Generic Audio Driver for Windows CE 6.0

Manfredas Zabarauskas, Dimitrios Papastamos
Manfredas Zabarauskas

Edinburgh University,
BSc Artificial Intelligence and Computer Science,
1st year

Wolfson Microelectronics,
Applications Software Engineer Intern

Dimitrios Papastamos

Heriot-Watt University,
BSc Computer Science,
2nd year

Wolfson Microelectronics,
Applications Software Engineer Intern
• **Generic Audio Driver**
  • Project description
  • Overall design

• **Control Framework**
  • Overall design
  • Sample driver

• **Generic Wavedev2**

• **Audio Framework**
  • Overall design
  • Supported hardware
  • Sample implementations

• **Questions**
Generic Audio Driver
- Project description
- Overall design

Control Framework
- Overall design
- Sample driver

Generic Wavedev2

Audio Framework
- Overall design
- Supported hardware
- Sample implementations

Questions
• **Problem**
  The need to support multiple audio codec drivers on multiple platforms under WinCE 6.0.

• **Previous situation**
  Different platform vendors use different audio drivers. Linux example: four distinct drivers for WM8731.
  Leads to poor code re-use, maintenance/bug tracking difficulties, complicated new platform(codec) support, ... 

• **Solution**
  - Use standard WinCE 6.0 approach to audio drivers – Generic Wavedev2.
  - Separate chip control from audio data – Control Framework.
  - Separate CPU, board and codec – Audio Framework.
• **Generic Audio Driver**
  • Project description
  • Overall design
  • **Control Framework**
    • Overall design
    • Sample driver
  • **Generic Wavedev2**
  • **Audio Framework**
    • Overall design
    • Supported hardware
    • Sample implementations

• **Questions**
• Generic Audio Driver
  • Project description
  • Overall design
• Control Framework
  • Overall design
  • Sample driver
• Generic Wavedev2
• Audio Framework
  • Overall design
  • Supported hardware
  • Sample implementations

• Questions
Overall Design

WinCE 6.0 Kernel

Control Framework
- Driver Adaptor
- Driver Implementation

External driver
Control Bus

Wolfson Microelectronics Codec

Generic Wavedev2
- Driver Interface
- Stream Mixing
- I/O

Hardware Context

Audio Framework
- Codec Module
- CPU Module
- Board Module

CPU

 DMA, I²S
Audio Bus

Stream interface
Control Framework

- Provides IC control using same interface across different buses.
- Separates generic bus behaviour and platform-specific implementation details.
- Supports call forwarding if a bus driver exists in the system.
- Acts as a separate driver (.dll).
Control Framework

Control Framework.dll

CF2WireBus

CFSPiBus

CFSMDK6410SPIBus

(not yet implemented)

CF2WireBus

CFMainstonell2WireHWContext

CFFreescaleMX312WireHWContext

(not yet implemented)

Control bus

Stream interface

Wolfson Microelectronics Codec

Windows CE Device Manager

Manfredas Zabarauskas, Dimitrios Papastamos
What does the Control Framework do?

```c
PVOID CF2WireBus::Init(PVOID pData)
{
    CFInitContext *pInitContext = 0;
    CFHWContext *hwctx;

    // Initializing the error code
    DoSetLastError(CFS_SUCCESS);

    hwctx = ControlFramework::GetInstance().GetHWContext();
    if (!hwctx) { ... }

    pInitContext = (CFInitContext *)LocalAlloc(LPTR, sizeof(CFInitContext));
    if (!pInitContext) { ... }

    InitializeCriticalSection(&(pInitContext->critSec));
    if (!hwctx->Init(pInitContext)) { ... }

#if CF_2WIRE_INTR_ENABLE
    g_CF2WireBus_hEvent = CreateEvent(0, FALSE, FALSE, 0);
    if (!g_CF2WireBus_hEvent) { ... }

    irq2Wire = CF_2WIRE_IRQ;
    if (!KernelIoControl(...) { ... }

    if (!InterruptInitialize(g_CF2WireBus_sysIntr, ...) { ... }

    g_CF2WireBus_ISTRunning = TRUE;
    g_CF2WireBus_hThread = CreateThread( ... );
    if (!g_CF2WireBus_hThread) { ... }
#endif

    HKEY hKey = OpenDeviceKey(pContext);
    if (!hKey) { ... }

    if (RegQueryValueEx(hKey, L"ThreadPriority", ...) { ... }

    if (!CeSetThreadPriority(g_hTransferThread, threadPriority)) { ... }
    RegCloseKey(hKey);

    return pInitContext;
}

static DWORD WINAPI g_CF2WireBUS_IST(LPVOID lpParameter)
{
    CFHWContext *hwctx;
    hwctx = ControlFramework::GetInstance().GetHWContext();

    while (g_CF2WireBus_ISTRunning)
    {
        WaitForSingleObject(g_CF2WireBus_hEvent, INFINITE);
        if (!g_CF2WireBus_ISTRunning) { ... }
        if (hwctx)
        {
            hwctx->IST(lpParameter);
        } else { ... }
        InterruptDone(g_CF2WireBus_sysIntr);
    }

    return CFS_SUCCESS;
}
```
What does the hardware context do?

```c
CF_STATUS CFMainstonell2WireHWContext::Read(PVOID pOpenContext,
    CFio *in,
    BOOL isEndOfTransmission, DWORD *bytesRead)
{
    DWORD numberOfWriteTries = 5;
    DWORD numberOfReadTries = 5;
    CFOpenContext *pContext = (CFOpenContext *)pOpenContext;
    CF_STATUS retVal = CFS_SUCCESS;
    XLLP_UINT32_T reg;

    twowire_regs->ICR = XLLP_ICR_UIE | XLLP_ICR_SCLIE;

    RepeatedWrite:
    twowire_regs->IDBR = (in->GetSlaveAddress() << 1) | XLLP_IDBR_MODE;
    // Initiate the write
    reg = twowire_regs->ICR;
    reg &= ~(XLLP_ICR_ALDIE | XLLP_ICR_STOP);
    reg |= (XLLP_ICR_START | XLLP_ICR_TB);
    twowire_regs->ICR = reg;

    if (!WriteFinished(twowire_regs, WRITE_TIMEOUT)) {
        numberOfWriteTries--;
        if (numberOfWriteTries > 0) goto RepeatedWrite;
        // Send STOP condition
        twowire_regs->ICR &= ~(XLLP_ICR_STOP | XLLP_ICR_ACKNACK);
        twowire_regs->ICR &= ~(XLLP_ICR_UIE | XLLP_ICR_SCLIE);
        retVal = CFS_DATA_TIMED_OUT;
        goto StopCondition;
    }

    RepeatedRead:
    for (DWORD i = 0; i < in->GetCount(); i++)
    {
        twowire_regs->ICR
            = XLLP_ICR_UIE | XLLP_ICR_SCLIE;
        // Initiate the read
        reg = twowire_regs->ICR;

        if ((in->GetCount() - 1) != i) { ... }
        else // Last byte { ... }
        {
            UCHAR byte;
            reg |= (XLLP_ICR_START | XLLP_ICR_TB);
            if (!ReadFinished(twowire_regs, READ_TIMEOUT)) { ... }
        }
        else { ... }

        // Wait for transmission
        if (in->GetInData())[i] = (UCHAR)(twowire_regs->IDBR & 0xFF);
        if (bytesRead)
        {
            ++*bytesRead;
        }
    }

    StopCondition:
    // Signal stop
    twowire_regs->ICR &= ~(XLLP_ICR_STOP | XLLP_ICR_ACKNACK);
    twowire_regs->ICR &= ~(XLLP_ICR_UIE | XLLP_ICR_SCLIE);
    return retVal;
}
```
DWORD CFSMRP6400I2CBus::Read(PVOID hOpenContext, LPVOID pBuffer, DWORD count)
{
    CFOpenContext *pOpenContext = (CFOpenContext *)hOpenContext;
    CFIo *ioData = (CFIo*)pBuffer;
    DWORD retVal = 0;

    // Initializing the error code
    DoSetLastError(CFS_SUCCESS);

    // Initializing the error code
    DoSetLastError(CFS_SUCCESS);

    if (!pBuffer || sizeof(CFIo) != count ||
        !pOpenContext || !ioData)
    {
        SetLastError(ERROR_INVALID_PARAMETER);
        ...  
    }

    IIC_IO_DESC IIC_Data_In;
    IIC_Data_In.SlaveAddress = (ioData->GetSlaveAddress() << 1) + 1;
    IIC_Data_In.Data = ioData->GetInData();
    IIC_Data_In.Count = ioData->GetCount();

    IIC_IO_DESC IIC_Data_Out;
    IIC_Data_Out.SlaveAddress = (ioData->GetSlaveAddress() << 1) + 1;
    IIC_Data_Out.Data = ioData->GetOutData();
    IIC_Data_Out.Count = ioData->GetRegisterWidth();

    if (!DeviceIoControl(pOpenContext->hDriverHandle, IOCTL_IIC_READ, &IIC_Data_Out, sizeof(IIC_IO_DESC), &IIC_Data_In, sizeof(IIC_IO_DESC), &retVal, NULL))
    {
        SetLastError(ERROR_INVALID_PARAMETER);
    }

    return retVal;
}
Structure

- **Generic Audio Driver**
  - Project description
  - Overall design
- **Control Framework**
  - Overall design
  - Sample driver
- **Generic Wavedev2**
- **Audio Framework**
  - Overall design
  - Supported hardware
  - Sample implementations
- **Questions**
• **Generic Audio Driver**
  • Project description
  • Overall design
• **Control Framework**
  • Overall design
  • Sample driver
• **Generic Wavedev2**
• **Audio Framework**
  • Overall design
  • Supported hardware
  • Sample implementations

• **Questions**
Overall Design

Control Framework

Driver Adaptor

Driver Implementation

External driver

Control Bus

Wolfson Microelectronics Codec

WinCE 6.0 Kernel

Generic Wavedev2

Audio Framework

Codec Module

CPU Module

Board Module

DMA, I²S

CPU

Stream Interface

Driver Interface

Stream Mixing

Hardware Context

I/O

I/O...
Wavedev2

- Standardized Windows CE approach to audio drivers.
- Multiple-stream input/output support, built-in mixing, volume control, format/sample-rate conversion.

<table>
<thead>
<tr>
<th>Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>devctxt.cpp</td>
</tr>
<tr>
<td>input.cpp</td>
</tr>
<tr>
<td>midistrm.cpp</td>
</tr>
<tr>
<td>mixerdrv.cpp</td>
</tr>
<tr>
<td>output.cpp</td>
</tr>
<tr>
<td>strmctxt.cpp</td>
</tr>
<tr>
<td>wavemain.cpp</td>
</tr>
<tr>
<td>hwctxt.cpp</td>
</tr>
</tbody>
</table>
Problem

• Different (and incompatible) Wavedev2 drivers in every BSP.
• Insufficiently abstracted from hardware – portability, code re-use and support issues.
• Not flexible enough – many input/output formats not supported by the driver, e.g. stereo playback on mono output.
Solution

• Different Wavedev2 drivers analysed (Microsoft/Ensoniq, WM’s WDCL, others) best ideas used when writing a Generic Wavedev2 driver.
• Extended stream and mixing support.
• Hardware specific behaviour migrated into the hardware context (hwctxt.cpp).
• The latter formed the basis for the Audio Framework: further hardware separation into codec, board and CPU.
• Generic Audio Driver
  • Project description
  • Overall design
  • Control Framework
    • Overall design
    • Sample driver
  • Generic Wavedev2
  • Audio Framework
    • Overall design
    • Supported hardware
    • Sample implementations

• Questions
• **Generic Audio Driver**
  • Project description
  • Overall design
  • **Control Framework**
    • Overall design
    • Sample driver
  • **Generic Wavedev2**
  • **Audio Framework**
    • Overall design
    • Supported hardware
    • Sample implementations

• **Questions**
Overall Design

WinCE 6.0 Kernel

Control Framework
- Driver Adaptor
- Driver Implementation

Wolfson Microelectronics Codec

Generic Wavedev2
- Driver Interface
- Stream Mixing
- Hardware Context
- I/O

Audio Framework
- Codec Module
- CPU Module
- Board Module

CPU

I/O... DMA, I²S

Audio Bus

Stream interface

External driver

Control Bus
Audio Framework

• The Audio Framework is an abstraction layer that allows audio codec drivers to integrate with multiple target platforms.
• Separates CPU, board and codec implementation details.
• Provides auto-configuration facilities such as clocking, capabilities matching and path management.
• Features a flexible and extensible API. Future extensions might include advanced power management and dynamic routing.
Overall design:

AFCPU
- S3C6400
- PXA270

AFCore

AFCPU
- WM8988
- WM8903
- WM8523

AFCboard
- 6097-EV1-REV2
- 6228-EV1-REV1
- 6201-EV1-REV2
- 6220-EV1-REV1

AFCodec
- WM8993

Audio Framework
Overall design:

- AFCPU
  - AFCPU PXA270
  - AFCPU S3C6400

- AFCore

- AFCodec
  - AFCodec WM8993
  - AFCodec WM8988
  - AFCodec WM8903
  - AFCodec WM8523

- AFBoard
  - AFBoard 6097-EV1-REV2
  - AFBoard 6228-EV1-REV1
  - AFBoard 6201-EV1-REV2
  - AFBoard 6220-EV1-REV1
Overall design:

AFCPU
   - AFCPU PXA270
   - AFCPU S3C6400

AFCore

AFCodec
   - AFCCodec WM8993
   - AFCCodec WM8988
   - AFCCodec WM8903
   - AFCCodec WM8523

AFBoard
   - AFBoard 6097-EV1-REV2
   - AFBoard 6228-EV1-REV1
   - AFBoard 6201-EV1-REV2
   - AFBoard 6220-EV1-REV1
Supported platforms:

- MainstonelII (I²C, I²S, DMA)
- Samsung SMRP6400 (I²C, I²S, DMA)

Supported codecs:

- WM8993 (master/slave, playback/recording, multiple formats)
- WM8903 (slave, playback/recording, multiple formats)
- WM8523 (slave, playback, multiple formats)
- WM8988 (slave, playback/recording, multiple formats, basic power management)

Supported boards:

- 6201-EV1-REV2
- 6220-EV1-REV1
- 6228-EV1-REV1
- 6097-EV1-REV2
To support a new codec you need to:

• Fill in the codec capabilities so it can be picked up by the Audio Framework when doing automatic configuration.
• Implement `AFCodec::Init` to initialize the codec given the currently used path.
• Implement `AFCodec::InitI2SBus` to support all the featured capabilities of the codec such as different sample rates, master/slave playback and recording.
• Implement `AFCodec::Deinit` for shutting down the codec (optional).
• Implement other codec specific functionality like muting/unmuting, actions on power-up/power-down and so on. (optional).
AF_STATUS AFCodecHardware::InitI2SBus(AFI2SBusCap *cap)
{
    ...
    if (MASTER == cap->mode)
    {
        USHORT LRCLK_DIV = 2 * cap->bitLength;
        // Master LRCLK
        WRITE_CODEC(WM8993_AUDIO_INTERFACE_4, LRCLK_DIV);
        FLOAT BCLK_DIV =
            (FLOAT)((USHORT)((core->GetBoard()->GetSysClk() / (FLOAT)(cap->samplingFreq) + 0.5f)) / (FLOAT)LRLCLK_DIV;
        UINT16 i, BCLK_DIV_count = sizeof(BCLK_DIV_valid) / sizeof(FLOAT);
        for (i = 0; i < BCLK_DIV_count; ++i)
        {
            if (BCLK_DIV_valid[i] == BCLK_DIV)
            {
                WRITE_CODEC(WM8993_CLOCKING_1, (i << 1));
                break;
            }
        }
        if (BCLK_DIV_count == i) { ... }
        WRITE_CODEC(WM8993_AUDIO_INTERFACE_3, 0x8000);
    }
    ...
    UINT32 sysRates[] = { FS_64, FS_128, FS_192, FS_256, FS_384, ... };
    UINT32 sampleFreqs[] = { HZ_8000, HZ_11025, HZ_16000, ... };
    for (int i = 0; i < LEN(sysRates); ++i)
    {
        if (cap->sampleRate == sysRates[i])
        {
            val |= ((i & 0xf) << 1);
            break;
        }
    }
    for (int i = 0; i < LEN(sampleFreqs); ++i)
    {
        if (cap->samplingFreq == sampleFreqs[i])
        {
            val |= ((i & 0xf) << 7);
            break;
        }
    }
    WRITE_CODEC(WM8993_CLOCKING_3, val);
    WRITE_CODEC(WM8993_WRITE_SEQ_3, 0x0108);
    return AFS_SUCCESS;
}
To support a new CPU you need to:

- Fill in the CPU capabilities so it can be picked up by the Audio Framework when doing automatic configuration.
- Map/unmap the registers to memory for configuring the controllers.
- Initialize I2S (either directly or via another driver or library).
- Initialize and prepare DMA (either directly or via another driver or library).
- Create an IST to trigger input/output events in case the CPU does not have distinct IRQs for input/output.
- Implement power up/down functionality.
- Implement other CPU specific functionality (optional).
AF_STATUS AfCPUHardware::InitI2SBus(AFI2SBusCap *cap)
{
    ...
    pSysConReg->CLK_SRC = (pSysConReg->CLK_SRC & ~(0x7<<7)) | 0x4;
    pSysConReg->CLK_DIV2 = (pSysConReg->CLK_DIV2 & ~(0xF<<8));
    ...
    IIS_set_active_off();
    IIS_port_initialize(IIS_CH_0);
    ...
    switch (cap->mode)
    {
        case MASTER:
        {
            IIS_set_interface_master_slave_mode(IIS_MASTER_BYPASS_MODE);
        }
        break;
        case SLAVE: { ... }
        break;
        default: { ... }
    }
    sysClk = 16933333.33f;
    switch (cap->samplingFreq)
    {
        case HZ_44100:
        {
            IIS_set_interface_bit_clock_frequency(IIS_BIT_CLOCK_32FS);}
        break;
        case HZ_32000: { ... }
        default: { ... }
    }
    switch (cap->bitLength)
    {
        case BIT_16:
        {
            IIS_set_interface_bit_length(IIS_BIT_LENGTH_16BIT);
        }
        ...
        ...
        ...
        switch (cap->dataAlign)
        {
            case I2S:
            {
                IIS_set_interface_transmit_receive_mode(IIS_TRANSFER_BOTH);
            }
            ...
            ...
            ...
        }
        ...
        return AFS_SUCCESS;
    }
Overall Design

WinCE 6.0 Kernel

Control Framework
- Driver Adaptor
- Driver Implementation

Generic Wavedev2
- Driver Interface
- Stream Mixing
- I/O

Audio Framework
- Codec Module
- CPU Module
- Board Module

External driver

Wolfson Microelectronics Codec

CPU

Stream interface

Audio Bus

Control Bus

DMA, I²S
Overall Design

WinCE 6.0 Kernel

Control Framework

Driver Adaptor

Driver Implementation

External driver

Control Bus

Wolfson Microelectronic Codec

Generic Wavedev2

Driver Interface

Stream Mixing

I/O

Hardware Context

Audio Framework

Codec Module

CPU Module

Board Module

DMA, I²S

CPU

Stream interface

Audio Bus

Wolfson Microelectronics

Manfredas Zabarauskas, Dimitrios Papastamos

September 09 | www.wolfsonmicro.com
Questions?