i.MX6 Solo X

Take advantage of the heterogeneous architecture

DwF Rochester - 10/01/2015

Sébastien Ronsse

Solutions Architect

TABLE OF CONTENTS

Introduction

ABOUT THE AUTHOR

- Sébastien Ronsse
- Solutions Architect @ **Adeneo Embedded**
- Bellevue, WA
- **sronsse@adeneo-embedded.us**

I.MX6 SOLO X

Latest of the i.MX6 family:

- Hybrid architecture
  - 1x Cortex A9
  - 1x Cortex M4
- Advanced HW IP:
  - CAN, Dual Gigabit Ethernet, USB, etc
  - 12 bits ADC
  - PCIe
  - Security
  - GPU
  - ...

Why i.MX 6 Solo X?

YOUR PRODUCT

Your product requirements are:

- I want "fancy" features: WiFi, UI, Multimedia, you name it.
- I want real time capabilities.
- I want runtime robustness.
- I want all of this, but cheap.

WITHOUT SOLO X

You can use a high-end SoC (like i.MX6 dual or quad).

- I want "fancy" features: WiFi, UI, Multimedia, name it.

⇒ **Linux on CPU0**

- I want real time capabilities.

⇒ **Linux + Xenomai on CPU1**

- I want runtime robustness.

⇒ Possible, but hard.

- I want all of this, but cheap.

⇒ Yes, it is cheap - but could it be cheaper?

WITHOUT SOLO X

This is not optimal because:

- Real-time part doesn't need high end CPU
- Xenomai integration can be time consuming
- Likely to have a higher power usage
- There are cheaper solutions

WITH I.MX6 SOLO X

The i.MX6 Solo X is the perfect match for your needs:

- I want "fancy" features: WiFi, UI, Multimedia, name it.

⇒ **Linux on Cortex A9**

- I want real time capabilities.

⇒ **RTOS on Cortex M4**

- I want runtime robustness.

⇒ **HW designed for this purpose**

- I want all of this, but cheap.

⇒ **i.MX6 Solo X is cheaper!**

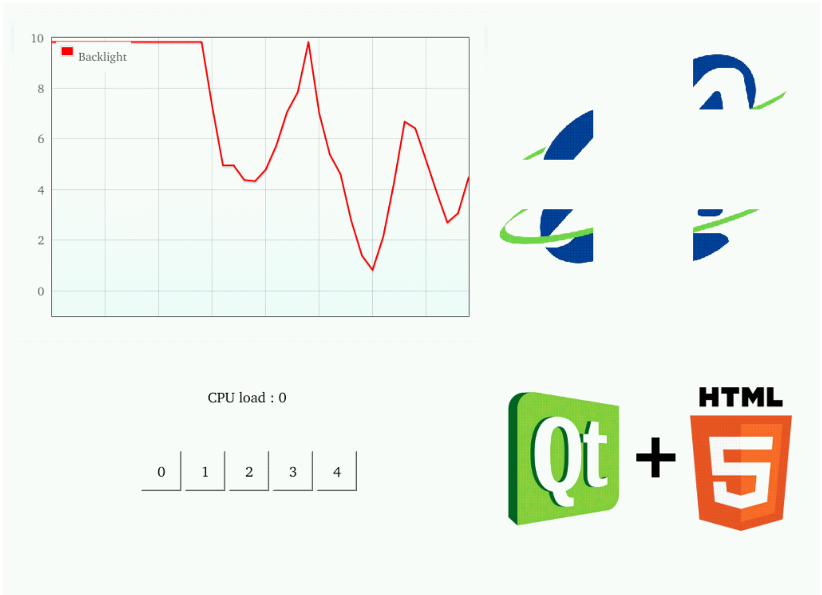
Use-case: Our demo

OUR DEMO

Our demo displays the advantage of Solo X, on the reference board (Sabre SD)

- "Fancy" features: QT and HTML5 UI, with Node.JS
- Realtime processing placeholder
  - Accelerometer value drive LVDS backlight
- Load CPU "feature": Highlights the need of separation between the UI and the critical task

OUR DEMO



CPU load : 0

IMPLEMENTATION

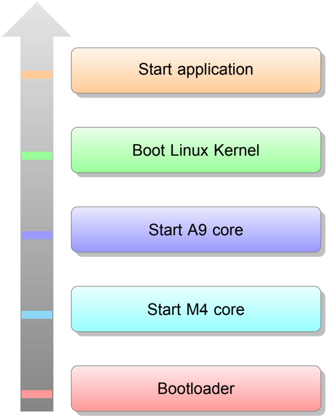
Our demo uses:

- MQX on Cortex M4
  - Simple application, using drivers provided with MQX for SDB
  - Custom driver for PWM control
  - Read accelerometer X-axis value, and set PWM duty cycle
- Linux on Cortex A9 (Yocto)
  - Standard Linux
  - Start QT application with HTML5 view and Node.js features

Take advantage of Solo X

BOOT PROCESS IN SOLO X

Standard boot process of Linux, with a twist:

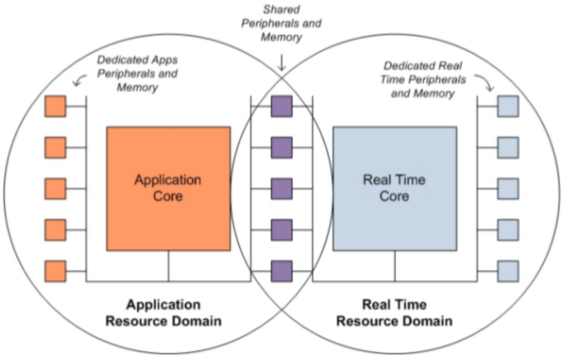


M4 AND A9 INTERACTION

- Master (A9)/Slave (M4) organization.
  - A9 starts clocks and puts M4 out of reset
- M4 and A9 access the same peripherals
  - Separate interrupt controllers.
  - Resource Domain Controller: Ensures safe access to shared resources and allow access restrictions for peripherals/memory.
- The MultiCore Communication, MCC, to share data between cores.

RESOURCE DOMAIN CONTROLLER

Resource Domain Controller: Big advantage of the Solo X.


Shared Peripherals and Memory
Dedicated Apps Peripherals and Memory
Dedicated Real Time Peripherals and Memory
Application Core
Real Time Core
Application Resource Domain
Real Time Resource Domain

Source: i.MX6 Solo X TRM

DEDICATED ACCESS

Dedicated peripherals and memory:

- Enhance security
- Remove programming error concerns
- Easy to implement

DEDICATED ACCESS

Different parts of the RDC (fine control):

- Domain ID (0-3): Set of cores, bus, IP
  - M4 Core, A9 Core, CSI, SDMA, uSDHC, etc
  - At reset, all of them are part of Domain ID 0.
- Peripheral Domain Access Permissions: Permission (R/W) for each domain
  - At reset, R/W allowed with no safe access for shared peripherals.
- Memory Region Control (54 sections): Permission (R/W) for each domain, on a specific memory section.
  - Disabled at reset.

DEDICATED ACCESS: EXAMPLE

In our demo, the M4 controls the PWM4 (backlight), and we reserve I2C3 (accelerometer).

- Define Domain ID for M4 core to 1 (RDC_MDA1).
- Allow R/W permission for I2C3 only for Domain ID 1 (RDC_PDAP44).
- Allow R/W permission for PWM4 only for Domain ID 1 (RDC_PDAP3) and read permission for Domain ID 0.
  - We keep the read permission on ID 0 to allow A9 to read the current value of the duty cycle.
- And that's it! The I2C3 and PWM4 is now reserved for the M4.

DEDICATED ACCESS: EXAMPLE

Use it!

Easy to implement on MQX, and it will make your design robust! For example, for the I2C3 (R/W only for M4).

- Implemented by default in MQX for Solo X with `_bsp_rdc_init`.
- For R/W control, add your peripheral to the list `rdc_peripheral_m4` (RDC_PDAP_I2C3_ID on our example).
- You're done! For a more complex implementation, the code within MQX is a great start.

DEDICATED ACCESS: EXAMPLE

By default, these peripherals are reserved for M4 (not locked, 0x0000000C):

```
static uint8_t rdc_peripheral_m4[] = {
    RDC_PDAP_UART2_ID,
    RDC_PDAP_I2C3_ID,
    RDC_PDAP_ECSPI4_ID,
    RDC_PDAP_ECSPI5_ID,
    RDC_PDAP_ADC1_ID,
    RDC_PDAP_ADC2_ID,
    RDC_PDAP_CAN1_ID,
    RDC_PDAP_CAN2_ID,
    RDC_PDAP_EPIT1_ID,
    RDC_PDAP_EPIT2_ID,
    RDC_PDAP_WDOG3_ID
};
```

DEDICATED ACCESS: EXAMPLE

In our application (Domain 1 R/W, Domain 0 R, and lock the value):

```
RDC_MemMapPtr rdc = RDC_BASE_PTR;
rdc->PDAP[RDC_PDAP_PWM4_ID] = 0x8000000E;
```

Source: commons.wikimedia.org

SHARED ACCESS

For the shared peripherals:

- Safe sharing: HW semaphore to ensure exclusive access to peripherals.
- If the semaphore is enabled on a peripheral, the domain needs to lock the HW semaphore before accessing the peripheral.
- Disabled by default.

SOLO X IMPLEMENTATION

- Two blocks of 64 gates each (RDC_SEMAPHORE(1/2)_GATE(1-64))
- One gate per peripheral
- Enable/Disable per peripheral, apply to all domains

SOLO X IMPLEMENTATION

To lock PWM4 for example (to be sure to read the current duty cycle)

- Enforce semaphore control.
  - `rdc->PDAP[RDC_PDAP_PWM4_ID] |= 0x40000000;`
- Find the proper semaphore block/gate.
  - `RDC_SEMAPHORE1_GATE3`
- To lock the gate, write `master_index+1` (5 for M4, 1 for A9) into Gate Finite State Machine (GTFSM).
  - `rdc_semi1->GATE[RDC_PDAP_PWM4_ID] = 0x6`
- To unlock the gate, write `0` into Gate Finite State Machine (GTFSM).
  - **Only master index used to lock the device can unlock it**

SOLO X IMPLEMENTATION

MQX offer API to deal with the safe sharing:

```
_bsp_rdc_sema42_lock(RDC_PDAP_PWM4_ID);
_bsp_rdc_sema42_unlock(RDC_PDAP_PWM4_ID);
```

Linux side limitation...

There is currently no support for RDC semaphore within Linux.

... But

The implementation is simple, and can be easily added to a driver

MULTICORE COMMUNICATION

Sharing data between cores:

- Useful to "export" data out of the M4, for example CAN frames.
- MCC examples (Linux/MQX available within MQX release)
- Use hardware semaphores (SEMA4 module) and shared memory.

MULTICORE COMMUNICATION

Two implementations for MCC are available on the Linux side:

- TTYMCC: Communicate between cores as a serial interface
  - virtual_tty example in MQX
  - imx6sx-mcc-tty driver in Linux
- PingPong, FlexCan 'forwarding' implementation
  - flexcan, pingpong example in MQX
  - imx6sx-mcc-test driver in Linux
- These are good start for a custom implementation.

MU: MESSAGE UNIT

- Share messages (events) between cores.
- CPU-to-CPU interrupts.
- Control power states between cores.

MU: MESSAGE UNIT

The current use of it is:

- M4 requests A9 to change clock parents.
- Switch between low power mode and normal state.
- Implemented within MQX and Linux (CAN receiver in low-power mode, etc).

LINUX INTEGRATION

Linux kernel is "M4 aware":

- Peripherals dedicated to M4 are disabled by device-tree
- MU driver in Linux `fsl,imx6sx-mu`
- `fsl,shared-clks-*` device-tree info, to enable clock/power management.
  - Implemented on both sides (`bsp_clk_shared_mgmt.c` on MQX, `clk-imx6sx.c` on Linux)
- Shared memory for MCC usage via `linux,usable-memory` (1MB by default)

DEVICE TREE INTEGRATION

```
memory {
                linux,usable-memory = <0x80000000 0x3ff00000>;
                reg = <0x80000000 0x40000000>;
};
[...]
&i2c3 {
        status = "disabled";
};
[...]
&clks {
        fsl,shared-clks-number = <0x23>;
        fsl,shared-clks-index = <IMX6SX_CLK_I2C3 [...]>;
        fsl,shared-mem-addr = <0x91F000>;
        fsl,shared-mem-size = <0x1000>;
};
```

YOUR SCENARIO

- Out-of-box, everything already works.
- A lot of examples (including source) are available to start your own implementation (on MQX and Linux).
- Take the most out of this hybrid architecture!

Questions?