Interfaces

Uwe R. Zimmer - The Australian National University
References

[Burns2007] Burns, A & Wellings, A
Concurrent and Real-Time Programming in Ada, edition
Cambridge University Press 2007

[Motorola1996]
Motorola
Time Processing Unit

[Motorola2000]
Motorola
MPC565 & MPC566
2000 pp. 1-1312

[Peacock1997]
Peacock, GR
Standards for temperature sensors

[Motorola2000]
Motorola
MPC565 & MPC566
2000 pp. 1-1312
Interfaces

Real-Time Systems Components: Interfaces

Real-Time Software: Algorithms, Languages, Operating systems, Communication Systems
A/D, D/A & Interfaces

Signal chain

Original signal
(weakish, noise- and interference-affected, carries additional higher frequency signals)
Interfaces

A/D, D/A & Interfaces

Signal chain

Amplified signal

(stronger, noise- and interference-affected, carries additional higher frequency signals)
Interfaces

A/D, D/A & Interfaces

Signal chain

Filtered ("low passed") signal
(higher frequency signals have been eliminated – essential precondition for next stage)
A/D, D/A & Interfaces

Signal chain

Signal after sample-and-hold circuit
(samples are taken (over short time-span) and held until the next sample time)
A/D, D/A & Interfaces

Signal chain

Discrete values inside CPU memory (after A/D conversion)
(Discrete, quantized representation)
Interfaces

A/D, D/A & Interfaces

Signal chain

Discrete signal at CPU output gate
(Discrete, quantized representation presented on digital output interface)
Signal chain

Analogue signal after D/A conversion
(Limited bandwidth leads to glitches in the analogue signal)
Signal chain

Smoothed ("deglitched") signal
(Synchronized filter leads to predictable, analogue transition steps)
Signal chain

Filtered (“low passed”) signal
(Signals introduced by the conversion process are eliminated)
Sampling

Sample data with frequency $f_s$
Sampling

Sample data with frequency $f_s$

Interpolation suggests a source signal
Interfaces

A/D, D/A & Interfaces

Sampling

Sample data with frequency $f_s$
Sample data with frequency $f_s$

- Interpolation suggests a source signal
- The phenomenon of the wrongly observed signal at a lower frequency $f_s$ is called **aliasing**.
An analog signal with bandwidth $f_a$ must be sampled at:

$$f_s > 2f_a$$

Perfect measurements taken at $f_s > 2f_a$ result in no information loss due to sampling.
An analog signal with bandwidth $f_a$ must be sampled at:

$$f_s > 2f_a$$

Perfect measurements taken at $f_s > 2f_a$ result in no information loss due to sampling.

Due to actual (quantized) measurements: oversampling is required.
A resolution of $N$ bits provides $2^N$ possible discrete output levels:

- Smallest distinguishable value $q$ (Least Significant Bit or LSB):
  $$q = \frac{1}{2^N}$$

- Ratio $\frac{1}{q}$ expressed in decibel (dB):
  $$10 \log 2^N = N \cdot 20 \log 2 \approx N \cdot 6.02 dB$$

(Decibel (dB) is a ration of powers defined as:
$$10 \log \frac{p_1}{p_0}$$
or by signal amplitudes:
$$10 \log \frac{A_1^2}{A_0^2}$$)
Quantization

The mean square error over one step:

\[ \overline{E^2} = \frac{1}{q} \int_{-q/2}^{q/2} E^2 dE = \frac{q^2}{12} \]

Root mean square (rms) noise voltage:

\[ \frac{q}{\sqrt{12}} \]
Quantization

The signal $S$ with respect to the rms noise:

$$\frac{S}{q} = S \cdot 2^N \sqrt{12}$$

or as the signal to noise ratio in decibel:

$$SNR[\text{dB}] = 10 \log \left( \frac{S^2}{q^2} \right)$$
Quantization

Assuming an ideal input signal:

\[ F(t) = A \sin \omega t \text{ with } q = \frac{2A}{2^N} \text{ and } S^2 = \frac{A^2}{2} \]

then the signal to noise ratio is:

\[ SNR[\text{dB}] = 10 \log \left( \frac{S^2}{q^2} \right) = 10 \log \left( \frac{3 \cdot 2^{2N}}{12} \right) \]

\[ SNR[\text{dB}] = 20N \log 2 + 10 \log \frac{3}{2} \]

\[ SNR[\text{dB}] \approx N \cdot 6.02 + 1.76 \]
Quantization

Determining the **effective number of bits** (ENOB):

\[
SNR_{\text{ideal}}[\text{dB}] = 20N \log 2 + 10 \log \frac{3}{2}
\]

\[
ENOB = \frac{SNR_{\text{actual}} - 10 \log \frac{3}{2}}{20 \log 2}
\]

\[
ENOB = \frac{SNR_{\text{actual}} - 1.76}{6.02}
\]

\[
ENOB = N \quad \text{for} \quad SNR_{\text{actual}} = SNR_{\text{ideal}}
\]
Quantization

Actual A/D converters are also characterized by:

- **Integral Non-Linearity (INL):** Maximal difference between the actual and ideal code centres.
- **Differential Non-Linearity (DNL):** Differences between successive code widths.
- **Missing codes:** reduce SNR by $20 \log_2 2$ or 6.02 dB for each missing code.
- **Response time / Latency, Throughput / Maximal sampling rate**
A/D converters

Some criteria to select the fitting A/D converter for an application:

- Throughput (maximal sampling frequency).
- Accuracy (ENOB, SNR).
- Latency (time from sensing to delivery).
- Power consumption.
- Complexity (also affects: price).

-trade-offs are to be expected:
  - Maximizing throughput will reduce accuracy and increase power consumption.
  - Maximizing accuracy will reduce throughput and increase latency. (… other trade-offs)
Integrating A/D converters (Single Slope)
Integrating A/D converters (Single Slope)

Integrate a reference voltage $U_{\text{ref}}$ until it matches the input voltage $A_{\text{IN}}$.

- **Sampling frequency** depends on input signal.
- **Accuracy** depends on $U_{\text{ref}}$, integrator and clock.
- **Simple** components.
- **Slow** (typically ~100 Hz)
Integrating A/D converters (Dual Slope)
Integrating A/D converters (Dual Slope)

Input voltage $A_{IN}$ is integrated for a constant time. The integrator is then discharged by a constant reference voltage $U_{ref}$. The discharge time is measured and is proportional to $A_{IN}$.

- Can **smooth** the input signal, and **suppress** specific frequencies.
Flash A/D converters

2
N
-1 reference voltages

\[ V_{\text{ref}} \times \frac{2^N - 1}{2^N} \]

\[ V_{\text{ref}} \times \frac{2^{N-2}}{2^N} \]

\[ \vdots \]

\[ V_{\text{ref}} \times \frac{2}{2^N} \]

\[ V_{\text{ref}} \times \frac{1}{2^N} \]

Comparators

Digital out

Level decoder

© 2013 Uwe R. Zimmer, The Australian National University
**Flash A/D converters**

$2^N - 1$ concurrent comparators identify the signal in one step.

- **Fastest converter technology**: Single step conversion, minimal latency.
- **Complex for higher resolution**: Required circuitry scales with $2^N - 1$.
- **Accuracy** depends on the accuracy of the reference voltages.

Typical applications: high speed, low resolution (e.g. video and radar converters).
Pipelined A/D converters
Pipelined A/D converters

$p$ pipeline stages with $m$ bits each provide a $pm$ bits converter. Each stage subtracts the analog value which has been converted (provides the rest as a residue value $A_{\text{RES}}$) and accumulates the digital output.

- **Keeps the throughput** (almost): All pipeline stages operate concurrently.
- **Trades resolution for latency**.
- **Accuracy** depends on components.

Typical applications:
- high resolution, high throughput, low cost.
Successive Approximation Register (SAR) converters
Successive Approximation Register (SAR) converters

Single bit A/D converter converts one bit at a time, starting with the most significant bit, e.g. comparing to $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots, \frac{1}{2^N}$ of full scale.

- **Minimal circuitry** – (almost) independent of resolution.
- **Typically slow**.
- **Accuracy** depends on the accuracy of the single bit ADC and the DAC.

Typical applications: typically slow, low budget applications – yet can also be used in high accuracy applications.
Tracking Register (TR) A/D converters
Tracking Register (TR) A/D converters

Continuous single bit conversion compares the current digital output with the analog input and counts a register up/down accordingly.

- **Minimal circuitry** (no S&H) – (almost) independent of resolution.
- **Speed** depends on amplitude changes in the input signal (no constant sampling frequency).
- **Accuracy** depends on the accuracy of the single bit ADC and the DAC.

Typical applications: Tracking slowly changing signals at high frequency. Rarely used today.
A/D, D/A & Interfaces

Σ-Δ A/D converters
\[ \Sigma - \Delta \text{ A/D converters} \]

\[ + U_{\text{ref}} \text{ or } - U_{\text{ref}} \text{ is subtracted from the input signal and integrated.} \]

The high frequent comparison of the integrator against ground results in a bitstream signal (which is also fed back).

The density of ‘1’s in the bitstream represents the input signal.

The bitstream can be deployed as such or be translated into digital words of varying lengths.
The sampling frequency of the bitstream with respect to the
digital output frequency (oversampling) determines accuracy.

- **Latency** depends on bit-stream frequency.
- **Accuracy** depends on number of bits decimated.
- Method is inherently **linear**.

Typical applications: High accuracy (typically 16-24 bits), moderate throughput (24 bit high quality audio converters are usually of this type).
Digital form converts to bitstream or to analog.

Typical application: Audio bitstreams.
Effective Number Of Bits (Signal to Noise Ratio) can be improved by adding further integrator stages.

- \( \Sigma - \Delta \) converters are not subject to aliasing, as they implicitly implement a low pass filter via the integrators.
Higher order $\Sigma$-$\Delta$ A/D converters

Dependency of the accuracy on the sampling frequency of the bitstream with respect to the output frequency (oversampling).

Theoretical/qualitative result only – achievable accuracy depends on more factors datasheets.
## A/D converters matrix

<table>
<thead>
<tr>
<th></th>
<th>Integrating</th>
<th>SAR</th>
<th>Σ-Δ</th>
<th>Pipeline Flash</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Throughput</strong></td>
<td>O(1/2^N)</td>
<td>O(1/N)</td>
<td>throughput/</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>latency vs. accuracy</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>O(2^N)</td>
<td>O(N)</td>
<td>typ. high accuracy</td>
<td>O(ρ)</td>
<td>O(1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Accuracy</strong></td>
<td>can be high</td>
<td>medium-high</td>
<td></td>
<td>typ. low-medium</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>typ. low</td>
<td></td>
</tr>
<tr>
<td><strong>Resolution</strong></td>
<td>typ. 8-16 bit</td>
<td>typ. 8-24 bit</td>
<td>typ. 16-24 bit</td>
<td>typ. 8-16 bit</td>
<td>typ. 4-8 bit</td>
</tr>
<tr>
<td><strong>Size / Components</strong></td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(ρ · 2^m)</td>
<td>O(2^N)</td>
</tr>
<tr>
<td><strong>Notes</strong></td>
<td>Can suppress some frequencies</td>
<td>Cost efficient and can be very accurate</td>
<td>Flexible architecture, typ. very accurate</td>
<td>Compromise between speed and cost</td>
<td>Fastest converter</td>
</tr>
</tbody>
</table>
ADC 08200
(National Semiconductors)
ADC 08200
(National Semiconductors)

typ. application: video processing – here: board by KNJN LLC, U.S.A. featuring four ADC 08200
ADC 08200 – Basic specifications

- **Resolution**: 8 bit
- **Sampling frequency**: 10 - 200 (230) MHz
- **Differential Non-Linearity (DNL)**: ±0.4 LSB (typical), ±0.95 LSB (max.)
- **ENOB**: 7.5 (at 4 MHz), 7.3 (at 50 MHz), 7.0 (at 100 MHz)
- **No missing codes**
- **Power consumption**: 1.05 mW/MSPS, 1 mW (power down)
- **Latency**: 6 cycles (pipeline delay)
- **Aperture** (sampling delay): 2.6 ns, with 2 ps rms jitter
ADC 08200 – Timing

- $V_{IN}$
- $t_{AD}$
- $t_{CL}$
- $t_{CH}$
- $V_{DR}/2$
- $t_{OD}$
- $t_{OH}$
- $90\%$
- $10\%$
ADC 08200 – Non-Linearities

Integral Non-Linearity

Differential Non-Linearity
Interfaces

A/D, D/A & Interfaces: Examples

LM12L458
(National Semiconductors)
A/D, D/A & Interfaces: Examples

**LM12L458 – Basic specifications**

- **Channels**: 8 (multiplexed).
- **Resolution**: 8 bit + sign or 12 bit + sign (SAR converter).
- **Sampling frequency**: max. 106 kHz.
- **Power consumption**: 15 mW; 6 μW (power down, clocked stopped).
- **Programmable** acquisition times, sequences and conversion rates.
- **32 word conversion FIFO buffer**.
- **Self-calibration and diagnostic** mode.
- **8 or 16 bit wide** data bus.

Typical applications: Data logging, process control, low power devices.
## Interfaces

### LM12L458 – Register bank

<table>
<thead>
<tr>
<th>A4 A3 A2 A1</th>
<th>Purpose</th>
<th>Type</th>
<th>D15</th>
<th>D14</th>
<th>D13</th>
<th>D12</th>
<th>D11</th>
<th>D10</th>
<th>D9</th>
<th>D8</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>Instruction RAM (RAM Pointer = 00)</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 to 1</td>
<td></td>
<td></td>
<td>Acquisition Time</td>
<td>Watch-</td>
<td></td>
<td></td>
<td></td>
<td>8/12</td>
<td>Timer</td>
<td>Sync</td>
<td>V&lt;sub&gt;IN−&lt;/sub&gt;</td>
<td>V&lt;sub&gt;IN+&lt;/sub&gt;</td>
<td>Pause</td>
<td>Loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td></td>
<td></td>
<td></td>
<td>dog</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>Instruction RAM (RAM Pointer = 01)</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 to 1</td>
<td></td>
<td></td>
<td>Instruction RAM</td>
<td>RAM</td>
<td></td>
<td></td>
<td></td>
<td>Don’t Care</td>
<td>&gt;/≤</td>
<td>Sign</td>
<td>Limit #1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td></td>
<td></td>
<td>RAM Pointer = 01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>Instruction RAM (RAM Pointer = 10)</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 to 1</td>
<td></td>
<td></td>
<td>Instruction RAM</td>
<td>RAM</td>
<td></td>
<td></td>
<td></td>
<td>Don’t Care</td>
<td>&gt;/≤</td>
<td>Sign</td>
<td>Limit #2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td></td>
<td></td>
<td>RAM Pointer = 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>Configuration Register</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 0</td>
<td></td>
<td></td>
<td>Configuration Register</td>
<td>RAM</td>
<td></td>
<td></td>
<td></td>
<td>Don’t Care</td>
<td>DIAG</td>
<td>Test</td>
<td>I/O</td>
<td>Auto</td>
<td>Chan</td>
<td>Stand-</td>
<td>Full</td>
<td>Auto-</td>
<td>Reset</td>
<td>Start</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>Interrupt Enable Register</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 0</td>
<td></td>
<td></td>
<td>Interrupt Enable Register</td>
<td>INT7</td>
<td></td>
<td></td>
<td></td>
<td>Number of Conversions in Conversion FIFO</td>
<td>Sequencer Address to Generate INT1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 0</td>
<td></td>
<td></td>
<td>Interrupt Status Register</td>
<td>INST7</td>
<td></td>
<td></td>
<td></td>
<td>“0”</td>
<td>INST5</td>
<td>INST4</td>
<td>INST3</td>
<td>INST2</td>
<td>INST1</td>
<td>INST0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>Timer Register</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 1</td>
<td></td>
<td></td>
<td>Timer Register</td>
<td>Timer</td>
<td>Preset High Byte</td>
<td>Timer</td>
<td>Preset Low Byte</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>Conversion FIFO</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>Limit Status Register</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: The table provides a summary of the interfaces and registers available in the LM12L458. Each row details the purpose, type, and bit configuration of a specific register or interface. The table includes columns for the address of the register, the purpose of the register, and various states or values that the register can take. The values are represented in hexadecimal notation, and the table highlights the specific functions and settings that can be controlled or monitored through these interfaces.
LM12L458 – Instructions

<table>
<thead>
<tr>
<th>A4 A3 A2 A1</th>
<th>Purpose</th>
<th>Type</th>
<th>D15</th>
<th>D14</th>
<th>D13</th>
<th>D12</th>
<th>D11</th>
<th>D10</th>
<th>D9</th>
<th>D8</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>Instruction RAM (RAM Pointer = 00)</td>
<td>R/W</td>
<td>Acquisition Time</td>
<td>Watch-dog</td>
<td>8/12</td>
<td>Timer</td>
<td>Sync</td>
<td>V_{IN-}</td>
<td>V_{IN+}</td>
<td>Pause</td>
<td>Loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>Instruction RAM (RAM Pointer = 01)</td>
<td>R/W</td>
<td>Don’t Care</td>
<td>Don’t Care</td>
<td>&gt;/&lt;=</td>
<td>Sign</td>
<td>Limit #1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>Instruction RAM (RAM Pointer = 10)</td>
<td>R/W</td>
<td>Don’t Care</td>
<td>Don’t Care</td>
<td>&gt;/&lt;=</td>
<td>Sign</td>
<td>Limit #2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Instruction RAM entries consist of:

- **Loop** (1 bit): indicates the last instruction and branches to the first one.
- **Pause** (1 bit): halts the sequencer before this instruction.
- **V_{IN+}, V_{IN-}** (2 · 3 bits): select the input channels (`000` selects ground in **V_{IN-}**).
- **Sync** (1 bit): wait for an external sync. signal before this instruction.
- **Timer** (1 bit): wait for a preset 16-bit counter delay before this instruction.
### Instruction RAM entries consist of (cont.):

- **8/12** (1 bit): selects the resolution (8 bit + sign or 12 bit + sign).
- **Watchdog** (1 bit): activates comparisons with two programmed limits.
- **Acquisition time** $D$ (4 bits): the converter takes $9 + 2D$ cycles (12 bit mode) or $2 + 2D$ cycles (8 bit mode) to sample the input. Reasonable times depend on the input resistance and clock frequency: $D \approx 0.45 \cdot R_S[k\Omega] \cdot f_{CLK}[MHz]$ for 12 bit conversions.
- **Limits** (2 · 9 bits, including sign and comparator): trigger levels for watchdog operation.

<table>
<thead>
<tr>
<th>A4 A3 A2 A1</th>
<th>Purpose</th>
<th>Type</th>
<th>D15</th>
<th>D14</th>
<th>D13</th>
<th>D12</th>
<th>D11</th>
<th>D10</th>
<th>D9</th>
<th>D8</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>Instruction RAM (RAM Pointer = 00)</td>
<td>R/W</td>
<td>Acquisition Time</td>
<td>Watchdog</td>
<td>8/12</td>
<td>Timer</td>
<td>Sync</td>
<td>$V_{IN-}$</td>
<td>$V_{IN+}$</td>
<td>Pause</td>
<td>Loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>Instruction RAM (RAM Pointer = 01)</td>
<td>R/W</td>
<td>Don’t Care</td>
<td></td>
<td>&lt;&gt;</td>
<td>Sign</td>
<td></td>
<td>Limit #1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>Instruction RAM (RAM Pointer = 10)</td>
<td>R/W</td>
<td>Don’t Care</td>
<td></td>
<td>&lt;&gt;</td>
<td>Sign</td>
<td></td>
<td>Limit #2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Interfaces

## LM12L458 – Instructions

<table>
<thead>
<tr>
<th>A4</th>
<th>A3</th>
<th>A2</th>
<th>A1</th>
<th>Purpose</th>
<th>Type</th>
<th>D15</th>
<th>D14</th>
<th>D13</th>
<th>D12</th>
<th>D11</th>
<th>D10</th>
<th>D9</th>
<th>D8</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Instruction RAM</td>
<td>R/W</td>
<td>Acquisition Time</td>
<td>Watchdog</td>
<td>8/12</td>
<td>Timer</td>
<td>Sync</td>
<td>V_in+</td>
<td>V_in+</td>
<td>Pause</td>
<td>Loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>to</td>
<td>1</td>
<td>1</td>
<td>(RAM Pointer = 00)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
type ChannelPlus is (Ch0, Ch1, Ch2, Ch3, Ch4, Ch5, Ch6, Ch7);

type ChannelMinus is (Gnd, Ch1, Ch2, Ch3, Ch4, Ch5, Ch6, Ch7);

type Resolutions is (TwelveBit, EightBit);

type Acquisition_D is new Natural range 0..15; -- 9+2D (12bit), 2+2D (8bit)

for ChannelPlus use (Ch0 => 0, Ch1 => 1, Ch2 => 2, Ch3 => 3,
                       Ch4 => 4, Ch5 => 5, Ch6 => 6, Ch7 => 7);

for ChannelMinus use (Gnd => 0, Ch1 => 1, Ch2 => 2, Ch3 => 3,
                       Ch4 => 4, Ch5 => 5, Ch6 => 6, Ch7 => 7);

for Resolutions use (TwelveBit => 0, EightBit => 1);

type Instruction is record
  EndOfLoop, Pause, Sync, Timer, Watchdog : Boolean;
  Vplus : ChannelPlus;
  Vminus : ChannelMinus;
  Resolution : Resolutions;
  AcquisitionTime : Acquisition_D;
end record;
```
### LM12L458 – Instructions

| A4 | A3 | A2 | A1 | Purpose | Type | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|----|----|----|----|---------|------|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|----|
| 0  | 0  | 0  | 0  | Instruction RAM | R/W  | Acquisition Time | Watchdog | 8/12 | Timer | Sync | V\_IN- | V\_IN+ | Pause | Loop |
| 0  | 0  | 1  | 1  | (RAM Pointer = 00) |       |                   |          |      |       |       |       |       |       |     |     |     |     |     |

Units\_Per\_Word : constant Integer := Word\_Size / Storage\_Unit;

for Instruction use record

- EndOfLoop at 0*Units\_Per\_Word range 0..0;
- Pause at 0*Units\_Per\_Word range 1..1;
- Vplus at 0*Units\_Per\_Word range 2..4;
- Vminus at 0*Units\_Per\_Word range 5..7;
- Sync at 0*Units\_Per\_Word range 8..8;
- Timer at 0*Units\_Per\_Word range 9..9;
- Resolution at 0*Units\_Per\_Word range 10..10;
- Watchdog at 0*Units\_Per\_Word range 11..11;
- AcquisitionTime at 0*Units\_Per\_Word range 12..15;

end record;
### LM12L458 – Instructions

<table>
<thead>
<tr>
<th>A4</th>
<th>A3</th>
<th>A2</th>
<th>A1</th>
<th><strong>Purpose</strong></th>
<th>Type</th>
<th>D15</th>
<th>D14</th>
<th>D13</th>
<th>D12</th>
<th>D11</th>
<th>D10</th>
<th>D9</th>
<th>D8</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Instruction RAM (RAM Pointer = 00)</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```haskell
for Instruction’Size use 16; -- Bits
for Instruction’Alignment use 2; -- Storage_Units (Bytes)
for Instruction’Bit_Order use High_Order_First;

type Instructions is array (0..7) of Instruction;
  pragma Pack (Instructions);
ADC_Instructions : Instructions;
for ADC_Instructions’Address use To_Address (16#0000132D#);
```
## Interfaces

### LM12L458 – Instructions

<table>
<thead>
<tr>
<th>A4</th>
<th>A3</th>
<th>A2</th>
<th>A1</th>
<th>Purpose</th>
<th>Type</th>
<th>D15</th>
<th>D14</th>
<th>D13</th>
<th>D12</th>
<th>D11</th>
<th>D10</th>
<th>D9</th>
<th>D8</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Instruction RAM</td>
<td>R/W</td>
<td>Acquisition Time</td>
<td>Watchdog</td>
<td>8/12</td>
<td>Timer</td>
<td>Sync</td>
<td>V_{IN+}</td>
<td>V_{IN-}</td>
<td>Pause</td>
<td>Loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>to</td>
<td></td>
<td></td>
<td>(RAM Pointer = 00)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ADC_Instructions (0) := (EndOfLoop => False,
Pause => False,
Vplus => Ch0,
Vminus => Gnd,
Sync => True,
Timer => False,
Resolution => EightBit,
Watchdog => False,
AquisitionTime => 10);

ADC_Instructions (1) := (EndOfLoop => True, -- last instruction
Pause => False,
Vplus => Ch1,
Vminus => Ch2,
Sync => False,
Timer => False,
Resolution => TwelveBit,
Watchdog => False,
AquisitionTime => 0);
Interfaces

LM12L458 – Instructions

| A4 | A3 | A2 | A1 | Purpose                                                                 | Type | D15 | D14 | D13 | D12 | D11 | D10 | D9  | D8  | D7  | D6  | D5  | D4  | D3  | D2  | D1  | D0  |
|----|----|----|----|-------------------------------------------------------------------------|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0  | 0  | 0  | 0  | Instruction RAM (RAM Pointer = 00)                                      | R/W  |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

ADC_Instructions (0) := (EndOfLoop => False,
Pause    => False,
Vplus    => Ch0,

AquisitionTime => 10);

ADC_Instructions (1) := (EndOfLoop => True, -- last instruction
Pause    => False,
Vplus    => Ch1,
Vminus   => Ch2,
Sync     => False,
Timer    => False,
Resolution => TwelveBit,
Watchdog => False,
AquisitionTime => 0);
### Interfaces

**LM12L458 – Instructions**

<table>
<thead>
<tr>
<th>A4</th>
<th>A3</th>
<th>A2</th>
<th>A1</th>
<th>Purpose</th>
<th>Type</th>
<th>D15</th>
<th>D14</th>
<th>D13</th>
<th>D12</th>
<th>D11</th>
<th>D10</th>
<th>D9</th>
<th>D8</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Instruction RAM</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>to</td>
<td>1</td>
<td>1</td>
<td>(RAM Pointer = 00)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td>Acquisition/Watchdog</td>
<td></td>
<td>8/12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>V_IN+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>V_IN-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Data structures in ‘C’:

```c
enum ChannelPlus  {Ch0=0, Ch1, Ch2, Ch3, Ch4, Ch5, Ch6, Ch7};
enum ChannelMinus {Gnd=0, Ch1, Ch2, Ch3, Ch4, Ch5, Ch6, Ch7};
enum Resolutions  {TwelveBit=0, EightBit};

struct {
    unsigned int EndOfLoop      : 1;
    unsigned int Pause          : 1;
    ChannelPlus  Vplus          : 3;
    ChannelMinus Vminus         : 3;
    unsigned int Sync           : 1;
    unsigned int Timer          : 1;
    Resolutions  Resolution     : 1;
    unsigned int Watchdog       : 1;
    unsigned int AquisitionTime : 4;
} Instruction;
```
### LM12L458 – Instructions

| A4 | A3 | A2 | A1 | Purpose                        | Type | D15 | D14 | D13 | D12 | D11 | D10 | D9  | D8  | D7  | D6  | D5  | D4  | D3  | D2  | D1  | D0  |
|----|----|----|----|-------------------------------|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0  | 0  | 0  | 0  | Instruction RAM              | R/W  |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0  | 0  | 0  | 1  | Instruction RAM (RAM Pointer = 00) | R/W  |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 1  | 1  | 1  | 1  |                                  |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

#### Data structures in ‘C’:

```c
InstructionsA[8];
InstructionsA *Instructions;
Instructions = 0x0000132D;

*Instructions (0).EndOfLoop = 0;
*Instructions (0).Pause     = 0;
*Instructions (0).Vplus     = Ch0;
*Instructions (0).Vminus    = Gnd;
*Instructions (0).Sync      = 1;
*Instructions (0).Timer     = 0;
*Instructions (0).Resolution = EightBit;
*Instructions (0).Watchdog  = 0;
*Instructions (0).AquisitionTime = 10;
```
**Interfaces**

**LM12L458 – Instructions**

| A4 A3 A2 A1 | Purpose | Type | D15 D14 D13 D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|-------------|---------|------|-----------------|-----|-----|----|----|----|----|----|----|----|----|----|----|----|
| 0 0 0 0     | Instruction RAM (RAM Pointer = 00) | R/W  | Acquisition Time | Watchdog | 8/12 | Timer | Sync | V_\text{IN_-} | V_\text{IN_+} | Pause | Loop |

**Data structures in ‘C’:**

Instruction InstructionsA[],
InstructionsA *Instructions;
Instructions = 0x0000132D;

*Instructions (0).InOfLoop = 0;
*Instructions (0).Pause = 0;
*Instructions (0).Vplus = Ch0;
*Instructions (0).Vminus = Gnd;
*Instructions (0).Sync = 1;
*Instructions (0).Timer = 0;
*Instructions (0).Resolution = EightBit;
*Instructions (0).Watchdog = 0;
*Instructions (0).AquisitionTime = 1;
### LM12L458 – Instructions

<table>
<thead>
<tr>
<th>A4</th>
<th>A3</th>
<th>A2</th>
<th>A1</th>
<th>Purpose</th>
<th>Type</th>
<th>D15</th>
<th>D14</th>
<th>D13</th>
<th>D12</th>
<th>D11</th>
<th>D10</th>
<th>D9</th>
<th>D8</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Instruction RAM (RAM Pointer = 00)</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0 to Time dog 8/12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In C: use macro-assembler style programming instead:

Read up on the local bit and byte ordering and set bits inside an int:

```c
unsigned int setbits (unsigned int *r,
                      unsigned int n,          /* set n bits */
                      unsigned int p,          /* at position p */
                      unsigned int x)          /* to bitstring x */
{
    unsigned int mask;
    mask = ~(~0 << n);
    *r &= ~(mask << p);
    *r |= (x & mask) << p;
    return (*r);
}
```
LM12L458 – Configuration register

| A4 | A3 | A2 | A1 | Purpose                 | Type | D15 | D14 | D13 | D12 | D11 | D10 | D9  | D8  | D7  | D6  | D5  | D4  | D3  | D2  | D1  | D0  |
|----|----|----|----|-------------------------|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 1  | 0  | 0  | 0  | Configuration Register  | R/W  |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|    |    |    |    | Don’t Care              |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|    |    |    |    | DIAG                    |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|    |    |    |    | Test = 0                |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|    |    |    |    | RAM Pointer             |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|    |    |    |    | I/O Sel                 |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|    |    |    |    | Auto Zero                |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|    |    |    |    | Chan Mask                |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|    |    |    |    | Stand-by                 |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|    |    |    |    | Full CAL                |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|    |    |    |    | Auto Zero                |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|    |    |    |    | Reset                    |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|    |    |    |    | Start                    |      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

Configuration register entries consist of:

- **Start** (1 bit): starts the sequencer.
- **Reset** (1 bit): sets the instruction pointer to ‘000’.
- **Auto-Zero** (1 bit): triggers a ‘short’ calibration (76 cycles – 1 offset sample).
- **Full-Cal** (1 bit): initiates a full calibration (4944 cycles, 8 samples) with interrupt.
- **Stand-by** (1 bit): disconnects the external clock and preserves the registers. After powering up again (~ 10ms): a specific interrupt is issued.
- **Chan-Mask** (1 bit): format selection for the FIFO output registers.
Configuration register entries consist of (cont.):

- **Auto-Zero**<sub>_Ec_</sub> (1 bit): auto-zeros the ADC automatically in every conversion.
- **I/O Sel** (1 bit): sets the Sync pin to input or output mode.
- **RAM Pointer** (2 bits): selects the current (16-bit) part in each 48-bit instruction.
- **Test=0** (1 bit): production testing mode: leave this bit at ‘0’.
- **DIAG** (1 bit): connects $V_{IN}^+$ and $V_{IN}^-$ to $V_{REF}^+$ and $V_{REF}^-$ for testing purposes.
LM12L458 – Sequencer

IP := 0;

loop
    repeat
        if Auto_Zero or Full_Cal then Calibrate;
    until Start_bit;

if Instr (IP).Timer then Run_Timer;

Current_Signal := Aquisition (Instr (IP).Aquisition_Time);

if Instr (IP).Watchdog then begin
    if Instr (IP).Sync then Wait_for_external_sync;
    Compare (Current, Instr (IP).Limit_1);
    if Instr (IP).Sync then Wait_for_external_sync;
    Compare (Current, Instr (IP).Limit_2);
else
    if Instr (IP).Sync then Wait_for_external_sync;
    Convert_and_store_in_FIFO (Current_Signal);
end;

if Instr (IP).Loop then IP := 0;
else IP := IP + 1;

end loop;
Conversion FIFO buffer is a read-only register:

- Every read on this address will delete this result from the internal memory and will shift the next result into the visible conversion FIFO register.
- The FIFO holds up to 32 conversion results.
- Data will be lost, if the results are not read fast enough to prevent a buffer overrun.

The controller can issue specific interrupts or initiate a DMA transfer, when a given number of results are accumulated or a certain instruction is completed.
## Interrupt Enable R/W Number of Conversions Sequencer INT7 Don't INT5 INT4 INT3 INT2 INT1 INT0

### Interrupt Status Register

<table>
<thead>
<tr>
<th>A4</th>
<th>A3</th>
<th>A2</th>
<th>A1</th>
<th>Purpose</th>
<th>Type</th>
<th>D15</th>
<th>D14</th>
<th>D13</th>
<th>D12</th>
<th>D11</th>
<th>D10</th>
<th>D9</th>
<th>D8</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Interrupt Enable Register</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Actual Number of Conversion Results</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Interrupts:

- Can be associated by an existing instruction and/or triggered by a specific number of conversion results.
- The interrupt can be handled by another controller, DMA mechanism, or the actual CPU.

### Polling:

- In dedicated, high integrity, or low latency controller setups the client controller/CPU will poll rather than wait for interrupts.

---

© 2013 Uwe R. Zimmer, The Australian National University
Interfaces

A/D, D/A & Interfaces: Examples

LM12L458
(National Semiconductors)
Micro-controllers

Micro-controller definition

Short: “Computer system on a chip”

Typical elements found in a micro-controller include:

- CPU (typ. 4-64 bit word size) – also as multi-cores.
- Memory (typ. a few hundred Bytes to many MBytes) as RAM, ROM, EPROM and/or Flash.
- Clock generator.
- Timers and general interrupt logic.
- Basic I/O (often as multiple, partly autonomous I/O units):
  - General purpose digital I/O lines – often combined with PWM generators, signal width detectors, counters, watchdog- or timer-triggers.
  - A/D and D/A converters (typ. 6-12 bit).
- Higher level I/O channels: Ethernet, UART, I\(^2\)C, Fast serial, Can-bus, …
Micro-controller examples

**MC68HC05**

- **Port A**
  - PA0
  - PA1
  - PA2
  - PA3
  - PA4
  - PA5
  - PA6
  - PA7

- **Port B**
  - PB0
  - PB1
  - PB2
  - PB3
  - PB4
  - PB5
  - PB6
  - PB7

- **Port C**
  - PC0
  - PC1
  - PC2/ECLK
  - PC3
  - PC4
  - PC5
  - PC6
  - PC7

- **Port D**
  - PD0/AN0
  - PD1/AN1
  - PD2/AN2
  - PD3/AN3
  - PD4/AN4
  - PD5/AN5
  - PD6/AN6
  - PD7/AN7

- **振荡器**
  - OSC1
  - OSC2

- **MC68HC05 CPU**
  - 256 bytes EEPROM
  - COP watchdog
  - Oscillator
  - \( \div 2 \text{ or } \div 32 \)

- **5950 bytes User ROM** (including 14 bytes User vectors)

- **432 bytes self check ROM**

- **176 bytes RAM**

- **16-bit programmable timer**

- **M68HC05L8**

- **8-bit A/D converter**

- **VDD**
  - **VSS**
  - **OSC1**
  - **OSC2**
  - **M68HC05**
  - **COP watchdog**
  - **Charge pump**
  - **16-bit programmable timer**
  - **A/D converter**
  - **PLM**
  - **Charge pump**
  - **COP watchdog**
  - **Oscillator**
  - **\( \div 2 \text{ or } \div 32 \)**
  - **M68HC05 CPU**
  - **5950 bytes User ROM** (including 14 bytes User vectors)
  - **432 bytes self check ROM**
  - **176 bytes RAM**
  - **16-bit programmable timer**
  - **A/D converter**
  - **VDD**
  - **VSS**
  - **OSC1**
  - **OSC2**
  - **M68HC05**
  - **COP watchdog**
  - **Charge pump**
  - **16-bit programmable timer**
  - **A/D converter**
  - **VDD**
  - **VSS**
  - **OSC1**
  - **OSC2**
  - **M68HC05**
  - **COP watchdog**
  - **Charge pump**
  - **16-bit programmable timer**
  - **A/D converter**
  - **VDD**
  - **VSS**
  - **OSC1**
  - **OSC2**
  - **M68HC05**
  - **COP watchdog**
  - **Charge pump**
  - **16-bit programmable timer**
  - **A/D converter**

© 2013 Uwe R. Zimmer, The Australian National University
Micro-controller examples

**MC68HC05**

- **Clock:** max. 2.1 MHz internal (4.2 MHz external).
- **RAM:** 176 bytes.
- **ROM:** 5936 bytes.
- **EEPROM:** 256 bytes.
- **Power** saving modes (stop, wait, slow).
- **Serial:** 46-76800 baud (at 2.4576 MHz)
  79-131072 baud (at 4.194394 MHz).
- **Parallel** I/O: 3 · 8 bit; Parallel in: 1 · 8 bit.
- **Timers:** 1 · 16 bit.
- **A/D:** 8 channels, 8 bit.
- **PWM:** 2 generators.
MAIN BRCLR 6,TSR,MAIN ;Loop here till Output Compare flag set
LDA OCMP+1 ;Low byte of Output Compare register
ADD #$D4 ;Add
STA TEMPA ;Save till high half calculated
LDA OCMP ;High byte of Output Compare register
ADC #$30 ;Add (+carry)
STA OCMP ;Update high byte of Output Compare register
LDA TEMPA ;Get low half of updated value
STA OCMP+1 ;Update low half and reset Output Compare flag
LDA TIC ;Get current TIC value
INCA ;TIC := TIC + 1
STA TIC ;Update TIC
CMP #20 ;20th TIC?, 1 second passed?
BLO NOSEC ;If not, skip next clear
CLR TIC ;Clear TIC on 20th

NOSEC EQU *
JSR TIME ;Update time-of-day & day-of-week
JSR KYPAD ;Check/service keypad
JSR A2D ;Check Temp Sensors
JSR HVAC ;Update Heat/Air Cond Outputs
JSR LCD ;Update LCD display
BRA MAIN ;End of main loop
Micro-controller examples

AVR32
Micro-controller examples

AVR32

- CPU: 32 bit RISC with DSP extensions.
- Clock: 66 MHz.
- Memory: up to 32 kB SRAM, up to 512 kB Flash
- Separate DMA bus for peripherals.
- Power: up to 120 mW (132 µW in sleep)
- Nexus debug port.
- up to 113 GPIO with up to 7 PWM channels.
- 1 · 32 bit real time counter.
- 6 · 16 bit timers
- 1 · Ethernet, 1 · SSC, 4 · UART (incl. SPI)
- 8 · 10 bit ADC (SAR).
- 2 · DDC bitstream output (Σ-Δ).
Alternative Processor Architectures: Parallax Propeller

© 2013 Uwe R. Zimmer, The Australian National University

- 8 cores with 2 kB local memory
- Low cost 32 bit processor ($8)
- 40 kB shared memory
- 8 semaphores
- No interrupts!
Micro-controller examples

MPC565
**Micro-controller examples**

**MPC565**

- **Power**: power dissipation: 0.8 - 1.12 W, -40° - +125°C
- **CPU**: PowerPC core (incl. FPU & BBC), 56 MHz
- **RAM**: flash: 1 MB, static: 36 kB
- **Time processing units**: 3 (via dual-ported RAM)
- **Timers**: 22 channels (PWM & RTC supported)
- **A/D convertors**: 40 channels, 10 bit, 250 kHz
- **Can-bus**: 3 TOUCAN modules
- **Serial**: 2 interfaces
- **Data link controller**: SAE J1850 class B communications module
- **Real-time embedded application development interface**: NEXUS debug port (IEEE-ISTO 5001-1999)
- **Packing**: 352/388 ball PBGA
Micro-controller examples

Time Processing Unit

A special-purpose micro-controller:

- Independent µ-engine.
- 16 digital I/O channels with independent match and capture capabilities.
- Meant to operate these I/O channels for timing control purposes.
- Predefined µ-engine command set (ROM functions in control store).
- 2 · 16 bit time bases
**Micro-controller examples**

*Time Processing Unit – Some predefined µ-engine functions*

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Period- / Pulse-width accumulator</strong></td>
<td>The period/pulse-width accumulator (PPWA) algorithm accumulates a 16 bit or 24 bit sum of either the period or the pulse width of an input signal over a programmable number of periods or pulses (from 1 to 255).</td>
</tr>
<tr>
<td><strong>Stepper motor</strong></td>
<td>The stepper motor (SM) control algorithm provides for linear acceleration and deceleration control of a stepper motor with a programmable number of step rates of up to 14.</td>
</tr>
<tr>
<td><strong>Position-synchronized pulse generator</strong></td>
<td>The PSP function generates pulses of variable length at specified “angles.” Angle clock period is measured (in TCR1 clocks) using the PMA/PMM function on another channel.</td>
</tr>
<tr>
<td><strong>Period measurement</strong></td>
<td>This function measures the period (in TCR1 clocks) between regularly occurring input transitions and makes this period available for use by other functions or by the CPU (optional detection of misses and additional transitions).</td>
</tr>
<tr>
<td><strong>Pulse-width modulation</strong></td>
<td>The TPU can generate a pulse-width modulation (PWM) waveform with any duty cycle from zero to 100% (within the resolution and latency capability of the TPU).</td>
</tr>
<tr>
<td><strong>Synchronized pulse-width modulation</strong></td>
<td>Three different operating modes allow the function to maintain complex timing relationships between channels without CPU intervention.</td>
</tr>
<tr>
<td><strong>Quadruple decode</strong></td>
<td>QDEC uses two channels to decode a pair of out-of-phase signals in order to present the CPU with directional information and a position value.</td>
</tr>
</tbody>
</table>
Micro-controller examples

Time Processing Unit – Emulation Mode

Create your own µ-engine

Refer the control store of the µ-engine to the dual-ported RAM instead of the integrated ROM area and supply:

- Up to 16 µ-engine commands (functions).
- In 2-8 kB of long-word (32 bit) organized memory.
- Programmed in a 32 bit µ-instruction format (explained next).

The dual-ported RAM is then cut off from the CPU (the TPU parameter RAM is not affected).
Micro-controller examples
Time Processing Unit – μinstructions formats:

1: Execution unit and RAM:

```
<table>
<thead>
<tr>
<th>31</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW</td>
<td>T1ABS</td>
<td>T3ABD</td>
<td>SHF</td>
<td>SRC</td>
<td>CCL</td>
<td>T1BBS</td>
<td>BI</td>
</tr>
</tbody>
</table>
```

2: Execution unit, flag, and channel control:

```
<table>
<thead>
<tr>
<th>31</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW</td>
<td>T1ABS</td>
<td>T3ABD</td>
<td>SHF</td>
<td>TDL</td>
<td>MRL</td>
<td>T1BBS</td>
<td>BI</td>
</tr>
</tbody>
</table>
```

3: Conditional branch, flag, and channel control:

```
<table>
<thead>
<tr>
<th>31</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCC</td>
<td>BAF (8:0)</td>
<td>TBS</td>
<td>PAC</td>
<td>BCF</td>
<td>PSC</td>
<td>FLC</td>
<td>CIR</td>
</tr>
</tbody>
</table>
```

4: Jump, flag, and RAM:

```
<table>
<thead>
<tr>
<th>31</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW</td>
<td>NMA</td>
<td>BAF (8:0)</td>
<td>FLC</td>
<td>IOM</td>
<td>AIS (6:0)</td>
<td>DEC/END</td>
<td></td>
</tr>
</tbody>
</table>
```

5: Execution unit, immediate, and flag:

```
<table>
<thead>
<tr>
<th>31</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1ABS</td>
<td>T3ABD</td>
<td>SHF</td>
<td>SRC</td>
<td>CCL</td>
<td>IMMEDIATE DATA (7:0)</td>
<td>DEC/END</td>
<td></td>
</tr>
</tbody>
</table>
```
Micro-controller examples

Time Processing Unit – μinstructions formats:

1: Execution unit and RAM:

<table>
<thead>
<tr>
<th>RW</th>
<th>RAM</th>
<th>Read/Write Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1ABS</td>
<td>T1</td>
<td>A-Bus Source Control</td>
</tr>
<tr>
<td>T3ABD</td>
<td>T3</td>
<td>A-Bus Destination Control</td>
</tr>
<tr>
<td>SHF</td>
<td>AU</td>
<td>Shifter Control</td>
</tr>
<tr>
<td>SRC</td>
<td>AU</td>
<td>Shift Register Control</td>
</tr>
<tr>
<td>CCL</td>
<td>AU</td>
<td>Condition Code Latch Control</td>
</tr>
<tr>
<td>T1BBS</td>
<td>T1</td>
<td>B-Bus Source Control</td>
</tr>
<tr>
<td>CIN</td>
<td>AU</td>
<td>B-Bus Carry Control</td>
</tr>
<tr>
<td>BINV</td>
<td>AU</td>
<td>B-Bus Invert Control</td>
</tr>
<tr>
<td>IOM</td>
<td>RAM</td>
<td>Input/Output Mode Control</td>
</tr>
<tr>
<td>AIS</td>
<td>RAM</td>
<td>Address</td>
</tr>
<tr>
<td>DEC/END</td>
<td>SEQ</td>
<td>Decrementor / End Control</td>
</tr>
</tbody>
</table>

Operation groups:

- Execution unit
- Channel control
- RAM
- Sequencer
Interfaces

**Micro-controller examples**

*Time Processing Unit – µinstructions formats:*

2: Execution unit, flag, and channel control:

<table>
<thead>
<tr>
<th>Index</th>
<th>ERW</th>
<th>RAM</th>
<th>T1ABS</th>
<th>T3ABD</th>
<th>SHF</th>
<th>TDL</th>
<th>MRL</th>
<th>T1BBS</th>
<th>CIN</th>
<th>BINV</th>
<th>PAC</th>
<th>LLS</th>
<th>PSC</th>
<th>FLC</th>
<th>CIR</th>
<th>DEC/END</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/1</td>
<td>RW</td>
<td></td>
<td>T1</td>
<td>T3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2/3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4/5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8/9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12/13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14/15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16/17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18/19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20/21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22/23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24/25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26/27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28/29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30/31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Operation groups:**

- **Execution unit**
- **Channel control**
- **RAM**
- **Sequencer**

Event Register Write Control
A-Bus Source Control
A-Bus Destination Control
Shifter Control
Transition Detect Latch Negation Control
Match Recognition Latch Negation Control
B-Bus Source Control
B-Bus Carry Control
B-Bus Invert Control
Pin Action Control
Link Service Latch Negation Control
Pin State Control
Flag Control
Channel Interrupt Request
Decrementor / End Control
Micro-controller examples

Time Processing Unit – μinstructions formats:

3: Conditional branch, flag, and channel control:

<table>
<thead>
<tr>
<th>Operation groups:</th>
<th>Execution unit</th>
<th>Channel control</th>
<th>RAM</th>
<th>Sequencer</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCC</td>
<td>SEQ</td>
<td>Branch Condition Code Field</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLS</td>
<td>SEQ</td>
<td>μPC Flush Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BAF</td>
<td>SEQ</td>
<td>Branch Address Field</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBS</td>
<td>CC</td>
<td>Time Base Select Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAC</td>
<td>CC</td>
<td>Pin Action Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCF</td>
<td>SEQ</td>
<td>Branch Condition Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSC</td>
<td>CC</td>
<td>Pin State Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLC</td>
<td>CC</td>
<td>Flag Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIR</td>
<td>CC</td>
<td>Channel Interrupt Request</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTR</td>
<td>CC</td>
<td>Match/Transition Detect Service Request Inhibit Control</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>31</th>
<th>24 23</th>
<th>16 15</th>
<th>8 7 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0</td>
<td>BCC</td>
<td>FLS</td>
<td>BAF (8:0)</td>
</tr>
<tr>
<td></td>
<td>TBS</td>
<td>PAC</td>
<td>BCF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PSC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FLC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CIR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MTSR</td>
</tr>
</tbody>
</table>
Micro-controller examples

Time Processing Unit – μinstructions formats:

4: Jump, flag, and RAM:

<table>
<thead>
<tr>
<th>RW</th>
<th>RAM</th>
<th>Read/Write Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMA</td>
<td>SEQ</td>
<td>Next μPC Address Mode Control</td>
</tr>
<tr>
<td>FLS</td>
<td>SEQ</td>
<td>μPC Flush Control</td>
</tr>
<tr>
<td>BAF</td>
<td>SEQ</td>
<td>Branch Address Field</td>
</tr>
<tr>
<td>FLC</td>
<td>CC</td>
<td>Flag Control</td>
</tr>
<tr>
<td>LSL</td>
<td>CC</td>
<td>Link Service Latch Negation Control</td>
</tr>
<tr>
<td>IOM</td>
<td>RAM</td>
<td>Input/Output Mode Control</td>
</tr>
<tr>
<td>AIS</td>
<td>RAM</td>
<td>Address</td>
</tr>
<tr>
<td>DEC/END</td>
<td>SEQ</td>
<td>Decrementor / End Control</td>
</tr>
</tbody>
</table>

Operation groups:

- Execution unit
- Channel control
- RAM
- Sequencer

© 2013 Uwe R. Zimmer, The Australian National University
### Micro-controller examples

**Time Processing Unit – µinstructions formats:**

5: Execution unit, immediate, and flag:

```
 31  30  29  28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0
```

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>T1ABS</td>
</tr>
<tr>
<td>30</td>
<td>T3ABD</td>
</tr>
<tr>
<td>29</td>
<td>SHF</td>
</tr>
<tr>
<td>28</td>
<td>SRC</td>
</tr>
<tr>
<td>27</td>
<td>CCL</td>
</tr>
<tr>
<td>26</td>
<td>IMMEDIATE DATA (T1BBI)</td>
</tr>
<tr>
<td>25</td>
<td>LSL</td>
</tr>
<tr>
<td>24</td>
<td>EQ/GE</td>
</tr>
<tr>
<td>23</td>
<td>FLC</td>
</tr>
<tr>
<td>22</td>
<td>CIR</td>
</tr>
<tr>
<td>21</td>
<td>DEC/END</td>
</tr>
</tbody>
</table>

#### Operation groups:

- **Execution unit**
- **Channel control**
- **RAM**
- **Sequencer**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1ABS</td>
<td>T1 A-Bus Source Control</td>
</tr>
<tr>
<td>T3ABD</td>
<td>T3 A-Bus Destination Control</td>
</tr>
<tr>
<td>SHF</td>
<td>AU Shifter Control</td>
</tr>
<tr>
<td>SRC</td>
<td>Shift Register Control</td>
</tr>
<tr>
<td>CCL</td>
<td>AU Condition Code Latch Control</td>
</tr>
<tr>
<td>T1BBI</td>
<td>T1 B-Bus Immediate Data</td>
</tr>
<tr>
<td>LSL</td>
<td>CC Link Service Latch Negation Control</td>
</tr>
<tr>
<td>EQ/GE</td>
<td>CC Match Compare Register Control</td>
</tr>
<tr>
<td>FLC</td>
<td>CC Flag Control</td>
</tr>
<tr>
<td>CIR</td>
<td>CC Channel Interrupt Request</td>
</tr>
<tr>
<td>DEC/END</td>
<td>SEQ Decrementor / End Control</td>
</tr>
</tbody>
</table>
**Micro-controller examples**

*Time Processing Unit – μinstructions formats:*

1: Execution unit and RAM:

2: Execution unit, flag, and channel control:

3: Conditional branch, flag, and channel control:

4: Jump, flag, and RAM:

5: Execution unit, immediate, and flag:

---

© 2013 Uwe R. Zimmer, The Australian National University
Micro-controller examples

Time Processing Unit – Example Code

One state of a function (example):

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>24</td>
<td>23</td>
<td>16</td>
<td>15</td>
<td>8</td>
<td>7</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>R</td>
<td>T1ABS</td>
<td>T3ABD</td>
<td>SHF</td>
<td>SRCL</td>
<td>T1BBS</td>
<td>CIN</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>R</td>
<td>T1ABS</td>
<td>T3ABD</td>
<td>SHF</td>
<td>TDRL</td>
<td>T1BBS</td>
<td>CIN</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>R</td>
<td>T1ABS</td>
<td>T3ABD</td>
<td>SHF</td>
<td>SRCL</td>
<td>IMMEDIATE DATA (T1BBI)</td>
<td>CIN</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>R</td>
<td>NMA</td>
<td>FLS</td>
<td>BAF (8:0)</td>
<td>FLC</td>
<td>LSL</td>
<td>AIS (6:0)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>R</td>
<td>T1ABS</td>
<td>T3ABD</td>
<td>SHF</td>
<td>TDRL</td>
<td>T1BBS</td>
<td>CIN</td>
</tr>
</tbody>
</table>

- Proceed to next µ-instruction
- Decrement register – Proceed at ‘0’
- Decrement register – Call at ‘0’
- End of state

Functions are composed of one or multiple states.
States are *atomic* instruction blocks, i.e. the scheduler will not interrupt.
Micro-controller examples

Composing the µengine:

Entities to consider:

- **States**: non-interruptible µ-code-blocks.
- **Functions**: constructed of one or multiple states.
- **Channels**: 16 digital I/O lines with match and capture.
- **Priorities** of channels.
- **Timers**: 2 · 16 bit time-bases.

Associate functions, time-bases, channels and priorities ... and let it run!
**Micro-controller examples**

**TPU Fixed scheduled, prioritized time slots**

Round Robin schedule for all runnable states inside each priority.

- No state will be starved.
Micro-controller examples

**TPU Fixed scheduled, prioritized time slots**

Unused slots will be re-assigned according to priorities and channel numbers.
Micro-controller examples

TPU Fixed scheduled, prioritized time slots

States have different and variable lengths.

Calculating actual and maximal latencies requires full understanding of all states.
Micro-controller examples

Latencies on TPUs

... for latencies of capture and match at each channel mind:

- only the time-base resolution (all channels are evaluated independently and in parallel).

... for the functions associated with individual channels mind the:

- number of active channels (max. 16).
- number of channels on each priority level (add max. 2 µ-cycles for each “state-switch”).
- number of available time slots on each priority level per full scheduler-cycle (4, 2, 1 slot{[s]}).
- number of µ-cycles to execute individual states of a function (2 µ-cycles per µ-instruction).
- number of RAM accesses during the execution of a state (each access may stall for 2 CPU-cycles).
- TPU clock cycle frequency.
Micro-controller examples

Determining actual TPU latencies

Emulate the known executing ready times for all states.

Add two 2 µ-cycles for each state switch.
**Micro-controller examples**

**Determining actual TPU latencies**

Emulate the known memory access patterns for all states.

Add two 2 µ-cycles for each memory access (potential stall times).
**Micro-controller examples**

**Determining maximal TPU latencies**

Assume all states on the same priority runnable at all times and run their maximal lengths.

Assume the longest state out of all higher priorities runnable at all times.

Assume the longest states on each lower priority runnable at all times.

Deploy a set out of those states which will cause the longest latency.

Determine the longest latency inside a full hyper-cycle.
Micro-controller examples

Determining maximal TPU latencies

Assume all states on the same priority runnable at all times and run their maximal lengths.

Assume the longest state out of all higher priorities runnable at all times.

Assume the longest states on each lower priority runnable at all times. Deploy a set out of those states which will cause the longest latency.

Determine the longest latency inside a full hyper-cycle.
### Micro-controller examples

#### Determining maximal TPU latencies

- Assume all states on the same priority runnable at all times and run their maximal lengths.
- Assume the longest state out of all higher priorities runnable at all times.
- Assume the longest states on each lower priority runnable at all times.
  Deploy a set out of those states which will cause the longest latency.
- Determine the longest latency inside a full hyper-cycle.
Micro-controller examples

Time Processing Unit

A special-purpose micro-controller:

- Independent µ-engine.
- 16 digital I/O channels with independent match and capture capabilities.
- Meant to operate these I/O channels for timing control purposes.
- Predefined µ-engine command set (ROM functions in control store).
- 2 · 16 bit time bases
Micro-controller examples

*MPC565: Time-base & Real-time clock*

- **64 bit time base**
  (driven by an external clock: e.g. 20 MHz resolution: 50 ns; range: ~30,000 years).

- **Free running**
  (not influenced by any CPU action or resets).

- **2 reference registers**
  (used for compares and interrupt generation).

- **Real-Time clock**
  supplies full seconds (32 bit range: ~136 years)
  (not affected by CPU, resets, and operates in all low-power modes).
Micro-controller examples

**MPC565 : Interrupt controller**

- Handles up to 48 different sources
  (32 from internal modules, 8 from timers and clocks, and 8 external vectorized sources) and supply each of them with a unique interrupt-vector

- **8 interrupt levels** are distinguished by the interrupt controller
  (32 interrupt levels are supplied by the internal modules, prioritized and vectorized interrupts are supplied by external sources)

- **Latency: 20 clock cycles**
  + bus collisions + CPU state saving + tasking system overhead
Micro-controller examples

MPC565: Nexus debug port (IEEE-ISTO 5001-1999)
(Real-time embedded application development interface)

On-Line mode:

- **Program trace**: via branch trace messaging.
- **Data trace**: via data write messaging and data read messaging (can be reduced to selected areas).
- **Owner trace**: via ownership trace messaging (also indicates task creation and activation).
- **Run-time access** to memory map and special CPU registers.
- **Watchpoints**: CPU watchdog status signals are snooped and transferred with high priority.

Off-line mode:

- **Read / Write access**: the READI module can take over the L-bus to manipulate data.
- **Access to all CPU registers** during halt.
**Micro-controller examples**

**MPC565**

- **Power**: power dissipation: 0.8 - 1.12 W, -40° - +125°C
- **CPU**: PowerPC core (incl. FPU & BBC), 56 MHz
- **RAM**: flash: 1 MB, static: 36 kB
- **Time processing units**: 3 (via dual-ported RAM)
- **Timers**: 22 channels (PWM & RTC supported)
- **A/D convertors**: 40 channels, 10 bit, 250 kHz
- **Can-bus**: 3 TOUCAN modules
- **Serial**: 2 interfaces
- **Data link controller**: SAE J1850 class B communications module
- **Real-time embedded application development interface**: NEXUS debug port (IEEE-ISTO 5001-1999)
- **Packing**: 352/388 ball PBGA
Interface architectures

Basic sampling control mechanisms

- **Status driven**: the computer polls for information (used in dedicated micro-controllers and pre-scheduled hard real-time environments).

- **Interrupt driven**: The data generating device may issue an interrupt when new data had been detected / converted or when internal buffers are full.

- **Program controlled**: The interrupts are handled by the CPU directly (by changing tasks, calling a procedure, raising an exception, free tasks on a semaphore, sending a message to a task, …).

- **Program initiated**: The interrupts are handled by a DMA-controller. No processing is performed. Depending on the DMA setup, cycle stealing can occur and needs to be considered for the worst case computing times.

- **Channel program controlled**: The interrupts are handled by a dedicated channel device. The data is transferred and processed. Optional memory-based communication with the CPU. The channel controller is usually itself a dedicated µ-engine / -controller.
Interface architectures

Handling device responses

Responses from devices can be:

- Immediate.
- With a constant delay or within a defined time-frame.
- Unpredictable / sporadic.

Device handlers may thus:

- Perform a ‘busy-wait’ for the response.
- Reschedule the device-process by a constant delay.
- Schedule the device-process periodically and employ different time-slots for sending control / data and receiving status / data.
- React to triggers / calls / interrupts from the device.

The device handler can be implemented as a process / interrupt routine / dedicated μ-controller / DMA-controller or a mixture of those.
Interface architectures

Handling device responses

How to embed the unpredictable in predictable systems?

By providing the resources to cope with the assumed worst case (and fall back to a lower, yet safe functionality beyond that).

Concrete:

- Either the unpredictable events need to be synchronized with the remaining real-time tasks without violating real-time constraints.

- Or exclusive processing resources (e.g. a dedicated micro-controller) for a specific device need to be provided.
Interface architectures

Language requirements for interfaces

- Specify the device interface (protocol and formats) in all detail
  (candidates: Ada95, CHILL, ERLANG, Modula-2, …
  …or Macro-Assemblers level (if platform-independence or abstraction is not required)).

- Handling asynchronous hardware messages (devices, timers, …)
  Many different methods to implement a context-switch
  (candidates: all languages with some real-time orientation:
  PEARL, CHILL, ERLANG, Ada95, RT-Java, POSIX, …)

- The term “high-level languages” in the real-time interface context:
  Allow for abstractions while being specific down to the actual level of interface realities
Summary

Converters & Interfaces

• Analogue signal chain in a digital system
  • Sampling data, aliasing, Nyquist’s criterion, oversampling
  • Quantization (LSB, rms noise voltage, SNR, ENOB), Missing codes, DNL, INL

• A/D converters:
  • Integrating (Single- / Dual-slope), Flash, Pipelined, SAR, Tracking, Σ-Δ, Σ-Δ DDA, n-th order Σ-Δ.

• Examples:
  • Fast and simple A/D converter example: National Semiconductor ADC08200
  • Multi-channel A/D data logging interface example: National Semiconductor LM12L458
  • Simple 8-bit µ-controller example: Motorola MC68HC05, Propeller.
  • Complex 32-bit µ-controller examples: AVR32 and Motorola MPC565 (including TPUs).

• General device handling / sampling control / language requirements