Topic 3: (Part C)
I/O and Peripheral Devices Management
KEYS and TOUCHSCREEN in the Nintendo DS

Systèmes Embarqués Microprogrammés

Prof. David Atienza Alonso, SEL-STI

Content of Session

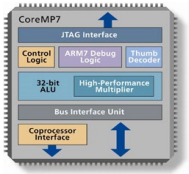
- Use of controls in the NDS: keys and touchscreen
  - Detection of events in controls and I/O handling modes
  - Polling for keys and touchscreen: libnds
  - Efficient keys management: interrupts triggers and identification
  - Tracking complex shapes using the touchscreen

- Using controls and graphics in the NDS
  - Developing a basic Paint tool
  - Try to be the fastest with colors: Simon game
  - Adding keypad and touchscreen control to the Tetris

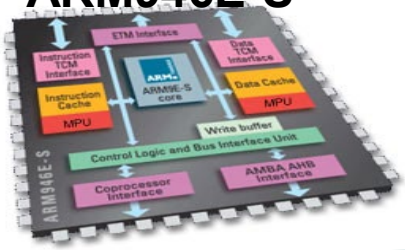
I/O subsystem management on the NDS: Keys and touchscreen

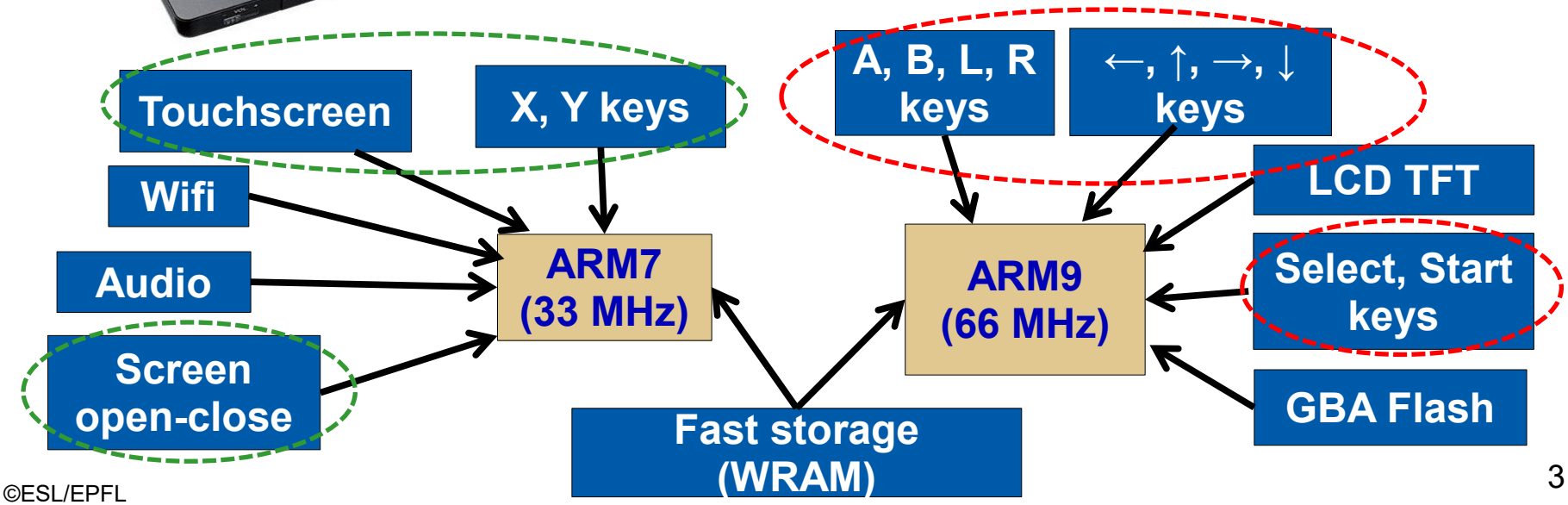
- The two 32-bit ARM cores manage together the keys and touchscreen
  - ARM 946E-S: most of the keys
  - ARM 7TDMI-S: touchscreen, screen open-close and X-Y keys

ARM7TDMI-S



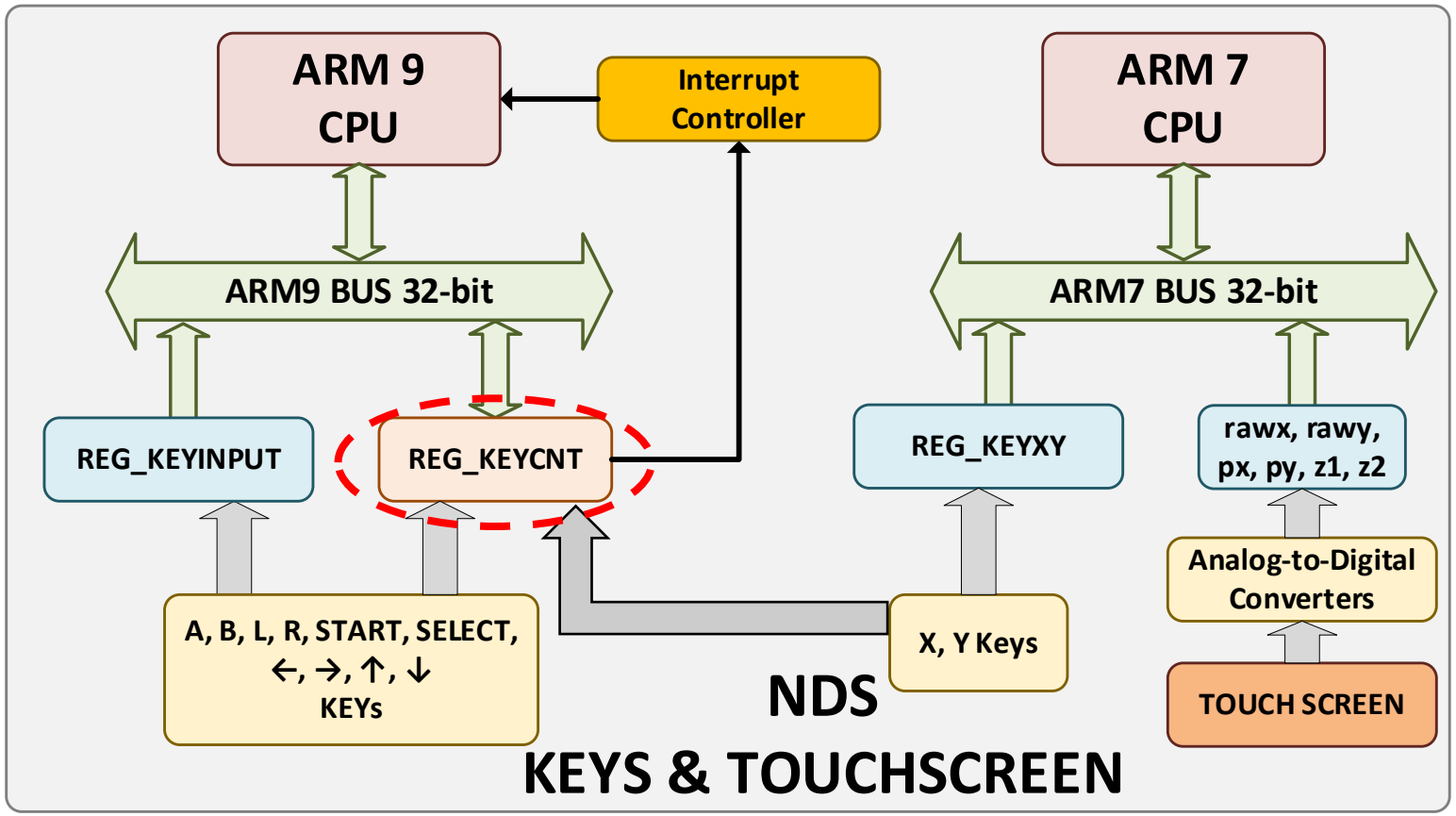
ARM946E-S





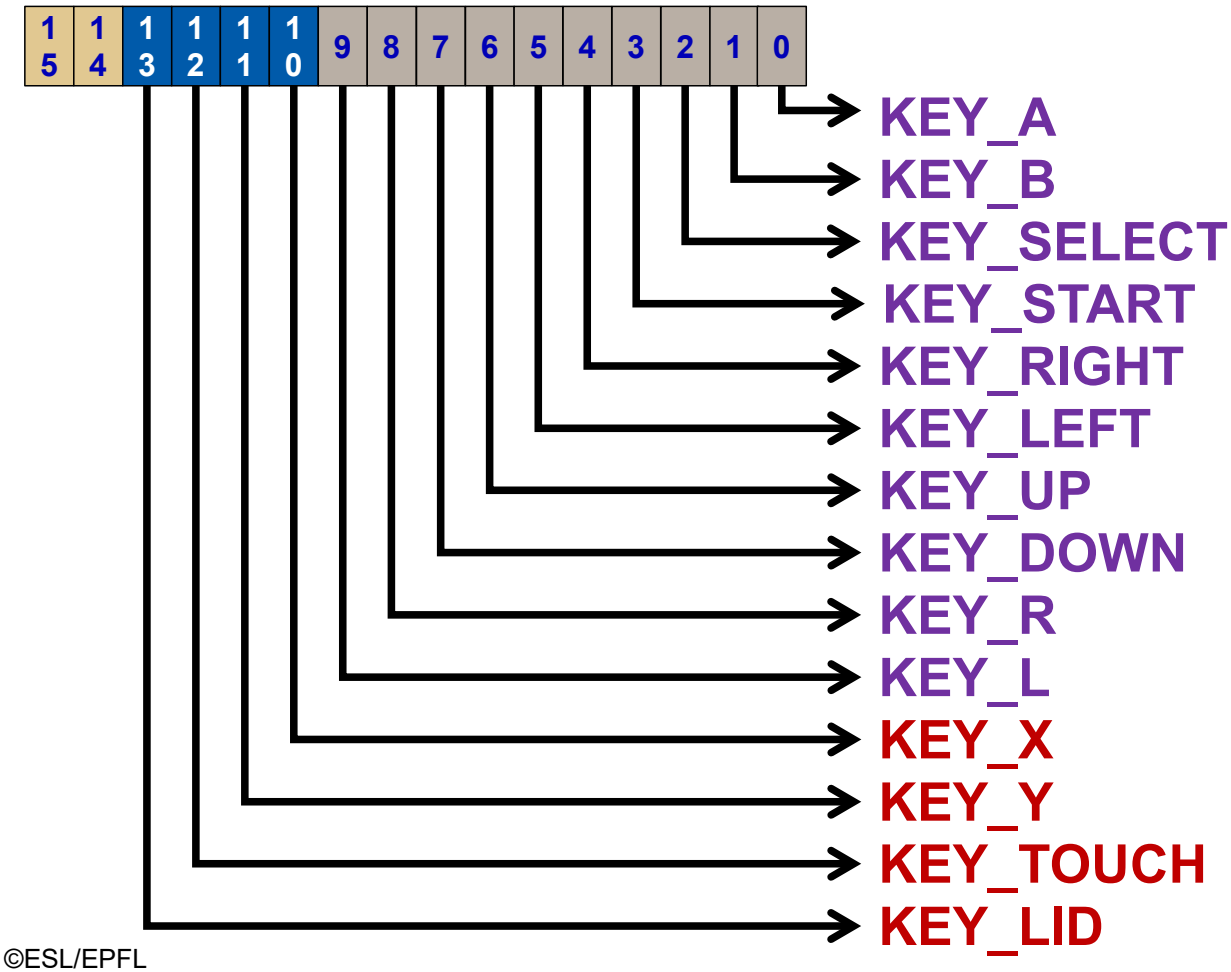
Keys and touchscreen on NDS

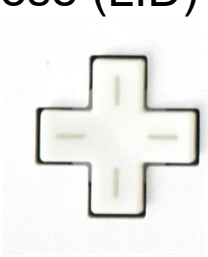
- The ARM9 and ARM7 manage the keys and touchscreen together

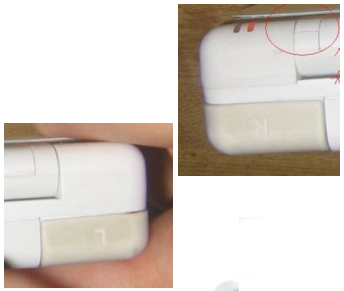


NDS keys/touchscreen identification: Specialized I/O register

- Events detected are configured in a 16-bit special register: **REG_KEYCNT**
  - **ARM 946E-S**: A, B, Select, Start, Right/Left/Up/Down, and R/L
  - **ARM 7TDMI-S**: X, Y, touchscreen, and screen open-close (LID)







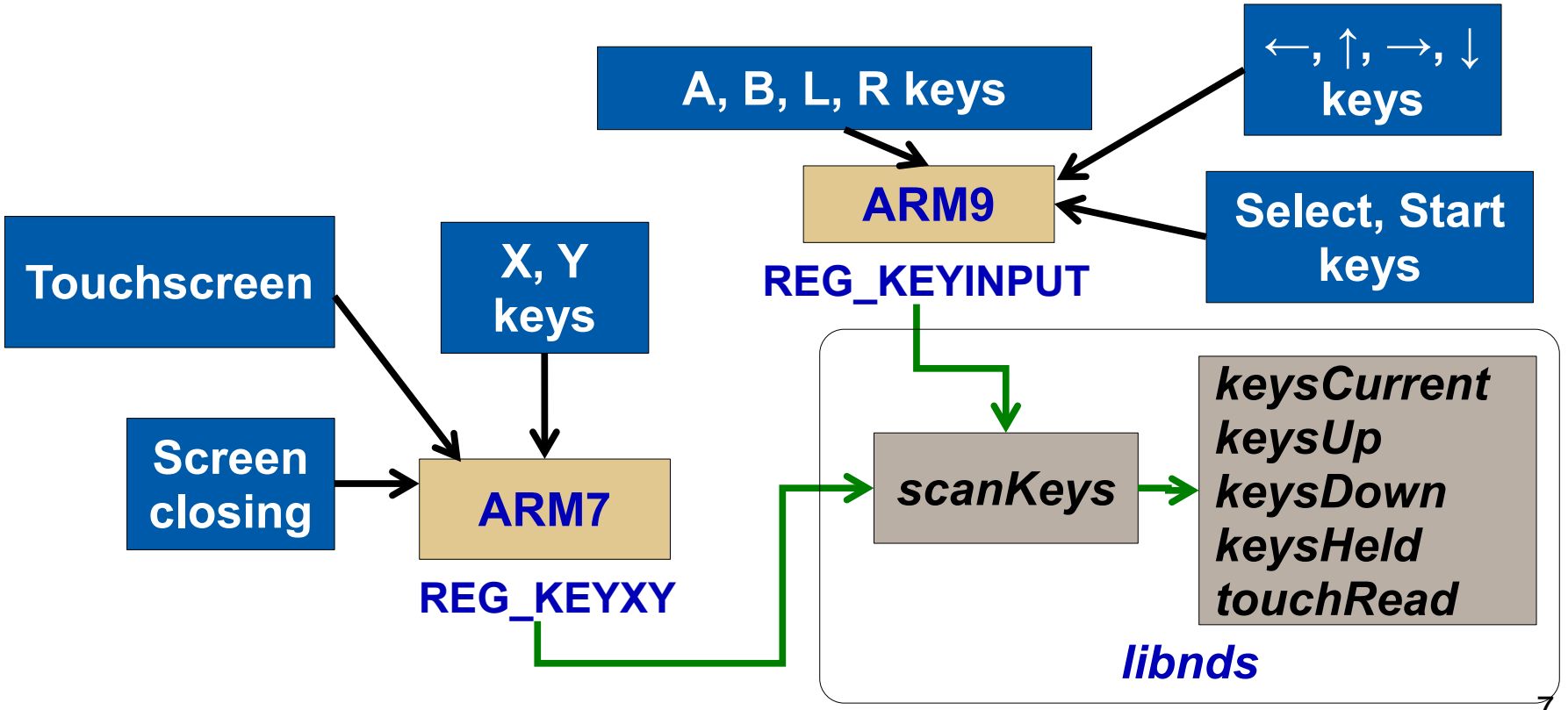
Synchronization of keys/touchscreen and NDS processors

- Keys: the two basic mechanisms for I/O syncronization exist
  - I/O interrupts: directly managing the REG_KEYCNT register
    - setup interrupts using bits manipulation of the register using C code
  - Polling: programmed I/O with active waiting, methods of libnds:
    - void ***scanKeys***() : scans and stores pressed keys
    - uint32 ***keysHeld***() : returns keys pressed and held
    - uint32 ***keysDown***() : returns keys just pressed
    - uint32 ***keysUp***() : returns keys just released

- Touchscreen: no buffering for pressed grids of the screen
  - Polling for I/O syncronization, methods of libnds:
    - void ***touchRead***(touchPosition* pos) : where was the touchscreen pressed?

Using the keys/touchscreen with polling: use of libnds methods

- Control of ARM9 and ARM7 interaction with keys/touchscreen in two steps
  1. Fetch keys currently pressed from both processors in internal memory buffers
  2. Use of the appropriate method according to desired event to check



Polling the keys: Check when pressing a single key

- Use of active loop: ***for(;;)***
- Just need to change ***keysHeld()*** to detect another event
  - ***keysDown()***, ***keysUp()***…
- Identify the pressed key using one of the macros: ***KEY_x***

```
#include <nds.h>
#include <stdio.h>

int main(void)
{
        consoleDemoInit();
        for(;;) {
                scanKeys();
                unsigned keys = keysHeld();
                swiWaitForVBlank();
                if (keys==KEY_A) printf("A");
        }
        return 0;
 }
```

Polling the keys: Check when pressing multiple keys

- It is the same method as before, just adding a case for each key…

```
#include <nds.h>
#include <stdio.h>
int main(void) {
        consoleDemoInit();
        for(;;) {
                scanKeys();
                unsigned keys = keysHeld();
                swiWaitForVBlank();
                printf("\x1b[23;0f%c%c%c%c %c%c%c%c %c%c",
                                (keys & KEY_DOWN ? 'v': '-'),
                                (keys & KEY_UP   ? '^': '-'),
                                (keys & KEY_LEFT ? '<': '-'),
                                (keys & KEY_RIGHT? '>': '-'),
                                (keys & KEY_A ? 'A': '-'),
                                (keys & KEY_B ? 'B': '-'),
                                (keys & KEY_X ? 'X': '-'),
                                (keys & KEY_Y ? 'Y': '-'),
                                (keys & KEY_L ? 'L': '-'),
                                (keys & KEY_R ? 'R': '-'));
        }
}
```

Polling the touchscreen

- Idem as with keys, get and store registers data: ***scanKeys()***
- Use ***touchRead(****touchPosition* pos****)*** to poll touchscreen
  - a ***touchPosition*** structure must be declared and provided as first argument to the function

```
typedef struct {
    u16 rawx; //!< Raw x value from the A2D
    u16 rawy; //!< Raw x value from the A2D
    u16 px;   //!< Processes pixel X value (0-255)
    u16 py;   //!< Processes pixel Y value (0-191)
    u16 z1;   //!< Raw cross panel resistance
    u16 z2;   //!< Raw cross panel resistance
} touchPosition;
```

- Fields ***px*** and ***py*** store the touched pixel coordinates
  - (0,0) is returned when the screen is not touched
  - The value of px and py is in the range of the screen size (0..254, 0..191)

Polling the touchscreen: Identify and print position on LCD screen

```
#include <nds.h>

int main(void) {
        consoleDemoInit();
        for(;;) {
                swiWaitForVBlank();
                scanKeys();
                unsigned held = keysHeld();
                if (held & KEY_TOUCH) {
                        touchPosition touch;
                        touchRead(&touch);
                        printf("\x1b[6;5HTouch x = %04X, %04X\n",
                                        touch.rawx, touch.px);
                        printf("\x1b[7;5HTouch y = %04X, %04X\n",
                                        touch.rawy, touch.py);
                }
        }
}
```

Polling the touchscreen: A Paint tool built in four steps

1. The framebuffer mode can be used as a canvas
   - FB mode only available in the Main video engine,
   - but touchscreen is controlled by default using the Sub video engine
2. Screens swapped by configuring the corresponding register
   - Toggle bit 15 in the register ***REG_POWERCNT***
3. The full canvas is whitened: all pixels set to white

```
int main(void) {

    //Configure the Engine in FB mode
    REG_DISPCNT = MODE_FB0;
    //Configure the VRAM block
    VRAM_A_CR = VRAM_ENABLE | VRAM_A_LCD;
    //Swap the LCD
    REG_POWERCNT ^= BIT(15);
    //Fill the canvas with WHITE color
    memset(VRAM_A,0xFF,256*192*2);
```

Polling the touchscreen: A Paint tool built in four steps

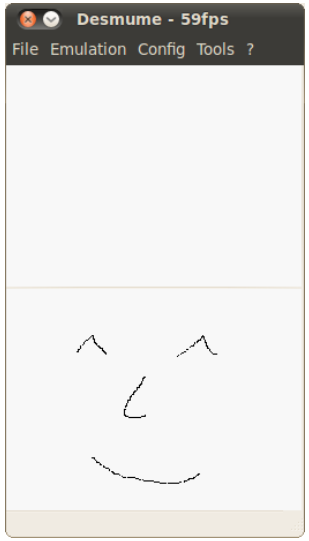
4. A ***touchPosition*** structure is used to poll the touchscreen
   - Infinite loop polling the touchscreen
   - The touched pixel, different from (0,0), is set to black

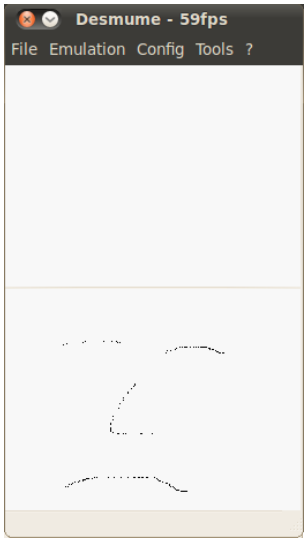
```
    //Declaration of the touch struct
    touchPosition touch;
    while(1) {scanKeys();
        //Poll the touch-screen
        touchRead(&touch);
        //If the touch is different to (0,0), change the color of the
        if(touch.px || touch.py)
            VRAM_A[touch.py * 256 + touch.px] = ARGB16(1,0,0,0);

        swiWaitForVBlank();
    }
}
```

Polling the touchscreen: A Paint tool built in four steps

- But drawbacks exist due to the speed of touchscreen device
  - 60 points read/sec with ***swiForVBlank()***
  - Plot diagonal line from one corner to opposite one (>256 points), which would take more than 4 seconds!





Slow drawing: OK!

Fast drawing: not OK!

- **Solution:** Interpolate values between two points and set the color in the intermediate pixels

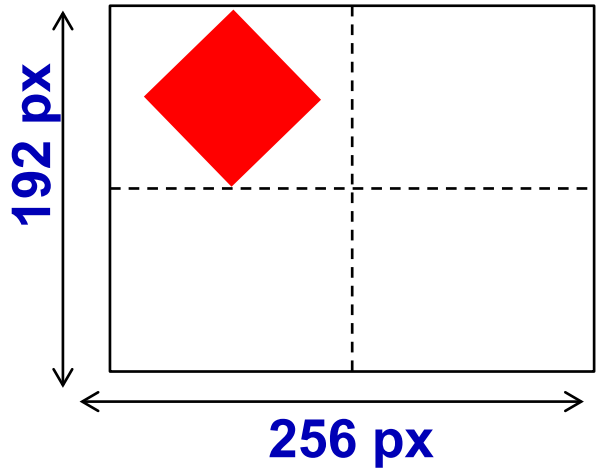
Tracking complex shapes using the touchscreen

- Active rectangular areas: typical in simple graphical user interface (GUI) with touch surface divided into regions
  - Easy to check with inequalities: >, <, <= or >=

- More complex shapes can be tracked: circles, ellipses, rhombus, etc. But additional information is needed
  - Starting point: changes in the coordinates reference
    - Example: A circle centered in the screen with coordinates (127,95)
  - Characteristics parameters of specific shape must be known
    - Example: The radius of the circle

Tracking more complex shapes using the touchscreen: A rhombus

- Centered in upper left quadrant of the screen
  - Center in coordinates (63, 47)



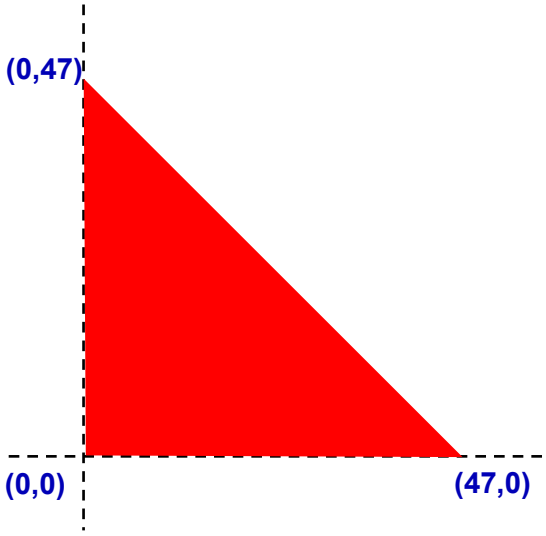
- Two steps:
  1. Obtain the touched pixel and move the coordinate reference

     x = px - 63;

     y = 47 - py;

  2. Check if the point is inside the rhombus

     abs(y) + abs(x) <= 47



For other shapes the process is again shape specific, so the steps must be developed on a case-per-case basis

Using the keys with interrupts: configuring interrupt triggers

- Keys trigger an interrupt when a specific key is pressed by configuring ***REG_KEYCNT***
- ***REG_KEYCNT*** can be configured in two modes
  - One key triggers an interrupt
    - Bit 14: Requests an interrupt
  - A combination of keys pressed together triggers an interrupt
    - Bit 15: AND of all the keys to trigger the interrupt
- Examples:
  - Key A, Left or Start trigger an interrupt when any of them is pressed

    REG_KEYCNT = (1<<14) | KEY_A | KEY_LEFT | KEY_START;
  - Key A and B trigger an interrupt only when pressed together

    REG_KEYCNT = (1<<14) | KEY_A | KEY_B | (1<<15);

Using the keys with interrupts: identifying the pressed key/s

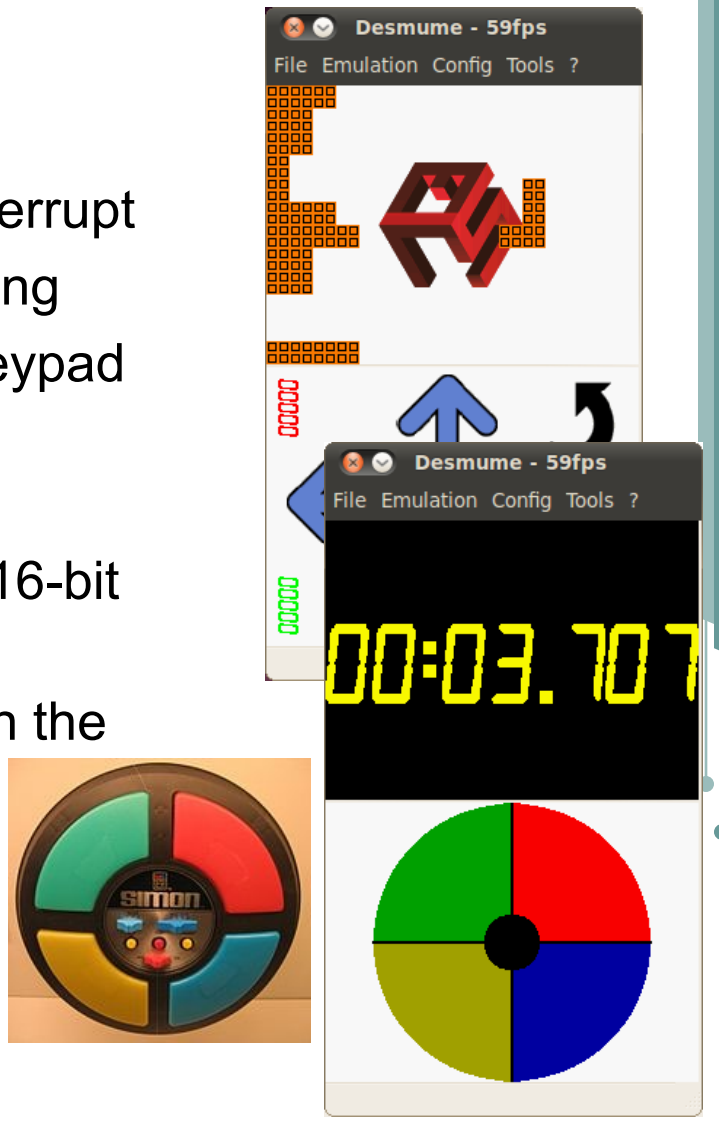
- Advanced libnds functionality not available with interrupts
  - The API provided by libnds will not work properly in the interrupt service routine (ISR)
    - ***keysHeld()***, ***keysDown()*** or ***keysUp()*** do not return correct values.
- Necessary to read the register ***REG_KEYINPUT*** and complement it to identify the pressed key with inverse logic
  - A clear bit (zero) means that the key is pressed
- Example: Check if the *START* key triggered the interrupt

```
void ISR_Keys()
{
    u16 keys = ~(REG_KEYINPUT);

    //START KEY = Restart the game
    if(keys & KEY_START)
```

Practical Work 9: The keypad and the touchscreen

- Exercises
  - Exercise 1 – Read keypad by polling
  - Exercise 2 – Read keypad triggering an interrupt
  - Exercise 3 – Read the touchscreen by polling
  - Exercise 4 – Assign Tetris actions to the keypad
  - Exercise 5 – Assign Tetris actions to the touchscreen
  - *Exercise 6 – Simon game: Graphics with 16-bit palettes
  - *Exercise 7 – Simon game: Start game with the keypad (START button)
  - *Exercise 8 – Simon game: Tracking the touchscreen in complex areas



* Additional exercises

EPFL

Questions?

Let’s use keys and touchscreen in the NDS!