Topic 3: (Part D)

I/O and Peripheral Devices Management

Sound in the Nintendo DS

Systèmes Embarqués Microprogrammés

Prof. David Atienza Alonso, SEL-STI

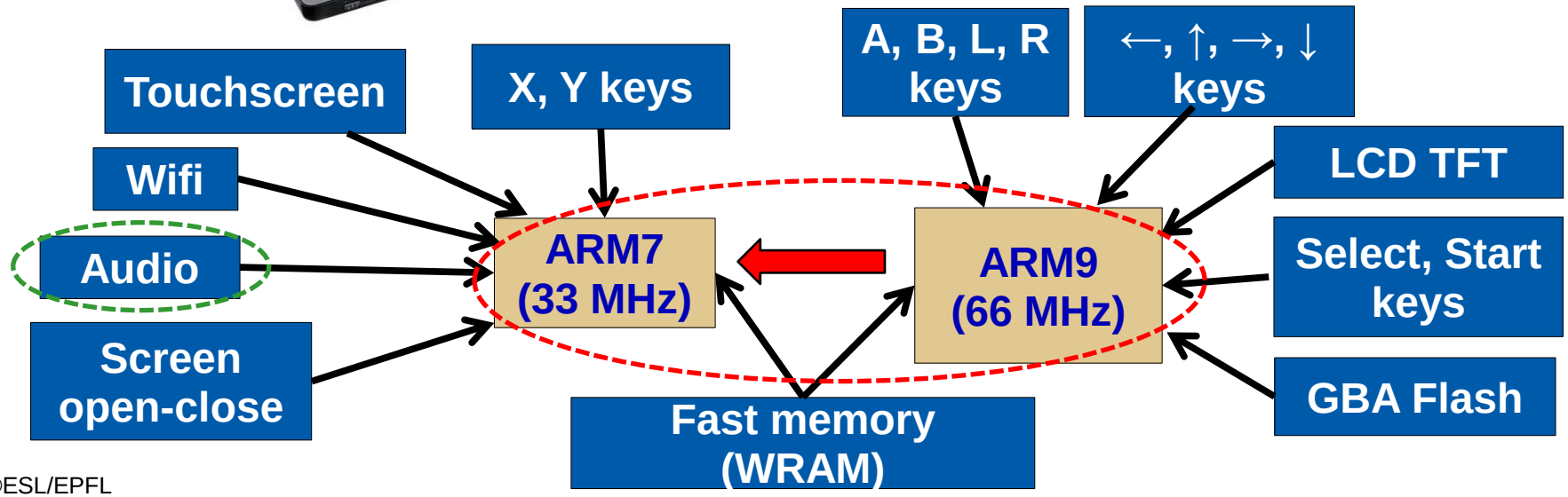
Content of Session

- Use of sounds in the NDS
  - Multiprocessor interaction: ARM7-ARM9
  - MaxMod library methods and libnds methods
  - Transformation utility and process for audio files: modules and sounds
  - Audio streaming (example)
  - Audio recording (example)

- Enabling sounds in combination to other I/O devices of NDS
  - Adding background music and sounds to the Simon game
  - Creating a piano keyboard for the NDS
  - Adding preexisting background music and effects to the Tetris

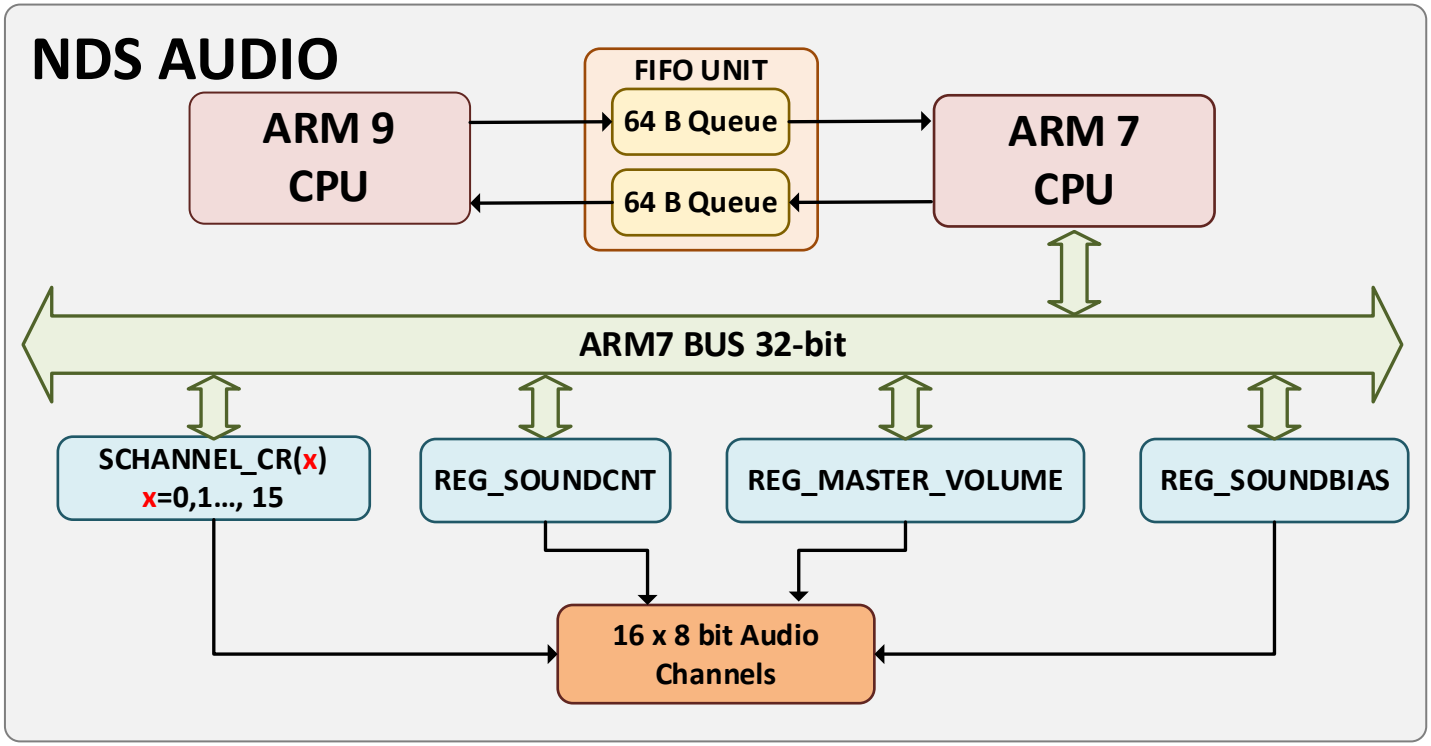
I/O subsystem management on the NDS: Audio

- Shared cooperation possible among two 32-bit ARM cores to create sound
  - ARM 7TDMI-S: only core with access to I/O interface (16 x 8-bit audio channels)
  - ARM 946E-S: performs complex audio transforms and send data to ARM7
    - Pointer to the data to play



Audio Peripheral on the NDS:

- FIFO is used for *inter-processor-communication (IPC)* between ARM9 and ARM7
- ARM7 has access to audio peripheral



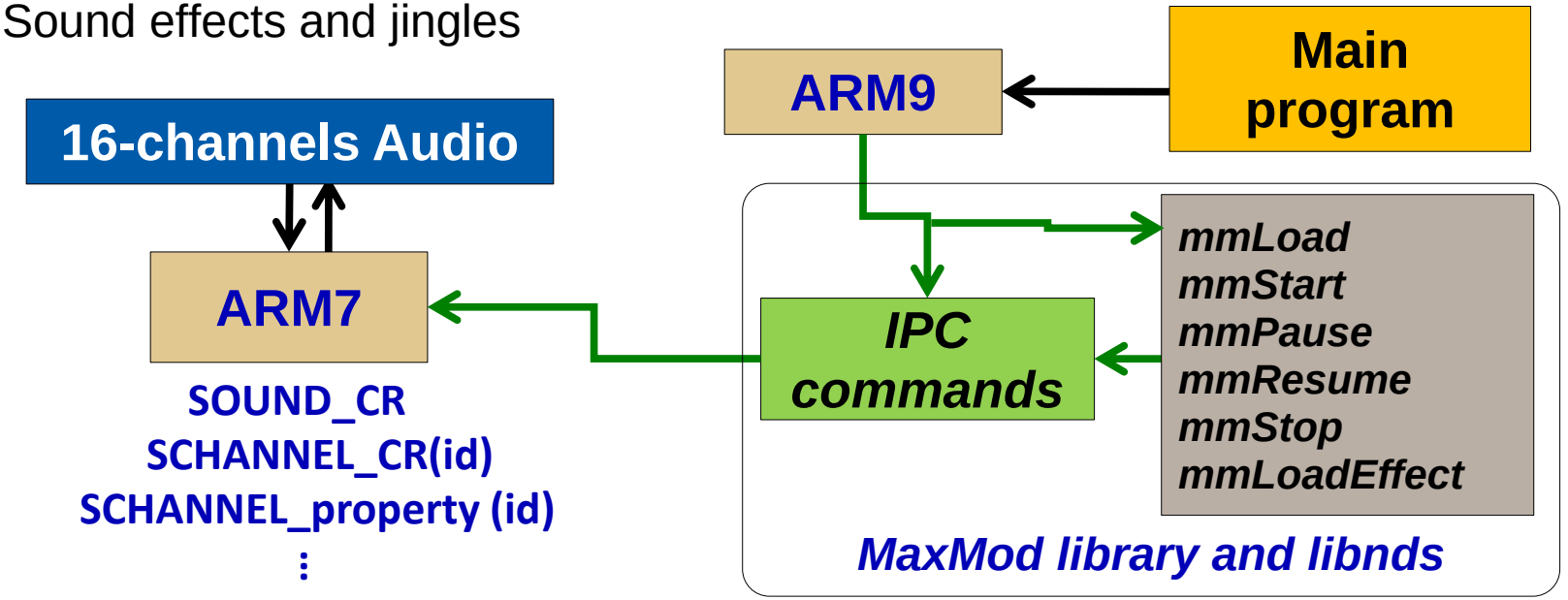
Audio I/O access: Specialized I/O registers from ARM7

- Write in control registers or macro to power up the sound I/O subsystem:
  powerON (POWER_SOUND);
- Global configuration stored in special configuration register: **SOUND_CR**
  - Enable and set vol.: **SOUND_CR= SOUND_ENABLE | SOUND_VOL(0x7F);**
    - 127 possible volumes: from silent (0x00) to full (0x7f)
- Each channel configured independently:
  - Channel activation through **SCHANNEL_CR(index)**
    - Configuring channel 0: **SCHANNEL_CR(0) = SCHANNEL_ENABLE | SOUND_ONE_SHOT | SOUND_8BIT;**
  - Configure at least three properties through **SCHANNEL_property(index)**
    - Configuring channel 0:
    1. Playback frequency: **SCHANNEL_TIMER(0) = SOUND_FREQ(11127);**
    2. Pointer to sound to play: **SCHANNEL_SOURCE(0) = (uint32)sound1;**
    3. Duration (in 32-bit words): **SCHANNEL_LENGTH(0) = ((int)sound1_end - (int)sound) >> 2;**

Requires the use of special project template in devKitPRO with two main programs: one for ARM9 and another one for ARM7

Synchronization of Audio Interfaces: use of MaxMod library in devkitPro

- Control ARM9-ARM7 for audio interaction with libnds and MaxMod:
  1. Synchronization: translates *inter-processor communication (IPC)* commands
  2. Buffering: multiple sound commands stored in internal memories (WRAMx)
     - Playing background music (or modules)
     - Sound effects and jingles



- Application Programmer Interface (API): http://www.maxmod.org/

  /opt/devkitPro/libnds/include/maxmod9.h

  /opt/devkitPro/libnds/include/mm_types.h

Audio input files for MaxMod in NDS

- The MaxMod library accepts two different types of files
  - Module files: background music
  - Sound effects: played on demand sporadically

- Module files in four possible formats: ***.mod, .s3m, .it*** or ***.xm***
- Sound effects can be in one predefined format: ***.wav***
  - A module sound format played only once (not looping) is accepted, but not recommended
  - Wav files generate usually large binary files: do not include many!

- A large number of free resources on the Internet with pre-created modules and sounds
  - Look for the links proposed in the course Moodle Site

MaxMod API: Main Initialization Methods

- Initializing the library pointers to sounds and internal buffers
  - void mmInitDefaultMem(mm_addr soundbank);
    - The input parameter is the name of the sound-bank binary object (by default it is ***soundbank_bin***)
- Load music modules
  - void mmLoad(mm_word module_ID);
    - The input parameter is the 32-bit index with the module identifier (by default all identifiers are defined in ***soundbank.h***)
- Load sound effect
  - void mmLoadEffect(mm_word sample_ID);
    - The input parameter is the 32-bit sample index with the effect identifier (by default all identifiers are defined in ***soundbank.h***)

MaxMod API: Music modules

- Play music
  - void mmStart(mm_word module_ID, mm_pmode mode);
    - The first input parameter is the module identifier
    - The second parameter specifies whether the music has to be played once (**MM_PLAY_ONCE**) or in an infinite loop (**MM_PLAY_LOOP**)
- Pause, resume or stop music using active module identifier
  - void mmPause();
  - void mmResume();
  - void mmStop();

MaxMod API: Sound effects

- Play sound effect
  - mm_sfxhand mmEffect(mm_word sample_ID);
    - The input parameter specifies sound effect identifier of effect to play
    - The effect will be played without modifying the sound configuration
- Play sound effect with specific sound configuration
  - mm_sfxhand mmEffectEx(mm_sound_effect* sound);
    - The input parameter is a structure with the effect identifier (id) and three main parameters:
      1. Volume
      2. Panning
      3. Rate (frequency)

```
typedef struct t_mmsoundeffect
{
    union {
// sample ID (defined in soundbank header)
    mm_word id;
// external sample address, not valid on GBA system
    mm_ds_sample* sample;
    };

    mm_hword rate; // playback rate
    mm_sfxhand handle; // sound handle
    mm_byte volume; // volume, 0..255
    mm_byte panning; // panning, 0..255

} mm_sound_effect;
```

Transformation utility to generate accepted NDS audio formats: mmutil

- Raw audio data is needed
  - We must convert audio files into uncompressed binary files
- Toolchain for MaxMod library in the devkitPro: ***mmutil***
  - As with images, the transformation of sound files is automatic:
    - Parameter to generate directly NDS compatible outputs: ***–d***
  - Possible to use it from the terminal
    - Generates a sound-bank output for NDS: ***soundbank.bin***
    - Header file to link with the C code**: *soundbank.h***
  - Example: Transform ***s1.wav*** and ***s2.mod*** and dump the binary data into the default sound-bank output for NDS

mmutil s1.wav s2.mod –d –osoundbank.bin –hsoundbank.h

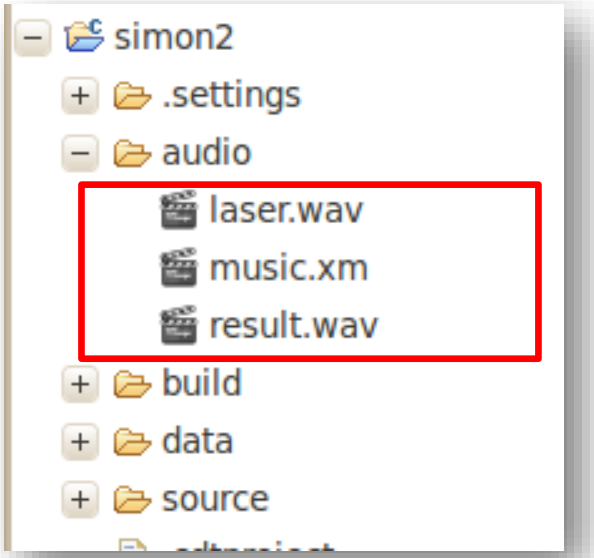
Automatic transformation and use of audio files in NDS project

1. Place audio files in the folder ***audio*** inside the C project
2. Rebuild the project (clean it if necessary). Several files will be generated in the ***build*** folder
   - **soundbank.bin**: Binary output from the mmutil tool.
   - **soundbank.h**: Header file including the necessary definitions to manage the sounds in the program
     - It must be linked with the user C code
   - **soundbank.bin.o** and **soundbank_bin.h**: Object and header file used by the library generated from the previous two files
3. Use the sounds in the C project with MaxMod API
   A. Include the library and sound-bank headers
   B. Initialize the library
   C. Load (1) music modules and (2) effects
   D. Play (1) music modules and (2) effects

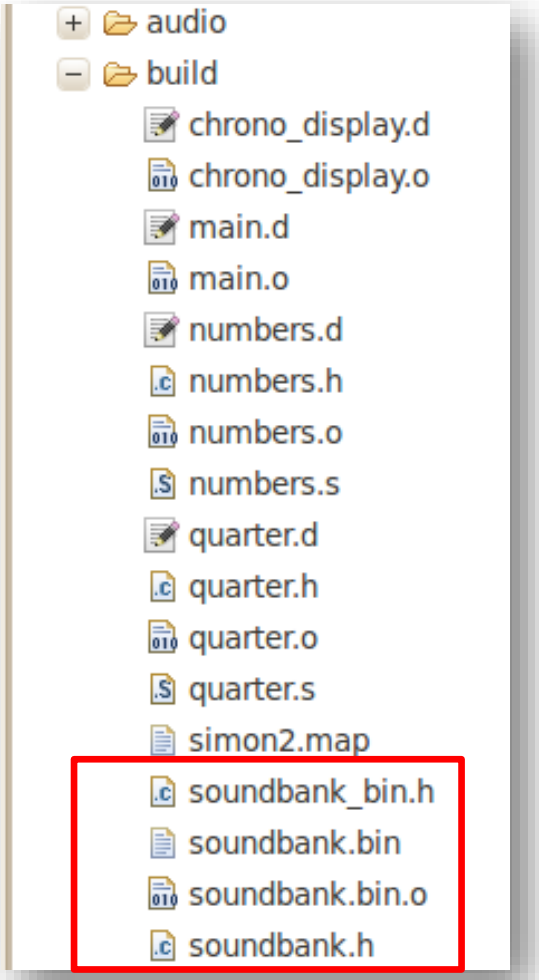
Example: Simon game sound

- How do we integrate sound in the Simon game of last week?

1. Put the sound files in the audio folder of the project



2. Remake the project (clean if necessary). The soundbank files will be created in the *build* folder



Example: Simon game sound

3.A. Include the library and the sound bank headers wherever it is needed

```
1 #include <nds.h>
2 #include <stdio.h>
3 #include "quarter.h"
4 #include "math.h"
5 #include "chrono_display.h"
6
7 #include <maxmod9.h>
8 #include "soundbank.h"
9 #include "soundbank_bin.h"
10
```

3.B. Initialize the sound library

```
63 //---------------------------------------------------
64 int main(void) {
65 //---------------------------------------------------
66     //SOUND
67     //Init the sound library
68     mmInitDefaultMem((mm_addr)soundbank_bin);
69     //Load module
70     mmLoad(MOD_MUSIC);
71     //Load effect
72     mmLoadEffect(SFX_LASER);
73     mmLoadEffect(SFX_RESULT);
74
```

3.C1. Load the music modules

3.C2. Load the sound effects

Example: Simon game sound

3.D1. Play / stop the music

```
//If start is pressed, the game starts
if(keysDown() & KEY_START)
{
    //Start music
    mmStart(MOD_MUSIC,MM_PLAY_LOOP);
    //Init the randomizer
```

```
if(active_button == NONE)
{
    //End of the game (Stop Time Timer)
    if(hits == 0)
    {
        hits--;
        TIMER2_CR &= ~TIMER_ENABLE;
        last = active_button;
        mmStop();
        mmEffect(SFX_RESULT);
    }
    else if(hits > 0)
```

3.D2. Play the effects

```
    if(active_button == GREEN) active_button = NONE;
if(x<=-3 && y >=3 && radius >=19 && radius <96) //YELLO touched
    if(active_button ==YELLOW) active_button = NONE;
//Play sound effect if the button was successfully touched
if(temp_butt != active_button) mmEffect(SFX_LASER);
```

Advanced audio management with MaxMod: Streaming

- Maxmod provides functions to play streams of sound
  - Created at run-time
  - Pre-stored in sound-banks and modified at run-time
- Four steps needed:
  1. Initialize the sound system: ***mm_ds_system*** structure to configure
     - If uses a sound-bank not previously created, we use special initialization
  2. Write the “filling” function: ***mm_word on_stream_request (…)***
     - Filling the buffer to be played: typically using a ***for***/***while*** loop
     - Return the number of samples placed in the buffer to play
  3. Create stream structure (***mm_stream***) and link to filling function, two options:
     - Automatic: filling function called automatically when needed
     - Manual: user updates stream periodically enough (not recommended)
  4. Open audio stream: ***void mmStreamOpen (mm_stream myStream);***

Advanced audio management: Streaming sound data filled at run time

1. Special initialization in case of not using a soundbank
   - Specify number of modules
   - Specify number of samples
   - Specify memory bank
   - Initialize the system

```
//---------------------------------------------------------------
// initialize maxmod without any soundbank (unusual setup)
//---------------------------------------------------------------
mm_ds_system sys;
sys.mod_count           = 0;
sys.samp_count          = 0;
sys.mem_bank            = 0;
sys.fifo_channel        = FIFO_MAXMOD;
mmInit( &sys );
```

Advanced audio management: Streaming sound data filled at run time

2. Write filling function (for / while loop)

- The returning types and input parameters are fixed
- It returns the number of samples written into the buffer
- Example: Create white noise

```
mm_word on_stream_request( mm_word length, mm_addr dest, mm_stream_formats format ) {

    s16 *target = dest;
    mm_word temp_length = length;

    //White noise
    for( ;length; length-- )
    {
        int sample = rand();
        // output sample for left
        *target++ = sample;
        // output inverted sample for right (STEREO)
        *target++ = -sample;
    }
    //Returns the number of samples filled
    return temp_length;
}
```

Advanced audio management: Streaming sound data filled at run time

3. Create a stream structure and link the filling function.

```
//---------------------------------------------------------------
// open stream
//---------------------------------------------------------------
mm_stream *mystream = malloc(sizeof(mm_stream));
mystream->sampling_rate = 25000;                // 25khz
mystream->buffer_length = 1200;                 // 1200 samples
mystream->callback      = on_stream_request;    // set callback function
mystream->format    = MM_STREAM_16BIT_STEREO;   // stereo 16-bit
mystream->timer     = MM_TIMER0;                // use hardware timer 0
mystream->manual    = 0;                        // use automatic filling
//Open stream
mmStreamOpen( mystream );
```

4. Open the stream

Advanced audio management: Recording sound from the microphone

- NDS has a built-in mono microphone
  - Software support given by libnds, **not using MaxMod**
  - Documented in **/opt/devkitPro/libnds/include/nds/arm9.sound.h**

1. Declare buffers to be filled by the microphone and (optionally) to play-back the recorded sound
2. Implement microphone handler
   - Called by the library when half or full buffer is filled
   - Process the sampled data (e.g.: copy it to a bigger storage buffer)
3. Initialize the sound library (only once, but compulsory)
4. Start / Stop the recording
5. Eventually play-back the recorded sound

Libnds sound API: Initialization and sound recording

- Enable the sound (initialization of the library)
  - void soundEnable(void)
- Disable the sound
  - void soundDisable(void)
- Start sound recording
  - int soundMicRecord(void *buffer, u32 bufLength,
    MicFormat format, int freq, MicCallback callback);
    - The handler/callback function must follow the following prototype

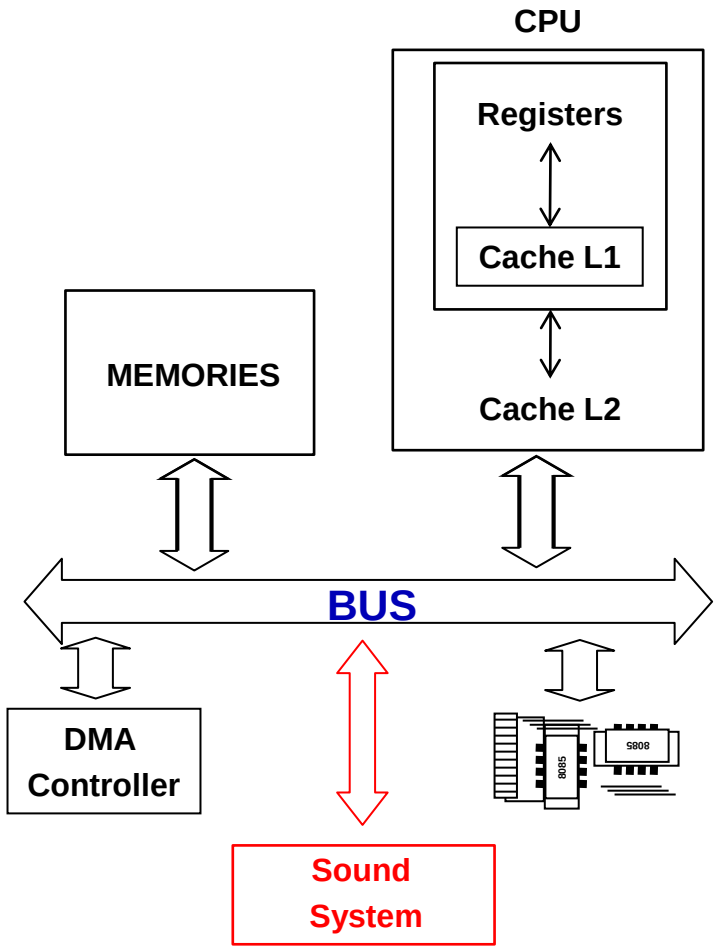
      void myFunction(void* firstNewSample, int numNewSamples)

```
typedef enum {
    MicFormat_8Bit = 1,
    MicFormat_12Bit = 0
}MicFormat;
```

- Stop recording
  - void soundMicOff(void)

Libnds API: Memory Inconsistency Hazard

- Processors usually are equipped with fast on-chip caches
  - Attempt to avoid latency on memory access by keeping an **updated copy of the data**
- Inconsistency hazards
  - Other peripherals may change memory contents without notifying the processor
- Sound System
  - When recording, it is necessary to invalidate **cache regions** corresponding to memory where newly sampled sound has been placed




```
DC_InvalidateRange(mem_addr, num_bytes);    //Invalidates region in Data Cache
```

Libnds sound API: Start sound play-back

- Play-back stored samples
  - int soundPlaySample(const void* buffer, SoundFormat format, u32 buffSize, u16 freq, u8 volume, u8 panning, bool loop, u16 loopPoint)
    - volume: (min) ← 0…127 → (max)
    - panning: (left) ← 0…127 → (right)
    - loop: if true the buffer will be played in a loop starting in the sample loopPoint
    - **Returns** handler (integer ID) to change other parameters using application programmer interface (API) functions

```
typedef enum {
    SoundFormat_16Bit = 1,
    SoundFormat_8Bit = 0,
    SoundFormat_PSG = 3,
    SoundFormat_ADPCM = 2
}SoundFormat;
```

Libnds sound API: Stop sound and modify parameters

- Stop sound
  - void soundKill(int soundId)
- Pause / resume sound
  - void soundPause(int soundId)
  - void soundResume(int soundId)
- Change frequency, panning or volume
  - void soundSetVolume(int soundId, u8 volume)
  - void soundSetPan(int soundId, u8 pan)
  - void soundSetFreq(int soundId, u16 freq)

Example: Libnds sound API

- Store up to 5 seconds of sound sampled at 8 KHz when key A is pressed and play it back when key B is pressed

1. Defines and buffer declarations

```
//Some defines
#define RECORDING_TIME      5                       //seconds
#define SAMPLING_RATE       (1<<13)                 //samples/second (8 KHz)
#define PLAY_BUFFER_SIZE    (RECORDING_TIME*SAMPLING_RATE)  //Samples
#define MIC_BUFFER_SIZE     (SAMPLING_RATE / 16)            //Samples

//Microphone buffer
u16 mic_buffer[MIC_BUFFER_SIZE];

//Play-back buffer
u16 playback_buffer[PLAY_BUFFER_SIZE];

//Counter for samples already stored in the mic buffer
int num_samples = 0;
```

- Recording buffer filled 16 times per second **(the microphone handler will be called 32 times per second)**

Example: Libnds sound API

2. Implement microphone handler function
   - Need to invalidate cache contents!

```
void micHandler(void* data, int length)
{
    int bytes_to_copy;

    //Check if play-back buffer is full
    if(num_samples < PLAY_BUFFER_SIZE) {

        //Invalidate lines in cache (recommended to avoid inconsistencies)
        DC_InvalidateRange(data, length);

        //Number of bytes fitting in the buffer
        if(num_samples + (length / 2) <= PLAY_BUFFER_SIZE)
            bytes_to_copy = length;
        else
            bytes_to_copy = (PLAY_BUFFER_SIZE - num_samples) * 2;

        //Copy data to play-back buffer
        dmaCopy(data,  (u8*)(&playback_buffer[num_samples]), bytes_to_copy);

        //Update the number of samples recorded by the mic
        num_samples = num_samples + (bytes_to_copy/2);
    }
}
```

Example: Libnds sound API

3. Initialize sound library and start recording with key A

```
int main(void)
{
    consoleDemoInit();

    //Initialize sound system
    soundEnable();

    while(1)
    {
        int key;
        scanKeys();
        key = keysDown();

        //Record sound
        if(key & KEY_A) {
            printf("Recording...\n");
            //Restart index of the play-back buffer
            num_samples = 0;

            //Start recording
            soundMicRecord( mic_buffer,          //Buffer to store samples
                            MIC_BUFFER_SIZE*2,   //Size of the buffer in bytes
                            MicFormat_12Bit,     //Recording format
                            SAMPLING_RATE,       //Sampling frequency
                            micHandler);         //Microphone handler
        }
```

Example: Libnds sound API

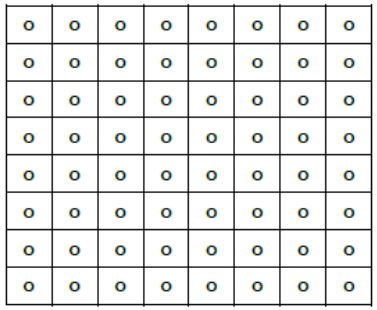
3. Play the recorded sound with key B

```
    //Playing back the recorded data
    if(key & KEY_B)
    {
        //Stop the mic recording if there is
        soundMicOff();

        printf("Playing...\n");
        //Start a single shot play-back
        soundPlaySample(playback_buffer,    //Buffer storing the samples
                        SoundFormat_16Bit,  //Format of the samples
                        num_samples*2, // Buffer size in bytes
                        SAMPLING_RATE,       //Sampling frequency
                        127,                //Speakers volume (maximum)
                        64,                 //Speakers panning (middle)
                        false,              // LOOP Play_back (NO)
                        0);                 // if LOOP, starting point in the buffer
    }

    swiWaitForVBlank();

} //End while(1) loop
```

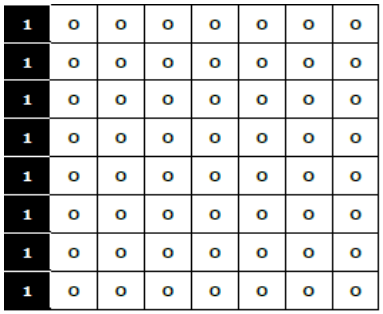
And what happens if we change the play-back frequency?

Practical Work 10: Creating a Piano Keyboard

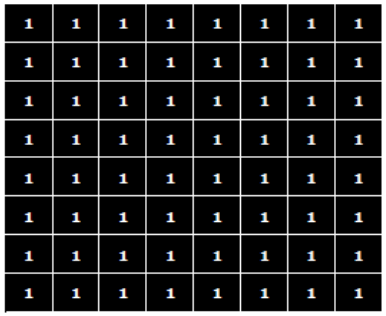
- Use of tiled background and MaxMod
  - 4 tiles, 4-bits pixel depth



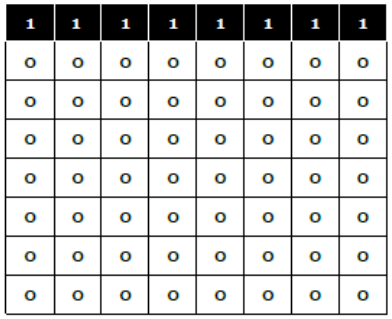
White tile



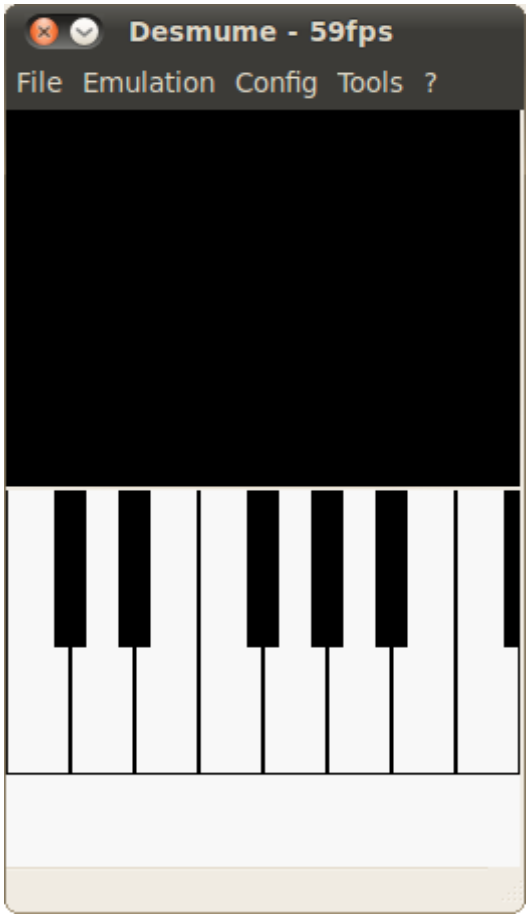
Semi-white left tile



Black tile



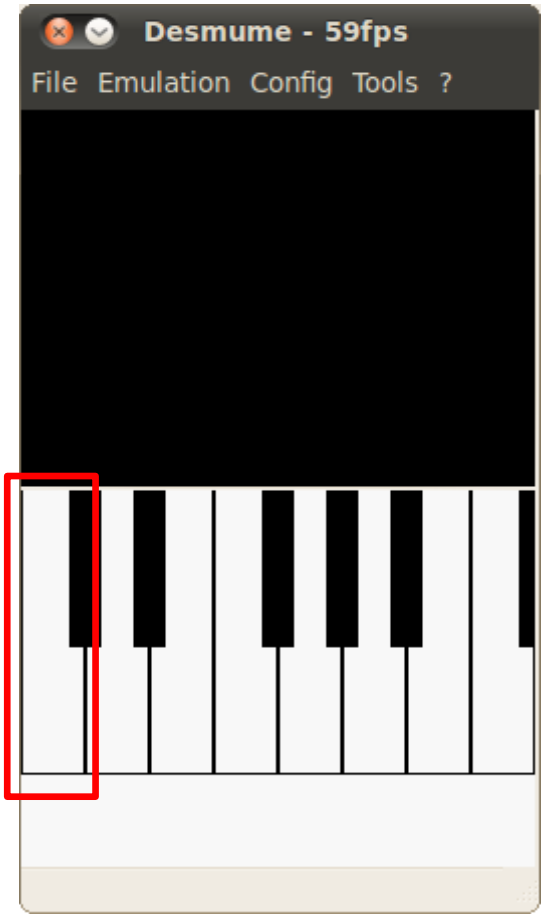
Semi-white up tile



Practical Work 10: Creating a Piano Keyboard

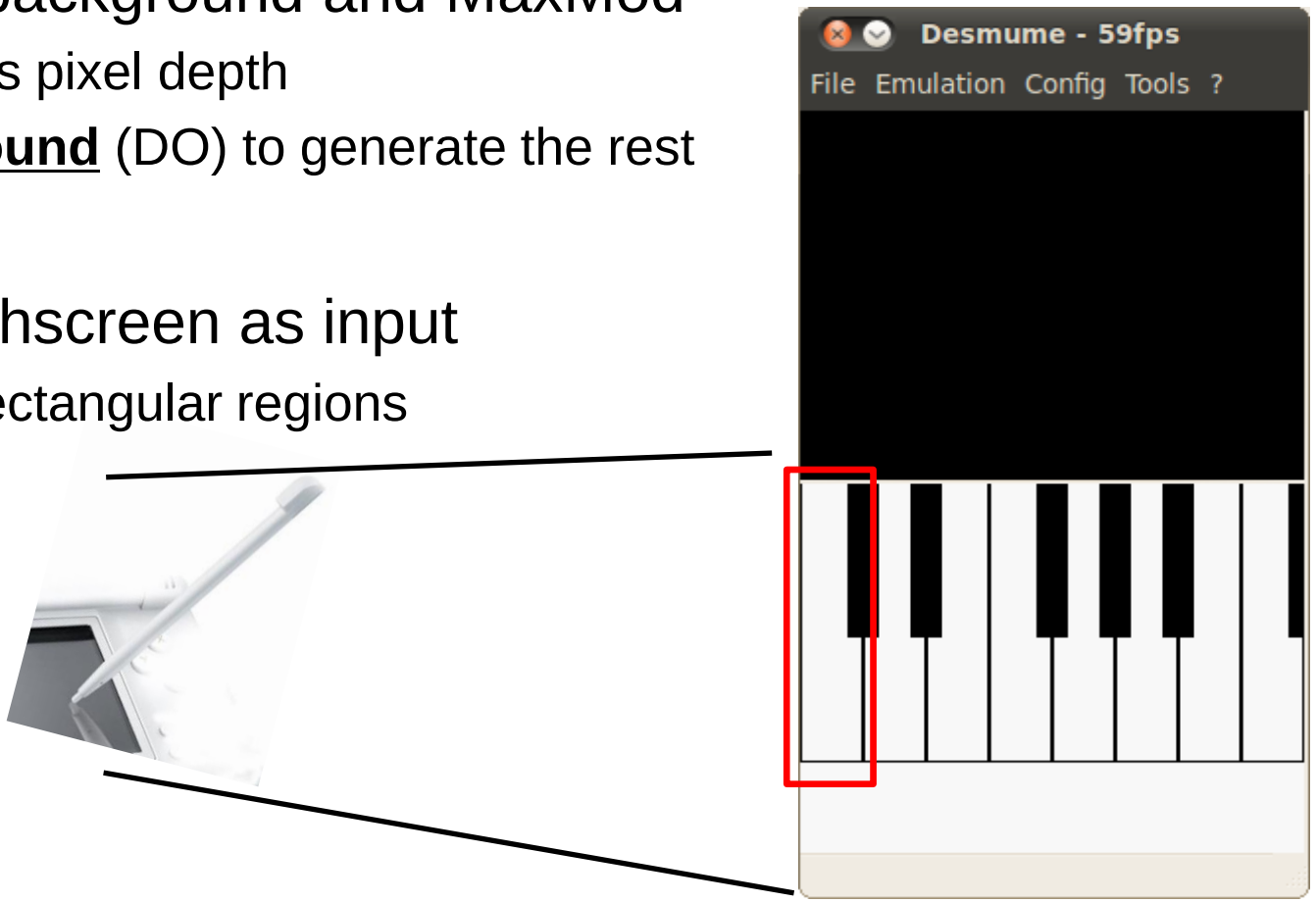
- Use of tiled background and MaxMod
  - 4 tiles, 4-bits pixel depth
  - Use **one sound** (DO) to generate the rest

Note	Freq. (Hz)
DO	261.626
DO#	277.183
RE	293.665
RE#	311.127
MI	329.628
FA	349.228
FA#	369.994
SOL	391.995
SOL#	415.305
LA	440.000
LA#	466.164
SI	493.883
DO2	523.251
DO2#	554.365



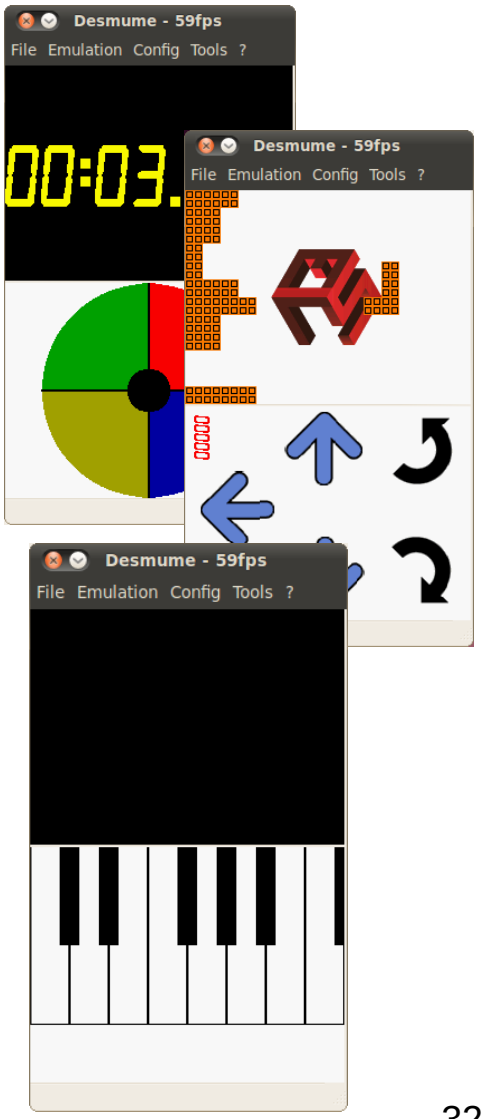
Practical Work 10: Creating a Piano Keyboard

- Use of tiled background and MaxMod
  - 4 tiles, 4-bits pixel depth
  - Use **one sound** (DO) to generate the rest
- Use the touchscreen as input
  - Checking rectangular regions



Practical Work 10: Sound in the Nintendo DS

- Exercises
  - Exercise 1 – Simon Game: Converting files
  - Exercise 2 – Simon Game: Initializing the library and introducing music
  - Exercise 3 – Simon Game: Adding sound effects
  - Exercise 4 – Tetris Game: Background music
  - Exercise 5 – Tetris Game: Adding sound effects
  - *Exercise 6 – Piano: Graphics part
  - *Exercise 7 – Piano: Touch tracking
  - *Exercise 8 – Piano: Playing key sounds in the simulator
  - *Exercise 9 – Piano: Playing key sounds in the device



* Additional exercises

Questions?

Let’s play sounds in the NDS!