overflow into AccM bit [31:34]
    - The EV flag is set in the CCR
    - The EV flag is updated (set or cleared) with every MAC instruction (or RMAC iteration)
  - MV Overflow:
    - Non-recoverable, catastrophic overflow into AccM bit [35]
    - MV overflow bit is set in the CCR
    - If AccM < 16 or AccM > 15.999969482, then MV=1 (and EV will also be set to 1)
    - Once set, MV can only be cleared by software

<table>
<thead>
<tr>
<th>AccM Magnitude</th>
<th>M35</th>
<th>M34</th>
<th>EV bit set in CCR</th>
<th>S and E Bit Values in Accumulator M</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ≤ AccM [35] ≤ 0.999969482</td>
<td>0</td>
<td>0</td>
<td>Yes (overflow)</td>
<td>S ≤ 1 ≤ E ≤ 1111</td>
</tr>
<tr>
<td>0 ≤ AccM ≤ 35</td>
<td>0</td>
<td>1</td>
<td>No (no overflow)</td>
<td>E = 0000</td>
</tr>
<tr>
<td>1 ≤ AccM &lt; 36</td>
<td>1</td>
<td>0</td>
<td>No (no overflow)</td>
<td>Sign Extension Only</td>
</tr>
<tr>
<td>1 ≤ AccM &lt; 36</td>
<td>1</td>
<td>1</td>
<td>Yes (overflow)</td>
<td>E bits should be set to 1 if AccM ≤ 1110</td>
</tr>
</tbody>
</table>

Convergent Rounding

After all MAC instructions have been completed, AccM must be transferred to AccE to access the result. Convergent rounding minimizes the error.
- TMET: no rounding occurs; least significant bits are truncated
- TMER: convergent rounding of AccM; transfers result to AccE after adding $00000000$

<table>
<thead>
<tr>
<th>S ≤ 1 ≤ E ≤ 1111</th>
<th>0 ≤ S ≤ 1 ≤ E ≤ 1111</th>
<th>AccM [15:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>S ≤ 1 ≤ E ≤ 1111</td>
<td>0 ≤ S ≤ 1 ≤ E ≤ 1111</td>
<td>36 bytes</td>
</tr>
<tr>
<td>S ≤ 1 ≤ E ≤ 1111</td>
<td>0 ≤ S ≤ 1 ≤ E ≤ 1111</td>
<td>36 bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AccM [16]</th>
<th>AccM [15:0]</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; $8000</td>
<td>$8000 - $F000</td>
<td>Rounds-up if remainder = 0.5</td>
</tr>
<tr>
<td>$8000</td>
<td>$8000 - $F000</td>
<td>No rounding if remainder = 0.5</td>
</tr>
<tr>
<td>1</td>
<td>$8000 - $F000</td>
<td>Rounds-up to even number</td>
</tr>
<tr>
<td>0</td>
<td>$8000 - $F000</td>
<td>Rounds-up to odd number</td>
</tr>
</tbody>
</table>

- EV and MV are updated during TMER operation (as well as N & Z, if applicable)
- AccM is not modified by the TMET or TMER instructions

Data Saturation

- Data Saturation simulates the effect of saturation in analog systems.
  - Saturation Mode is user specified by setting the SM bit in the Condition Code Register.
  - During a TMER or TMET instruction, a corrected saturation value will be loaded into AccM only if the SM bit in the CCR is set (requesting data saturation mode) and the EV and/or MV bit in the CCR is set (indicating overflow has occurred in AccM).

Saturation Values Table:

<table>
<thead>
<tr>
<th>AccM Bit Set</th>
<th>M35</th>
<th>M34</th>
<th>M31-M0</th>
<th>EV</th>
<th>MV</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Relative DSP Performance

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CPU/16 (16.67MHz)</th>
<th>TMS320C10 (16.67MHz/25MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC: 16 x 16 36 bit Accumulator</td>
<td>2 bytes, 720 nSec</td>
<td>2 bytes, 480 nSec/320 nSec</td>
</tr>
<tr>
<td>12th order IIR Filter</td>
<td>114 bytes, 15.95 KHz</td>
<td>58 bytes, 17.3 KHz</td>
</tr>
<tr>
<td>65 tap FIR Filter</td>
<td>26 bytes, 10.35 KHz</td>
<td>266 bytes, 15.7 KHz/23.5 KHz</td>
</tr>
<tr>
<td>PID (2)</td>
<td>24 bytes, 4.4 uSec/loop</td>
<td>22 bytes, 3.1 uSec/loop</td>
</tr>
</tbody>
</table>

(1) 6 x 2nd order subroutine calls
(2) Based on u(n) = u(n-1) + K1 * n(n-1) + K2 * u(n-1) + K3 * u(n-2)
Servo Motor Control System Diagram

The MC68HC16Y1 is used to implement a PID servo motor control using DSP instructions. The TPU outputs PWM signals to an interface module which centers the signals and feeds them to the H bridge drivers found on the Motor Interface Module. The output of the shaft encoder is fed back to the TPU for quadrature (position) decode. A SCI link to a terminal or PC provides a method for entering a new motor position, as well as specifying a maximum velocity and acceleration.

MC68HC16Z1 Frequency Analyzer System

Implementation: 5 - 1st Order IIR Filters, centered at 100, 200, 1K, 4K & 10KHz

Software Design Concept

Hardware Block Diagram

Implementation: Sum Left & Right Channels into anti-aliasing filter. Adjust output to feed ADC. Output 5 band peak value filters thru the QSPI to serial LED drivers for display on LED strips.
Servo Motor Control System Diagram

The MC68HC16Y1 is used to implement a PID servo motor control using DSP instructions. The TPU outputs PWM signals to an interface module which centers the signals and feeds them to the H bridge drivers found on the Motor Interface Module. The output of the shaft encoder is fed back to the TPU for quadrature (position) decode. A SCI link to a terminal or PC provides a method for entering a new motor position, as well as specifying a maximum velocity and acceleration.

MC68HC16Z1 Frequency Analyzer System

Implementation: 5 - 1st Order IIR Filters, centered at 100, 200, 1K, 4K & 10KHz

Software Design Concept

Hardware Block Diagram

Implementation: Sum Left & Right Channels into anti-aliasing filter. Adjust output to feed ADC. Output 5 band peak value filters thru the QSPI to serial LED drivers for display on LED strips.
Time Constraints

To avoid missing information from the audio signal, the incoming signal must be sampled at least 2x the highest filter rate (10KHz):

\[ \text{Fs} = \text{Sampling frequency} \]
\[ \text{Ts} = \text{Sampling period} \]
\[ \text{Fs} = \text{HC16 system clock frequency} \]
\[ \text{Fs} = 34.95 \text{ KHz} \]
\[ \text{Ts} = 14.9 \text{ usec} \]
\[ \text{Fs} = 16.76 \text{ MHz} \]
\[ \text{Ts} = 0.05 \text{ usec} \]

System clock cycle/sampling period =
\[ \text{Ts} / \text{Te} = 60 \text{ usec} / 60 \text{ nsec} \]

Implementation: The main software loop to do this had to be very tight in order to do everything necessary in 668 system clock cycles. 20 or 25 MHz speeds are really needed to do this.

Anti-Aliasing Filter

Needed to prevent unwanted input from frequencies in between the sampling frequency and 1/2 of the sampling frequency.

\[ \text{Mag} (\text{dB}) \]
\[ \text{Freq} (\text{KHz}) \]
\[ \text{Transfer Function without low pass anti-aliasing filter} \]
\[ \text{Transfer Function with low pass anti-aliasing filter} \]

Implementation: We used a filter chip by Maxim to do our anti-aliasing. This attenuates (decreases the magnitude of) the signals over 10 KHz.

Audio Signal Bias

Because the CPU16/DSP operates on left-justified signed numbers, the incoming analog signal was adjusted to be centered at 2.5 volts, rather than 0 volts, to provide the maximum magnitude.

By the way, this waveform is an example of what sound looks like. A 10KHz (x-axis) waveform would be a high pitched tone, and a 100 Hz waveform would be a low tone. The (y-axis) magnitude determines the loudness.

The ADC’s maximum negative number (-80) will be 0 volts, and the maximum positive number (7F) will be 5 volts.
Time Constraints

To avoid missing information from the audio signal, the incoming signal must be sampled at least 2x the highest filter rate (10KHz):

Fs = Sampling frequency
Ts = Sampling period
Fc = HC16 system clock frequency
Te = HC16 system clock period

Fs = 34.95 KHz
Ts = 1/16 = 60.00 usec
Fc = 16.76 MHz
Te = 3.96 = 60 nsec

System clock cycles/sampling period:
Ts/Te = 668 system clock cycles

Implementation: The main software loop to do this had to be very tight in order to do everything necessary in 668 system clock cycles! 20 or 25 MHz speeds are really needed to do this.

Anti-Aliasing Filter

Needed to prevent unwanted input from frequencies in between the sampling frequency and 1/2 of the sampling frequency

Implementation: We used a filter chip by Maxim to do our anti-aliasing. This attenuates (decreases the magnitude of) the signals over 10 KHz.

Audio Signal Bias

Because the CPU16/DSP operates on left-justified signed numbers, the incoming analog signal was adjusted to be centered at 2.5 volts, rather than 0 volts, to provide the maximum magnitude.

By the way, this waveform is an example of what sound looks like. A 10KHz (x-axis) waveform would be a high pitched tone, and a 100 Hz waveform would be a low tone. The (y-axis) magnitude determines the loudness.

The ADC's maximum negative number (3F) will be 0 volts, and the maximum positive number (7F) will be 5 volts.
Band-Pass Filters

Our software must sample the incoming signal at 25KHz, break the signal into 5 different bands, and determine the peak amplitude of each filter band. A band-pass 1st order IIR filter does this for us. Here's the equation:

\[ y(n) = 2 \cdot \left( x(n \cdot 2) - y(n-1) - y(n-2) \right) \]

\( X(n) \) are ADC samples taken at 25KHz, and \( Y(n) \) is the filtered output.

For our 1KHz filter for example, the coefficients can be calculated by:

\[ F_0 = 1 \text{ KHz} \quad F_s = 24.95 \text{ KHz} \]
\[ Q = 1.5 \quad \alpha = \left( 2 \cdot x \cdot F_0 \right) / F_s \]
\[ x = 0 \quad \text{if} \quad x \leq 4, \text{then} \quad x = \text{round}\left( x \cdot 0.75 \right) \]
\[ \beta = 0.5 \quad \left( 1 - \tan\left( \frac{\pi}{2} \cdot Q \right) \right) / \left( 1 + \tan\left( \frac{\pi}{2} \cdot Q \right) \right) \]
\[ \gamma = (0.5 - 5) \cdot \cos(\pi/2) \]
\[ \alpha = (0.5 - 5) / 2 \]

Displaying the Results

In order to display the results on the bar LED's, we must encode the output from the filters, so that it corresponds to the scale used on the LED's. We picked the analog value 0.775 to serve as a reference number, we'll say it is equal to 0 decibels (dB). This is the same number used by CD players for their input. This value will cause 6 of the 8 LED's to light up in our system.

The following equations are used to calculate the analog input voltage (Vin) equivalent to a discernable change in sound (about 3 dB). Vrms = Vpeak times the square root of 2 divided by 2.

\[ dB = 20 \cdot \log \left( \frac{Vin}{Vref} \right) \]
\[ 0 \text{ dB} \Rightarrow Vref = 0.775 \text{ Vrms} \]
\[ Vpeak = \sqrt{2} \cdot \text{Vrms} \]

Software Flow Diagram
**Band-Pass Filters**

Our software must sample the incoming signal at 25KHz, break the signal into 5 different bands, and determine the peak amplitude of each filter band. A band-pass 1st order IIR filter does this for us. Here’s the equation:

\[
y(n) = 2 \left[ (x - x(n - 2)) + y(n - 1) - y(n - 2) \right]
\]

\(x(n)\) are ADC samples taken at 25KHZ, and \(y(n)\) is the filtered output.

For our 1KHz filter for example, the coefficients can be calculated by:

\[
Q = \frac{10}{\log(2 \pi / F_0)}
\]

\[
\alpha = 0.5 (1 - \tan(\pi / (2 \cdot Q)) / (1 + \tan(\pi / (2 \cdot Q)) )
\]

\[
\gamma = (0.5 \beta) \cos(\alpha)
\]

\[
\beta = \frac{0.5 - \beta}{2}
\]

---

**Displaying the Results**

In order to display the results on the bar LED's, we must encode the output from the filters, so that it corresponds to the scale used on the LED's. We picked the analog value 0.775 to serve as a reference number, we'll say it is equal to 0 decibels (dB). This is the same number used by CD players for their input. This value will cause 6 of the 8 LED's to light up in our system.

The following equations are used to calculate the analog input voltage (Vin) equivalent to a discernable change in sound (about 3 dB). Vrms = Vpeak times the square root of 2 divided by 2.

\[
dB = 20 \cdot \log \left( \frac{Vin}{Vref} \right)
\]

0 dB \(\Rightarrow Vref = 0.775 Vrms\)

Vpeak = \(Vrms \cdot \sqrt{2}\)