Examples of Embedded Computer Systems

- automotive
- communications

Examples of General-Purpose Computer Systems

- medical
- appliances
- consumer electronics
- networking
- other

Embedded Computer Systems

- An *embedded computer system* is a system that includes a microcomputer configured to perform a dedicated application.
- Software is typically fixed into ROM and not user accessible.
- Microcomputer is embedded, or hidden, inside the device.
- Typical automobile contains an average of 10 microcomputers.
- Upscale homes may have as many as 150 microcomputers.
- Average consumer interacts with µ-controllers 300 times/day.
Real-Time Interfacing

- Microcomputers accept inputs, perform calculations, and generate outputs.
- **Real-time systems** have an upper bound on the time required to perform the input/calculation/output sequence.
- Must learn features built into microcomputers to handle time.
- An interface is the hardware and software that allow the computer to communicate with the external hardware.
- Must learn how to interface many types of inputs and outputs in both digital and analog form.

Chip Peripherals and Timer Features

- Microcontrollers support many I/O features:
  - Synchronous Serial Peripheral Interface (SPI)
  - Asynchronous Serial Communication Interface (SCI)
  - Analog-to-digital (ADC) converters
  - Fixed period rate interrupts
  - Computer Operating Properly (COP) protection
  - Pulse accumulator for external event counting
  - Pulse-width-modulations (PWM) outputs
  - Event counter system for advanced timer operations
  - Input capture used for period and pulse width measurement
  - Output capture used for generating signals and frequency measurement

Microcomputer Architectures

<table>
<thead>
<tr>
<th>Company</th>
<th>Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motorola</td>
<td>68HC05, 68HC08, <strong>68HC11</strong>, 68HC12, 68HC16, 68K, MCORE, Coldfire, PowerPC</td>
</tr>
<tr>
<td>Intel</td>
<td>8051, 80251, 8096, 80296</td>
</tr>
<tr>
<td>Philips</td>
<td>8051</td>
</tr>
<tr>
<td>Hitachi</td>
<td>H8</td>
</tr>
<tr>
<td>NEC</td>
<td>78K</td>
</tr>
<tr>
<td>Mitsubishi</td>
<td>740, 7600, 7700, M16C</td>
</tr>
<tr>
<td>Siemens</td>
<td>C500, C166, Tricore</td>
</tr>
<tr>
<td>Microchip</td>
<td>PIC12, PIC16, PIC17</td>
</tr>
</tbody>
</table>

Choosing a Microcomputer

- Many factors to consider when selecting a microcomputer:
  - Labor, material, manufacturing, maintenance costs.
  - ROM, RAM, and EEPROM size.
  - Speed and I/O bandwidth requirements for application.
  - 8-, 16-, or 32-bit data size.
  - Numerical or other special operations required.
  - Number of parallel and serial ports needed.
  - Timer, PWM, and ADC requirements.
  - Package size and environmental issues.
  - Second source availability.
  - Availability of compilers, simulators, and emulators.
  - Power requirements.
MC68HC11 Architecture

- CPU optimized for low power and operation up to 4 MHz.
- Uses either two separate 8-bit accumulators (A,B) or one combined 16-bit accumulator (D).
- Has two 16-bit index registers (X,Y).
- Has powerful bit-manipulation instructions.
- Supports 16-bit add/subtract, 16 × 16 integer divide, 16 × 16 fractional divide, and 8 × 8 unsigned multiply.
- Lends itself to C compiler implementations.

MC68HC11 Family

- A series - basic model.
- D series - economical alternative, less memory, peripherals.
- E series - has wide range of I/O capabilities.
- F series - higher speed, extra I/Os.
- G series - 10-bit ADC and better timer systems.
- K series - high speed, larger memories, MMU, and PWM.
- L series - high speed, low power, static design.
- M series - has math coprocessor and four channel DMA.
- P series - power-saving PLL and 3 SCI ports.

Operating Modes

- The 6811 can operate in one of four modes:
  - Single-chip mode uses internal memory for program & data.
  - Expanded mode allows for use of external memory.
  - Bootstrap mode used to load programs into RAM.
  - Test mode used by Motorola to test the chip is operational.
### Address Map for MC68HC11E1CP2

<table>
<thead>
<tr>
<th>Address (hex)</th>
<th>Size</th>
<th>Device</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 to 01FF</td>
<td>512</td>
<td>RAM</td>
<td>Variables and stack</td>
</tr>
<tr>
<td>1000 to 103F</td>
<td>64</td>
<td>I/O</td>
<td></td>
</tr>
<tr>
<td>B600 to B7FF</td>
<td>512</td>
<td>EEPROM</td>
<td>Program and constants</td>
</tr>
</tbody>
</table>

### Assembly Language
- Assembly language instructions have four fields:
  - Label
  - Opcode
  - Operand(s)
  - Comment

<table>
<thead>
<tr>
<th>Label</th>
<th>Opcode</th>
<th>Operand(s)</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>here</td>
<td>ldaa</td>
<td>100</td>
<td>RegA = [100]</td>
</tr>
</tbody>
</table>

- Assembly instructions are translated into machine code:
  - Object code
  - Instruction
  - Comment

<table>
<thead>
<tr>
<th>Object code</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$86 $64</td>
<td>ldaa 100</td>
<td>RegA = [100]</td>
</tr>
</tbody>
</table>

### MC68HC11 Registers

- **Accumulator:** A, B, C, D, E, F
- **Index Registers:** X, Y
- **Stack Pointer:** SP
- **Program Counter:** PC

### MC68HC11 Addressing Modes
- **Inherent addressing mode (INH)**
- **Immediate addressing mode (IMM)**
- **Direct page addressing mode (DIR)**
- **Extended addressing mode (EXT)**
- **Indexed addressing mode (IND)**
- **Program counter relative addressing mode (REL)**
Inherent Addressing Mode

- Has no operand field.

Obj code  Op  Comment
$1B  aba  RegA = RegA + RegB

Immediate Addressing Mode

- Uses a fixed constant.
- Data is included in the machine code.

Obj code  Op  Operand  Comment
$8E00FF  lds  #$00FF  Initialize stack

Direct Page Addressing Mode

- Uses a 8-bit address to access from addresses 0 to $00FF.

Obj code  Op  Operand  Comment
$9632  ldaa  50  RegA = [$0032]

Extended Addressing Mode

- Uses a 16-bit address to access all memory and I/O devices.

Obj code  Op  Operand  Comment
$B71004  staa  $1004  [$1004] = RegA

Obj code  Op  Operand  Comment
$FE8000  ldx  $8000  RegX = [$8000]
Indexed Addressing Mode
- Uses an 8-bit unsigned offset with either RegX or RegY.

<table>
<thead>
<tr>
<th>Obj code</th>
<th>Op</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A704</td>
<td>staa</td>
<td>4,X</td>
<td>$[X+4] = RegA</td>
</tr>
</tbody>
</table>

PC Relative Addressing Mode
- Used for branch and branch-to-subroutine instructions.
- Assume branch located at $F880.

<table>
<thead>
<tr>
<th>Obj code</th>
<th>Op</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$8105</td>
<td>cmpa</td>
<td>#5</td>
<td>Compare RegA to 5</td>
</tr>
<tr>
<td>$25BE</td>
<td>blo</td>
<td>$F840</td>
<td>goto $F840 if RegA&lt;5</td>
</tr>
</tbody>
</table>

Instruction Types
- Data movement
- Clear/set
- Arithmetic and comparisons
- Logical operations
- Data test and bit manipulation
- Shift and rotate
- Branch, jump, and subroutine calls
- Interrupt handling
- Miscellaneous

Load and Store Instructions
- Used to move data to (from) registers from (to) memory.
- Load instructions are: ldaa, ldab, ldd, lds, ldx, and ldy.
- Load addressing modes are: IMM, DIR, EXT, IND.
- Store instructions are: staa, stab, std, sds, sdx, and sdy.
- Store addressing modes are: DIR, EXT, IND.
- Examples:

<table>
<thead>
<tr>
<th>Obj code</th>
<th>Op</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$86FF</td>
<td>ldaa</td>
<td>#$FF</td>
<td>IMM</td>
</tr>
<tr>
<td>$9725</td>
<td>staa</td>
<td>$25</td>
<td>DIR</td>
</tr>
<tr>
<td>$F60025</td>
<td>ldab</td>
<td>$0025</td>
<td>EXT</td>
</tr>
<tr>
<td>$E205</td>
<td>std</td>
<td>$05,X</td>
<td>IND</td>
</tr>
<tr>
<td>$FCC025</td>
<td>ldd</td>
<td>$C025</td>
<td>EXT</td>
</tr>
</tbody>
</table>
Other Data Movement Instructions

- Push and pull instructions used to put data onto and take data off of the stack.
  - psha, pshb, pshx, pshy, pula, pulb, pulx, puly (all INH).
- Transfer instructions used to copy data between registers.
  - tab, tap, tba, tpa, tsx, tsy, txs, tys (all INH).
- Exchange instructions used to exchange data between accumulator D and index registers X and Y.
  - xgdx, xgdy (all INH).

Example: Saving State to Stack

<table>
<thead>
<tr>
<th>Obj code</th>
<th>Op</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$183C</td>
<td>pshy</td>
<td>INH</td>
<td></td>
</tr>
<tr>
<td>$3C</td>
<td>pshx</td>
<td>INH</td>
<td></td>
</tr>
<tr>
<td>$37</td>
<td>pshb</td>
<td>INH</td>
<td></td>
</tr>
<tr>
<td>$36</td>
<td>psha</td>
<td>INH</td>
<td></td>
</tr>
<tr>
<td>$07</td>
<td>tpa</td>
<td>INH</td>
<td></td>
</tr>
<tr>
<td>$36</td>
<td>psha</td>
<td>INH</td>
<td></td>
</tr>
<tr>
<td>$32</td>
<td>pula</td>
<td>INH</td>
<td>body of subroutine</td>
</tr>
<tr>
<td>$06</td>
<td>tap</td>
<td>INH</td>
<td></td>
</tr>
<tr>
<td>$32</td>
<td>pula</td>
<td>INH</td>
<td></td>
</tr>
<tr>
<td>$33</td>
<td>pulb</td>
<td>INH</td>
<td></td>
</tr>
<tr>
<td>$38</td>
<td>pulx</td>
<td>INH</td>
<td></td>
</tr>
<tr>
<td>$1838</td>
<td>puly</td>
<td>INH</td>
<td></td>
</tr>
</tbody>
</table>

Clear/Set Instructions

- Used to initialize memory (clr), accumulators (clra, clrb), or bits in the CCR (clc, cli, clv).
- clr addressing modes are: EXT, IND.
- clra, clrb, clc, cli, clv are INH.
- Examples:
<table>
<thead>
<tr>
<th>Obj code</th>
<th>Op</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4F</td>
<td>clra</td>
<td>INH</td>
<td></td>
</tr>
<tr>
<td>$9725</td>
<td>staa</td>
<td>$25</td>
<td>DIR</td>
</tr>
<tr>
<td>$7F0025</td>
<td>clr</td>
<td>$0025</td>
<td>EXT</td>
</tr>
</tbody>
</table>
- The carry (C), interrupt mask (I), and overflow (V) bits in the CCR can also be set (sec, sei, sev).

Add and Subtract Instructions

- Registers: aba, abx, aby, sba (all INH).
- With carry to memory: adca, adc, sbca, sbcb.
- w/o carry to memory: adda, add, addr, addb, suba, subb, subb.
- Addressing modes are: IMM, DIR, EXT, IND.
- Examples: 16-bit addition using only A
<table>
<thead>
<tr>
<th>Obj code</th>
<th>Op</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$9625</td>
<td>ldaa</td>
<td>$25</td>
<td>load least sig byte</td>
</tr>
<tr>
<td>$9835</td>
<td>adda</td>
<td>$35</td>
<td>add data at $35 to A</td>
</tr>
<tr>
<td>$9745</td>
<td>staa</td>
<td>$45</td>
<td>store least sig byte</td>
</tr>
<tr>
<td>$9624</td>
<td>ldaa</td>
<td>$24</td>
<td>load most sig byte</td>
</tr>
<tr>
<td>$9934</td>
<td>adca</td>
<td>$34</td>
<td>add data at $34 to A</td>
</tr>
<tr>
<td>$9744</td>
<td>staa</td>
<td>$44</td>
<td>store most sig byte</td>
</tr>
</tbody>
</table>

13
**Multiply Instruction**
- Multiplies two unsigned 8-bit values in $A$ and $B$ to produce a 16-bit unsigned product stored in $D$ (i.e., $A \times B \rightarrow D$).
- Example:

```
0bj code  Op  Operand  Comment
$86FF  ldaa  #$FF  IMM
$CG14  ldab  #$14  IMM
$3D    mul   INH
```
- At the end, accumulator $D$ contains $13EC$.

**Divide Instructions**
- Use $D$ register for the dividend and $X$ register for the divisor.
- Resultant placed in $X$ register and remainder in $D$ register.
- \texttt{idiv} performs integer division.
- \texttt{fdiv} performs fractional division resulting in binary weighted fraction between 0 and 0.999998 (i.e., $(65536 \times D)/X \rightarrow X$).
- Example:

```
0bj code  Op  Operand  Comment
$CCFFFF  ldd  #$FFFF  IMM
$CBE2710 ldx  #$2710  X contains $0006
$02    idiv   D contains $159F
$CC0003  ldd  #$03  After idiv executes
$CBE006 ldx  #$06  X contains $8000
$03    fdiv   D contains $0000
```

**Compare Instructions**
- Perform a subtraction to update the CCR, but do not alter any data.
- Typically used just before a branch instruction.
- Compare registers: \texttt{cba} (INH).
- Compare to memory: \texttt{cmpa}, \texttt{cmpb}, \texttt{cpd}, \texttt{cpx}, \texttt{cpy}.
- Addressing modes: IMM, DIR, EXT, IND.
- Example: comparing with a known set point

```
0bj code  Op  Operand  Comment
$8650  ldaa  #$50  load set point into $A$
$B11031 cmpa  $1031  compare $A$ to memory
```
- If Z flag is 1 then the contents of $1031$ equals $50$.

**Miscellaneous Arithmetic Instructions**
- \texttt{daa} - decimal adjust $A$ for BCD.
- \texttt{dec}, \texttt{deca}, \texttt{decb}, \texttt{des}, \texttt{dex}, \texttt{dey} - decrement.
- \texttt{inc}, \texttt{inca}, \texttt{incb}, \texttt{ins}, \texttt{inx}, \texttt{iny} - increment.
- \texttt{neg}, \texttt{nega}, \texttt{ngeb} - two’s complement.
- \texttt{tst}, \texttt{tsta}, \texttt{tstb} - test for zero or minus and set $Z$ and $N$ flags in the CCR.
Logical Operation Instructions

- **AND** - anda, andb (IMM, DIR, EXT, IND).
- **Inclusive OR** - oraa, orab (IMM, DIR, EXT, IND).
- **Exclusive OR** - eora, eorb (IMM, DIR, EXT, IND).
- **1’s complement** - com, coma, comb.
- **Example:** masking unwanted bits

<table>
<thead>
<tr>
<th>Obj code</th>
<th>Op</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$8634</td>
<td>ldaa</td>
<td>#$34</td>
<td>%00110100</td>
</tr>
<tr>
<td>$840F</td>
<td>anda</td>
<td>#$0F</td>
<td>%00001111</td>
</tr>
</tbody>
</table>

Result in A is %00000100

Data Test and Bit Manipulation

- **bita** and **bitb** instructions perform an AND operation and update **N** and **Z** flags of the CCR w/o altering the operand.
- **bclr** and **bset** instructions are used to clear or set bit(s) in a given memory location.

```
  bclr dd, mm
  bset dd, mm
```

where **dd** is a memory location on page zero ($00dd) and **mm** is the mask byte.

Shift/Rotate Instructions

- Logical shift right (lsr, lsrA, lsrB, lsrD) shifts 0's into MSB.

```
  01001101  ->  01001101
```

- Arithmetic shift right (asr, asrA, asrB) retains MSB value.

```
  01001101  ->  0110
```

- Logical shift left (lsl, lslA, lslB, lslD) and arithmetic shift left (asl, aslA, aslB, aslD) are equivalent.

```
  01001101  ->  01001101
```

- Rotate left (rol, rola, rolB) put carry bit into the LSB.

```
  01001101  ->  10011010
```

- Rotate right (ror, rora, rorb) put carry bit into the MSB.

```
  01001101  ->  11011001
```

Jump and Branch Always

- **jmp** and **bra** instructions are unconditional.
- **bra** uses relative addressing (REL) so it can only be used to jump – 128 or 127 instructions.
- **jmp** can use DIR, EXT, and IND addressing so it can be used to jump anywhere in the 64K address space.
- **bra $** stops the progress of the CPU although it continues to execute this instruction.
**Single Condition Branches**

- **bcc** - branch if carry clear (i.e., \( C = 0 \)).
- **bcs** - branch if carry set (i.e., \( C = 1 \)).
- **bne** - branch if not equal to zero (i.e., \( Z = 0 \)).
- **beq** - branch if equal to zero (i.e., \( Z = 1 \)).
- **bpl** - branch if positive or zero (i.e., \( N = 0 \)).
- **bmi** - branch if negative (i.e., \( N = 1 \)).
- **bvc** - branch if overflow clear (i.e., \( V = 0 \)).
- **bvs** - branch if overflow set (i.e., \( V = 1 \)).
- **brn** - branch never

**Unsigned Number Branches**

- These branches usually follow `cba`, `cmp(A,B,D)`, `cp(X,Y)`, `sba`, `sub(A,B,D)` instructions.
- **bhi** - branch if higher ‘\( > \)’ (i.e., \( C + Z = 0 \)).
- **bhs** - branch if higher or same ‘\( \geq \)’ (i.e., \( C = 0 \)).
- **blo** - branch if lower ‘\( < \)’ (i.e., \( C = 1 \)).
- **blos** - branch if lower or same ‘\( \leq \)’ (i.e., \( C + Z = 1 \)).

**Signed Number Branches**

- These branches usually follow `cba`, `cmp(A,B,D)`, `cp(X,Y)`, `sba`, `sub(A,B,D)` instructions.
- **bgt** - branch if greater ‘\( > \)’ (i.e., \( Z \cdot (N \oplus V) = 0 \)).
- **bge** - branch if greater or equal ‘\( \geq \)’ (i.e., \( N \oplus V = 0 \)).
- **blt** - branch if less ‘\( < \)’ (i.e., \( N \oplus V = 1 \)).
- **ble** - branch if less or equal ‘\( \leq \)’ (i.e., \( Z \cdot (N \oplus V) = 1 \)).

**Bit Masking Branches**

- **brset** - performs logical AND of memory address and mask provided and branches only when all bits in the mask are set.
- **brclr** - performs logical AND of memory address and mask provided and branches only when all bits in the mask are clear.

<table>
<thead>
<tr>
<th>Obj code</th>
<th>Op</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$12051005$</td>
<td>brset</td>
<td>5,$10,$F889</td>
<td>goto $F889 if bit 4 of loc 5 is 1</td>
</tr>
<tr>
<td>$13051005$</td>
<td>brclr</td>
<td>5,$10,$F889</td>
<td>goto $F889 if bit 4 of loc 5 is 0</td>
</tr>
</tbody>
</table>
**Long Branches and Delays**

- Use branch and jump to branch outside of range:
  ```
  Address  Op  Operand  Comment
  C3B0  bcs  $40
  C3B2  Next instruction
  ...
  C3F2  jmp  PI  PI may be any valid address.
  ```

- Branch loops can be used to insert a time delay:
  ```
  Label  Op  Operand  Comment
  ldab  #$Count  Load B with count
  Delay  decb  decrement B
  bne  Delay  If B ≠ 0 goto Delay
  ```

  \[ DelayTime = t(\text{ldab}) + t(\text{decb}) + t(\text{bne}) \times \text{count} \]

**Interrupt Handling**

- `wai` instruction puts CPU into standby mode waiting for an external interrupt.

- `swi` instruction executes a software interrupt.

- `rti` instruction is called at the end of an interrupt service routine to restore the CPU registers.

**Subroutine Calls and Return**

- `bsr` - branch to subroutine using REL addressing.

- `jsr` - jumps to subroutine using DIR, EXT, or IND addressing.

- On either `bsr` or `jsr`, PC is automatically pushed onto the stack (least significant byte first).

- `rts` - return from subroutine, PC automatically pulled off the stack and jumps to that location.

**Storing the State on the Stack**

<table>
<thead>
<tr>
<th>STACK</th>
<th>CONDITION CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP − 9</td>
<td>ACCUMULATOR B</td>
</tr>
<tr>
<td>SP − 8</td>
<td>ACCUMULATOR A</td>
</tr>
<tr>
<td>SP − 7</td>
<td>INDEX REGISTER (XH)</td>
</tr>
<tr>
<td>SP − 6</td>
<td>INDEX REGISTER (XL)</td>
</tr>
<tr>
<td>SP − 5</td>
<td>INDEX REGISTER (YH)</td>
</tr>
<tr>
<td>SP − 4</td>
<td>INDEX REGISTER (YL)</td>
</tr>
<tr>
<td>SP − 3</td>
<td>RETURN ADDRESS (PC)</td>
</tr>
<tr>
<td>SP − 1</td>
<td>RETURN ADDRESS (PC)</td>
</tr>
<tr>
<td>SP</td>
<td>RETURN ADDRESS (PC)</td>
</tr>
</tbody>
</table>
Miscellaneous

- **nop** - no operation, creates a 2-cycle delay.
- **stop** - if $S$ flag in CCR is 0 then stop clocks to save power, recover after \textit{RESET}, \textit{XIRQ}, or unmasked \textit{IRQ}.
- **test** - test instruction used by manufacturer.

References (all linked on course website)

- Motorola 68HC11 Reference Manual
- Motorola 68HC11E Family Data Sheet
- Motorola 68HC11 Programming Reference Guide