Tutorial T24

Assembly Language Programming: Street Smarts from OOP

119th Convention
Audio Engineering Society
October 10, 2005
What we will cover

• **Real-world case study: Inside an MP3 player**
• OOP Principles for Embedded
  – Object
  – Encapsulation
  – Inheritance
• Summary: Benefits of OOP for Embedded
iPOD

http://www.chipmunk.nl/iPod/ipodChipmunk1.jpg
What’s in a typical DSP-based consumer audio device?
What’s in a typical DSP-based consumer audio device?
“Typical” MP3 Player

- µP
- DSP chip
  - D/A converter
- FireWire port
- Memory
- Disk Drive

- User Interface
  - Buttons
  - LCD
  - Knobs
- On-chip clock
- Power
  - Battery
  - Regulator
MP3 Player

- Microprocessor
- Memory
- DSP chip
- D/A converter
- FireWire Port
- Knobs
- LCD
- Power Regulator
- Oscillator
- Buttons
- Switches
- Battery
What we will cover

• Real-world case study: Inside an MP3 Player

• OOP Principles for Embedded
  – Object
  – Encapsulation
  – Inheritance

• Summary: Benefits of OOP for Embedded
Object

• A set of software
• Corresponds to something: (physical object, person, peripheral, algorithm, ...)
• Cleanly defined boundaries
Sample Objects

• µP
• Serial Port
• External Memory
• FFT routine
• Filter routine
• One interrupt service routine
• All the interrupt service routines
• ...

...
Sample Object: Serial Port ---

What’s in one?
TI C672x Serial Port
Prototypical Serial Port

Memory

Input buffers

Data

Addr

Registers

Data

Clock

IRQ
Prototypical serial port: Hardware

• Memory buffer
  – Base address
  – Length ("mod")
  – Current read, write pointers
  – Storage for other pointers for double-buffer

• Registers
Prototypical serial port:
Software

• Interrupt service routine: handle one sample
• Initialization
• Start/Stop
• Routine to deal with full buffer
  – Swap buffer pointers
  – Tell DSP that the buffer is full
Serial Port Elements

- Registers
- Input buffer(s)
- Interrupt service routine
- Initialization/reset
- Allocate buffers
- Setup transfer
- Start transfer
- Check buffer full
- Stop transfer
μP Serial Port: where are the object boundaries?
µP Serial Port Object: Boundaries Version 1

Memory

Input buffers

Data

Addr

Registers

IRQ

Data

Clock
One Object: Serial Port

- Registers
- Input buffer(s)
- Interrupt service routine
- Initialization/reset
- Allocate buffers
- Setup transfer
- Start transfer
- Check buffer full
- Stop transfer
µP Serial Port Object: Boundaries Version 2
3 Objects: Serial Port, Memory, ISRs

- Memory
  - Registers
  - Input buffer(s)

- ISRs
  - Interrupt service routine

- Serial Port
  - Initialization/reset
  - Allocate buffers
  - Setup transfer
  - Start transfer
  - Notify buffer full
  - Stop transfer
How to make an “object”

- Group similar things
- Decide the boundaries
- Header
  - file, or section of file
  - “#define”
    - interface for other objects --- more later
- Code file (or section of file)
  - details inside
- Test file
Review: Boundaries Version 1
Review, One Object: Serial Port

Registers
Input buffer(s)
Interrupt service routine

Initialization/reset
Allocate buffers
Setup transfer
Start transfer
Check buffer full
Stop transfer
Files, 1 Object

Memory.inc

ISR.inc

Serial_port.inc
Serial_port.asm

includes memory.inc, ISR.inc, Serial_port.inc
contains implementation for registers, buffers, ISR
contains implementation of rest of serial port
µP Serial Port Object: Boundaries Version 2

Memory

Input buffers

Data

Addr

Registers

Data

Clock

IRQ
Review, 3 Objects:
Serial Port, Memory, ISRs

- Memory
  - Registers
  - Input buffer(s)

- ISRs
  - Interrupt service routine

- Serial Port
  - Initialization/reset
  - Allocate buffers
  - Setup transfer
  - Start transfer
  - Notify buffer full
  - Stop transfer
Files, 3 Objects

Memory.inc
Memory.asm
includes memory.inc
contains implementation for registers, buffers

ISR.inc
ISR.asm
includes ISR.inc
contains implementation for ISR

Serial_port.inc
Serial_port.asm
includes Memory.inc, ISR.inc, Serial_port.inc
calls on implementation in Memory.asm, ISR.asm
contains implementation of rest of serial port
Real-world example: objects

NR.inc
NR.asm
NR_init.asm
  test_NR.asm

NR_mem.asm
  NR_mem.inc

NR_io.asm
  NR_io.inc

NR_fft_hann.inc
NR_fft_sincos.inc
NR_atan2.asm
  test_atan2
NR_sqrt.asm
  test_sqrt.asm
NR_fft.asm
  NR_fft.inc
  test_fft.asm
What we will cover

• Real-world case study: Inside an Ipod
• OOP Principles for Embedded
  – Object
  – Encapsulation
  – Inheritance
• Summary: Benefits of OOP for Embedded
Encapsulation

• (from Object): Cleanly defined boundaries
• Gather the details inside those boundaries
• Hide the details from the outside world
• Provide an interface to the outside world
Encapsulation: levels of access (inspired by C++)

- "Public":
- "Private":
- "Protected":
Encapsulation: levels of access (inspired by C++)

- “Public”: anybody can modify/see
- “Private”: nobody else can modify/see directly
- (“Protected”: only trusted entities can modify/see)
Serial Port Object: What is public/private?

- Registers
- Input buffer(s)
- Interrupt service routine
- Initialization/reset
- Allocate buffers
- Setup transfer
- Start transfer
- Notify buffer full
- Stop transfer
Serial Port Object: What is public/private?

- Registers
- Input buffer(s)
- Interrupt service routine
- "public:"
  - Initialization/reset
  - Allocate buffers
  - Setup transfer
  - Start transfer
  - Notify buffer full
  - Stop transfer
- "private:"

(Roles and functions related to the Serial Port Object are illustrated in a flowchart.)
Private: C++ code

class Serial_Port
{

private:
    int * buf_read_ptr;
    int * buf_write_ptr;
    int buf_len;
    void IRQ();
    int buf[SP_MAX_NUM_BUF][SP_MAX_BUF_LEN];
    int control_reg;
}
Interface: Public in C++ code

public:

...  
void  SP_Reset();
void  SP_Allocate_Buffers(
    int  Num_Buffers,
    int  Num_Points_Per_Buffer);
void  SP_Setup_Xfer();
void  SP_Start_Xfer();
int   SP_Buffer_Ready();
void  SP_Stop_Xfer();
int * SP_getReadPtr();
};
#include "serial_port.h"

void main()
{

    Serial_Port s;

    s.SP_Reset();

    s.SP_Allocate_Buffers(2, 256);
    s.SP_Setup_Xfer();
    s.SP_Start_Xfer();
}
Interface: C++ Main Loop

```cpp
while (...) {
    iReady = s.SP_Buffer_Ready();
    if (iReady == ...) {
        ...
    }
}

s.SP_Stop_Xfer();
```
Private: Motorola 56K example

MAX_BUF_LEN EQU 512
MAX_NUM_BUF EQU 4

org x:0

; private:
buf_base   dsm MAX_BUF_LEN*MAX_NUM_BUF
buf_read   ds MAX_NUM_BUF
buf_write  ds MAX_NUM_BUF
Public Interface: 56K example

; public:
org p:

SP_Allocate_Buffers
; ...
rts

SP_Setup_Xfer
; ...
rts

SP_Start_Xfer
; ...
rts
Interface: Typical main loop (56K)

```
move #2,r0            ; 2 buffers
move #256,r1          ; 256 points per buffer
jsr  SP_Allocate_Buffers
; ...
jsr  SP_Setup_Xfer
; ...
jsr  SP_Start_Xfer
main_loop:
; ...
jsr  SP_Buffer_Ready  ; a = flag
tst  a                 ; if a = 0
jne  main_loop
; ...
jmp  main_loop
```
Interface Usage:
Typical startup (56K)

reset:
    ; ...
    jsr Stack_Reset
    jsr SP_Reset
    jsr DAC_Reset
    jsr GUI_Reset
    jsr Mem_Reset
    jsr Clock_Reset
    ; ...
How to make “encapsulation”

• Plan “private,” “public”, (“protected”).
• Make objects (see above).
• Keep the details inside the object.
• Provide an interface for the outside world.
• Don’t let the outside world muck with what’s inside the object (except through the interface).
What we will cover

• Real-world case study: Inside an Ipod
• OOP Principles for Embedded
  – Object
  – Encapsulation
  – Inheritance
• Summary: Benefits of OOP for Embedded
Inheritance

- OOP: child class can inherit from the parent class
- Embedded: Find what is common, create a parent, inherit from it (abstraction)
MP3 Player: what is common?

- Microprocessor
- Memory
- DSP chip
- D/A converter
- Knobs
- FireWire Port
- Switches
- Buttons
- LCD
- Power Regulator
- Oscillator
- Battery
MP3 Player: what is common?

- Microprocessor
- Memory
- DSP chip
- D/A converter
- Knobs
- FireWire Port
- LCD
- Power Regulator
- Switches
- Buttons
- Oscillator
- Battery
Inheritance: Relationship of Objects
(Abstraction)
Inheritance: Shared Memory

Microprocessor

Memory

DSP chip

D/A converter

FireWire Port

User Interface

Oscillator

Power Regulator

Battery

Knobs

Switches

Buttons

LCD
Prototypical Plugin Architecture
Inheritance: Biquad structure
Inheritance: DSP

Shared Memory

\[
\begin{align*}
  &\ldots \\
  &a1 \\
  &a2 \\
  &b0 \\
  &b1 \\
  &b2 \\
  &\ldots \\
\end{align*}
\]

Current values as used by DSP, in “DSP Space”
Interpolate Biquad Coefficients

Current value: DSP

Target value: host

$b_0$
Inheritance: \( \mu P + DSP \)

Shared Memory

\[
\begin{array}{c}
\ldots \\
a_1 \\
a_2 \\
b_0 \\
b_1 \\
b_2 \\
\ldots \\
\ldots \\
\end{array}
\]

Target values written by host, in “Host Space”

Current values as used by DSP, in “DSP Space”
Inheritance: Macro define, call

Biquad    MACRO name
name + _a1:   allocate one memory location
name + _a2:   allocate one memory location
name + _b0:   allocate one memory location
name + _b1:   allocate one memory location
name + _b2:   allocate one memory location
MACRO END
#define BIQUAD_LEN 5

...  
Biquad HostSpace  
...  
Biquad DSPSpace  
...
Inheritance: Macro results

Shared Memory

```
... HostSpace_a1
HostSpace_a2
HostSpace_b0
HostSpace_b1
HostSpace_b2
...
DSPSpace_a1
DSPSpace_a2
DSPSpace_b0
DSPSpace_b1
DSPSpace_b2
...
```

Target values written by host, in “Host Space”

Current values as used by DSP, in “DSP Space”
Inheritance: interp using memory

```c
pHost = &HostSpace_a1;
pDSP = &DSPSpace_a1;

for (i=0; i< BIQUAD_LEN; i++) {
    *pDsp = *pHost ... *pDSP ...;
    pHost++;
    pDSP++;
}
```
Inheritance: Using memory

move #HostSpace_a1,r0 ; init pHost
move #DSPSpace_a1,r1 ; init pDSP
move #BIQUAD_LEN,y0   ; do 0<=i<n {
  do y0,p_interp
    move x:(r0),x0        ;   *pHost
    move x:(r4),x1        ;   *pDSP
    ; ...                 ;   *pDSP = ...
    move (r0)+            ;   pHost++;
    move (r1)+            ;   pDSP++
  p_interp:               ; }
Inheritance: Modify the Biquad

Biquad MACRO name
name + _a1: allocate one memory location
name + _a2: allocate one memory location
name + _b0: allocate one memory location
name + _b1: allocate one memory location
name + _b2: allocate one memory location
name + _gain: allocate one memory location
MACRO END

#define BIQUAD_LEN 6
### Inheritance: Macro results

<table>
<thead>
<tr>
<th>HostSpace_a1</th>
<th>HostSpace_a2</th>
<th>HostSpace_b0</th>
<th>HostSpace_b1</th>
<th>HostSpace_b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>HostSpace_gain</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DSPSpace_a1</th>
<th>DSPSpace_a2</th>
<th>DSPSpace_b0</th>
<th>DSPSpace_b1</th>
<th>DSPSpace_b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSPSpace_gain</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Shared Memory**

- **Target values written by host, in “Host Space”**
- **Current values as used by DSP, in “DSP Space”**
How to make “abstraction” and “inheritance”

• Look for objects that have similar properties.
• Abstract those properties into a common header file and source file.
• Include the header file.
• Invoke the common properties.
Inheritance: header files

- Project master .h
  - DSP master .h
  - μP master .h
  - Memory master .h
    - DSP memory .h
    - μP memory .h
    - GUI master .h
      - Knob
      - Switch
Abstraction: header files

- Project master .h
- DSP master .h
- μP master .h
- Memory master .h
  - DSP memory .h
  - μP memory .h
  - GUI master .h
    - Knob
    - Switch
What we will cover

• Real-world case study: Inside an Ipod
• OOP Principles for Embedded
  – Object
  – Encapsulation
  – Inheritance
• Summary: Benefits of OOP for Embedded
Benefits of OOP for Embedded

• Speed up my development time.
• Improve reliability of my code.
• Easier to isolate my bugs.
• Easier for me to modify my own code now.
• Easier for me to modify my own code later.
• Easier for others to modify my code.
• Easier to re-use code for new projects (maybe)
Street Smarts to apply

- Object
- Encapsulation
- Inheritance
iPOD

http://www.chipmunk.nl/iPod/ipodChipmunk1.jpg
MP3 Player

- Microprocessor
- Memory
- DSP chip
- D/A converter
- FireWire Port
- User Interface
- Oscillator
- Power Regulator
- Battery
- Knobs
- Switches
- Buttons
- LCD
OOP Resources

• http://www.objectfaq.com/oofaq2/
• Grady Booch, *Object-Oriented Analysis and Design with Applications*
• Grady Booch, *The Unified Modeling Language User Guide*
John Strawn, Ph.D.

S Systems Inc.

jstrawn@s-systems-inc.com

http://www.s-systems-inc.com