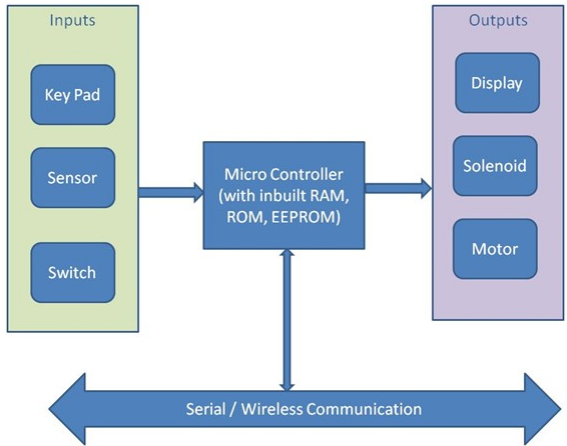
2. Mikrokontroléry v IoT I. – Operační systém. Vnitřní periferie.

B0B37NSI – Návrh systémů IoT

Stanislav Vítek

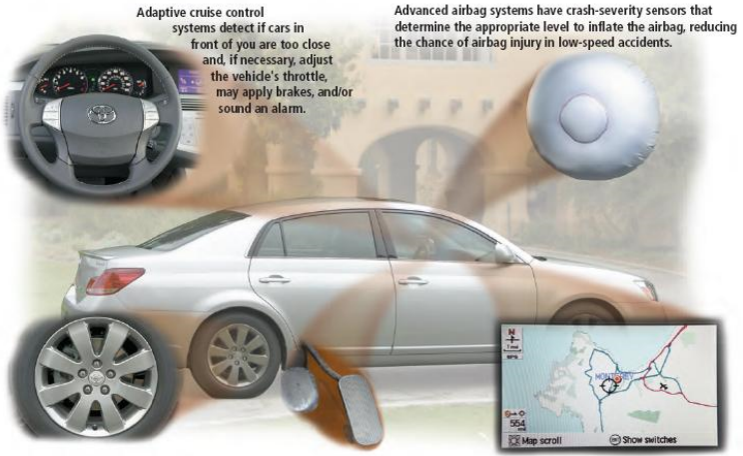
Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Komponenty IoT zařízení



Počítače: k všeobecnému použití (general purpose) nebo dedikované

Vestavné systémy v automobilitě 1/2

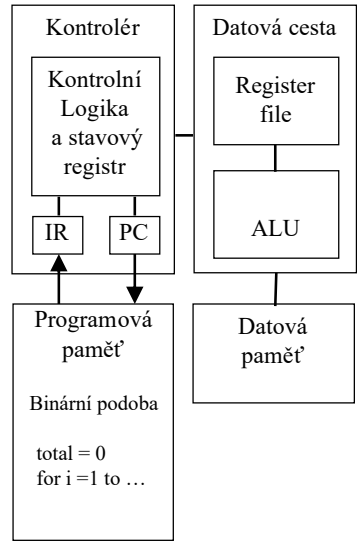


Vestavné systémy v automobilitě 2/2

Dnešní automobily vyšší střední třídy mívají více než 100 různých procesorů: Bezpečnostní pásy; přístrojová deska; ovládání motoru; ABS; automatická kontrola stability; navigační systém; informační a zábavní systém; systém zabraňující kolizím; kontrola tlaku v pneumatikách; hlídání jízdního pruhu; adaptivní tempomat; klimatizace; ovládání airbagů; elektrické ovládání oken a centrální zamykání; parkovací asistent; automatické ovládání stěračů; alarm a imobilizér; elektricky ovládaná sedadla; elektrický posilovač řízení; elektronická převodovka; aktivní odpružení.

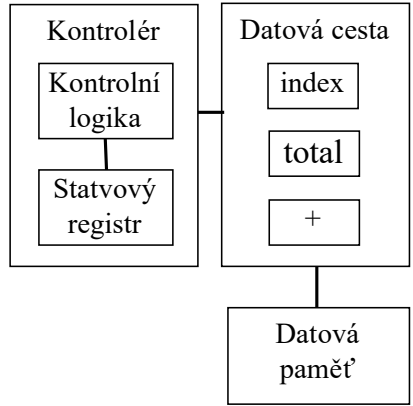
Obecné procesory

- Programovatelné zařízení
  - mikroprocesor
  - mikrokontrolér
- Hlavní části
  - Programová a datová paměť
  - Obecná datová cesta
  - Výbava registry
  - Obecná ALU
- Aplikačně-specifické procesory (ASIC)
  - Optimalizovaná datová cesta
  - Speciální funkční bloky



Dedikované procesory

- Jednoúčelový digitální obvod
- Hlavní části
  - Komponenty nutné k provádějí jednoho programu
  - Nemá programovou paměť
- Výhody
  - Malý
  - Rychlý
  - Nízká spotřeba



Vestavné systémy

- Dedikovaná funkcionalita
- Provoz v reálném čase
- Malé rozměry a nízká hmotnost
- Nízká spotřeba energie
- Drsné prostředí
- Provoz kritický z hlediska bezpečnosti
- Cenově výhodné

Vestavné systémy vs. systémy pracující v reálném čase

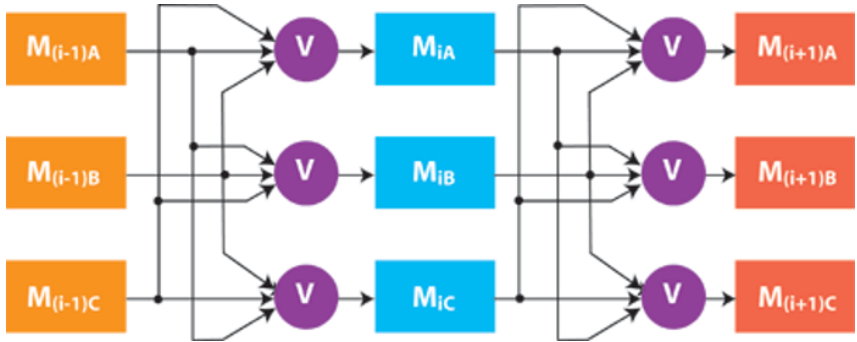
- Vestavný systém: vykonává omezený soubor specifických funkcí; často komunikuje se svým okolím.
- RT systém: správnost systému závisí nejen na logických výsledcích, ale také na čase, za který jsou výsledky vytvořeny.

Příklady

- Real-time vestavné:
  - Systémy kritické z hlediska bezpečnosti: řízení jaderného reaktoru, řízení letu
  - GPS, MP3 přehrávač, mobilní telefon
- Real-time, ale ne vestavné:
  - Plaforma pro burzovní operace
  - Skype, Youtube, Netflix
- Vestavné, ale ne real-time:
  - Domácí termostat, zavlažovací systém
  - Pračka, lednička

Charakteristiky real-time systémů

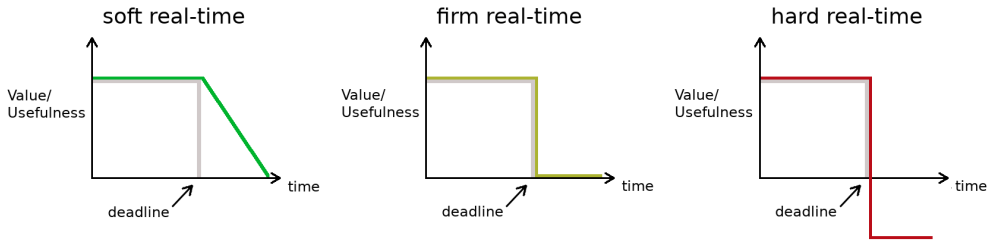
- Řízení událostí (reaktivní) vs. řízení podle času
- Požadavky na spolehlivost/odolnost proti poruchám (příklad: trojnásobná modulární redundance)
- Předvídatelnost
- Priority ve víceúlohových systémech



Klasifikace RT systémů 1/3

- Hard RT: reakce na vstupy musí přijít v požadovaném termínu. Např. systémy řízení letů.

  Real RT: navíc velmi krátká odezva, např. systém navádění raket
- Soft RT: termíny jsou důležité, ale systém bude správně fungovat i v případě, že budou termíny občas nedodrženy. Např. systém sběru dat.
- Firm RT: několik zmeškaných termínů nepovede k úplnému selhání, ale více než několik zmeškaných termínů může vést k úplnému nebo katastrofickému selhání systému.



Klasifikace RT systémů 2/3

Statické

- Lze předvídat časy příchodu úkolů
- Možnost statické analýzy (v době kompilace)
- Umožňuje dobré využití zdrojů (nízká doba nečinnosti procesorů)

Dynamické

- Nepředvídatelné časy příjezdu
- Statická analýza (v době kompilace) je možná pouze pro jednoduché případy
- Využití procesoru se dramaticky mění; je třeba navrhnout systém tak, aby zvládl "nejhorší případ".
- Je třeba se vyvarovat příliš zjednodušujících předpokladů, např. předpokladu, že všechny úlohy jsou nezávislé, i když je to nepravděpodobné.

Klasifikace RT systémů 3/3

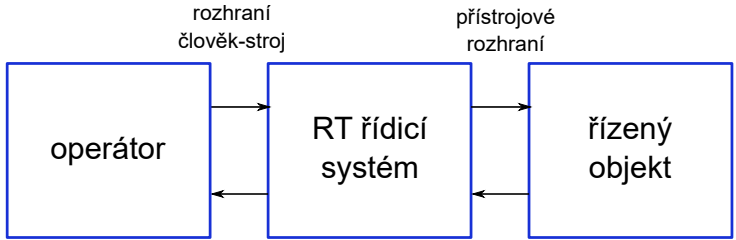
Periodické

- Každá úloha (nebo skupina úloh) se provádí opakovaně s určitou periodou.
- Umožňuje použití některých technik statické analýzy
- Odpovídá charakteristikám mnoha skutečných problémů
- Je možné mít úlohy s termíny menšími, rovnými nebo většími, než je jejich perioda – pozdější jsou obtížně zpracovatelné (tj. vyskytuje se více souběžných instancí úloh).

Neperiodické

- Nazývají se také sporadické, asynchronní nebo reaktivní.
- Vytváří dynamickou situaci
- Časově omezené intervaly příchodu jsou snadněji zvládnutelné
- Neomezené časové intervaly příchodu nelze u systémů s omezenými zdroji zvládnout

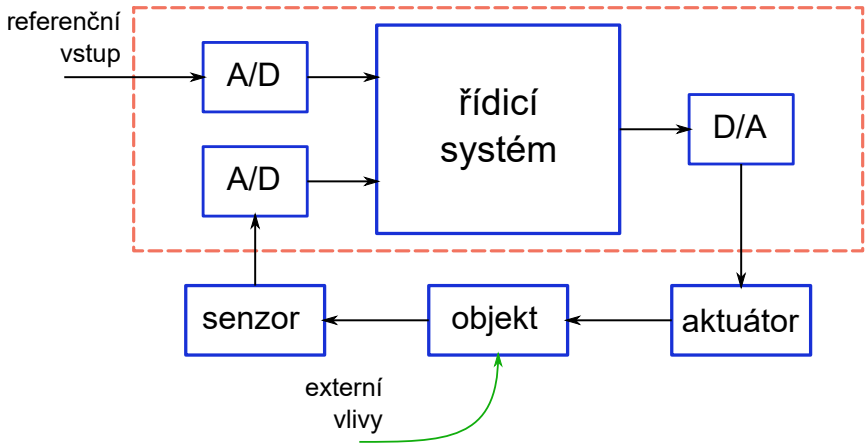
Řídicí systémy



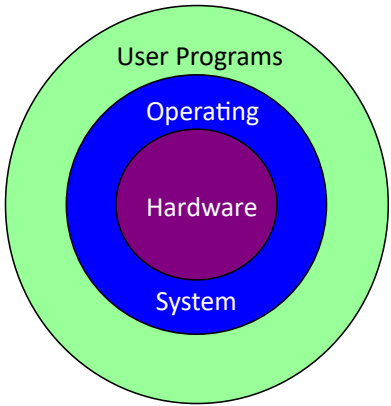
- Rozhraní člověk-stroj: vstupní zařízení, např. klávesnice, a výstupní zařízení, např. displej.
- Přístrojové rozhraní: snímače a akční členy, převádějí fyzikální signály na digitální data.
- Většina řídicích systémů je v tvrdém reálném čase
- Termíny jsou určeny řízeným objektem, tj. časovým chováním fyzikálního jevu (vstřikování paliva vs. ATM).

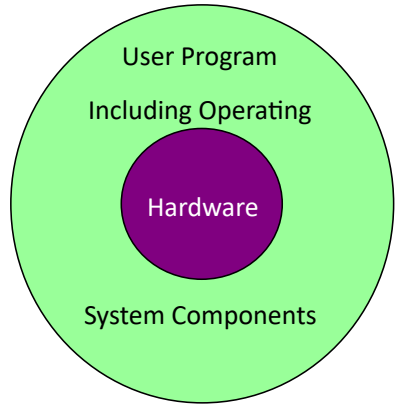
Příklad řídicího systému

- Jednoduchý systém s jedním senzorem a jedním aktuátorem



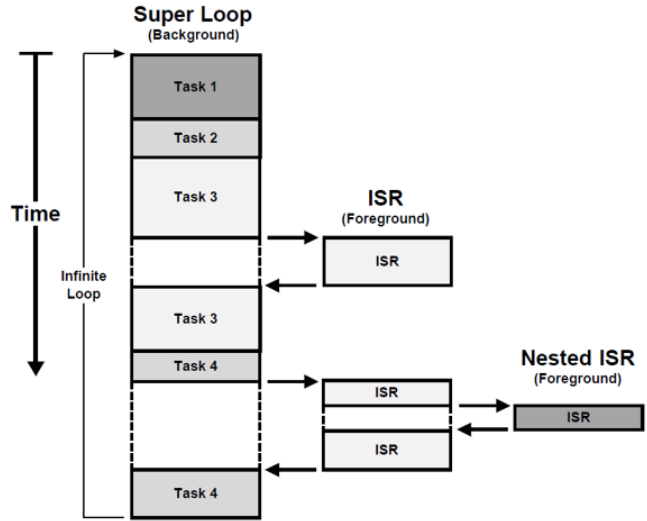
Operační systém?





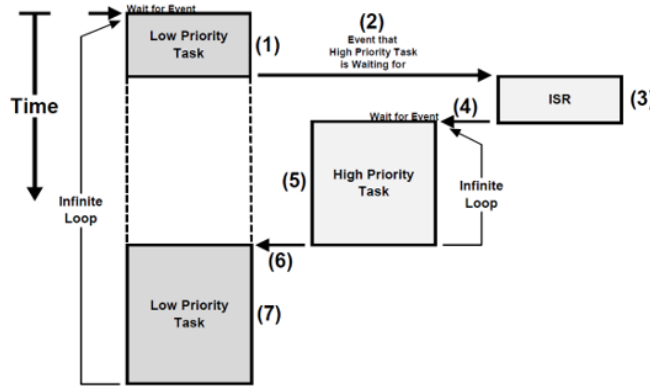
Běh systému

- Úroveň úlohy, úroveň přerušení
- Kritické operace se musí provádět na úrovni přerušení (není dobré)
- Doba odezvy/časování závisí na celé smyčce
- Změna kódu ovlivňuje časování
- Jednoduché, levné systémy



RT systémy

- Vzhledem k nutnosti reagovat na časové požadavky různých podnětů/odpovědí musí architektura systému umožňovat rychlé přepínání mezi zpracovateli podnětů.
- Vzhledem k různým prioritám, neznámému pořadí a různým časovým požadavkům různých podnětů jednoduchá sekvenční smyčka obvykle nevyhovuje.
- RT systémy se proto obvykle navrhují jako procesy spolupracující s jádrem reálného času, které tyto procesy řídí.



Plánovací strategie

- Nepreemptivní plánování
  - Jakmile je proces naplánován k provedení, běží až do dokončení nebo dokud není z nějakého důvodu zablokován (např. čekání na vstup/výstup).
- Preemptivní plánování
  - Provádění prováděných procesů může být zastaveno, pokud proces s vyšší prioritou vyžaduje obsluhu.
- Plánovací algoritmy
  - Round-robin
    - běžícímu procesu přiděluje kvantum času, po jeho uběhnutí je proces odstaven
    - předpokládá se konstatní priorita
  - Rate monotonic
    - statické přidělování priorit
  - Earliest deadline first
    - zpracování úloh probíhá na základě mezní doby platnosti procesu
  - atd.

RT operační systémy – RTOS

- RTOS jsou specializované operační systémy, které řídí procesy v RTS.
- Odpovídají za správu procesů a přidělování zdrojů (procesoru a paměti).
- Komponenty
  - Real-time clock – poskytuje informace pro plánování procesů
  - Interrupt handler – spravuje nepravidelné žádosti o obsluhu
  - Scheduler – vybere další proces, který má být spuštěn
  - Resource manager – alokuje paměť a zdroje (procesor, vlákno, ...)
  - Dispatcher – spustí proces
- Obsluha přerušení
  - Ovládání se automaticky přenese do předem určeného místa v paměti.
  - Toto místo obsahuje instrukci pro skok do rutiny obsluhy přerušení.
  - Další přerušení (od stejného zdroje) jsou zakázána.
  - Přerušení je obslouženo a řízení je vráceno přerušenému procesu.
  - Obslužné rutiny přerušení MUSÍ být krátké, jednoduché a rychlé.

Parametry pro výběr vhodného operačního systému

Režie OS Vestavná zařízení mají omezenou paměť – OS by měl mít minimální požadavky na paměť, spotřebu a výkon.

Přenositelnost OS je middleware nezávislý na hardware, obvykle je na různé platformy a rozhraní přenášen pomocí BSP (Board Support Package)

Modularita OS by měl mít pevné jádro a další funkce by měly být přidány ve formě doplňků. OS je pak možné přizpůsobit aplikaci a zredukovat potřebné zdroje.

Konektivita Operační systém by měl podporovat různé protokoly připojení, například Ethernet, Wi-Fi, BLE, IEEE 802.15.4 a další.

Škálovatelnost OS by měl být škálovatelný pro jakýkoli typ zařízení – vývojáři a integrátoři budou znát pouze jeden OS pro uzly i brány.

Spolehlivost Vestavná zařízení jsou často na odlehlých místech a musejí pracovat roky bez selhání. Dále by OS měl splňovat příslušné certifikáty pro určité aplikace.

Bezpečnost OS má doplňky, které přinášejí zabezpečení zařízení pomocí bezpečného spouštění, podpory SSL a ovladačů pro šifrování.

Spolehlivost – certifikace

DO-178B	pro systémy avioniky
IEC 61508	pro průmyslové řídicí systémy
ISO 62304	pro zdravotnické zařízení
SIL3 / SIL4	pro dopravu a jaderné systémy (Safety integrity level)

MISRA

- Motor Industry Software Reliability Association
- Sada doporučení pro bezpečné programy C a C++
- Cílem je zajistit bezpečnost a robustustnost implementace
- Gudelines nejsou volně dostupné

Operační systém RIOT

- Bezplatný operační systém s otevřeným zdrojovým kódem (LGPLv2.1)
- Programování je v jazyku C nebo C++
- Podporuje standardní nástroje jako jsou gcc, gdb nebo valgrid.
- Architektury: AVR, ARM7, Cortex-M0, Cortex-M0 +, Cortex-M3, Cortex-M4, Cortex-M7, ESP8266, MIPS32, MSP430, PIC32, x86.
- Desky: Airfy Beacon, Arduino Due, Arduino Mega 2560, Arduino Zero, Atmel samr21-Xplained Pro, f4vi, mbed NXP LPC1768, Micro::bit, Nordic nrf51822 (DevKit), Nordic nrf52840 (DevKit), Nucleo desky (téměř všechny) a mnoho dalších.
- Web: `https://www.riot-os.org/`

Mongoose OS

- framework pro vývoj firmwaru internetu věcí (Apache Licence 2.0 nebo Enterprise)
- cílem je komplexní řešení pro vývoj a správu
- nízkopříkonové MCU: ESP32, ESP8266, TI CC3200, STM32
- Cloudové integrace: AWS IoT, Google IoT, Microsoft Azure, IBM Watson a podpora MQTT / REST
- K dispozici je Dashboard mdash, kde získáte plnou správu svých připojených zařízeních. mdash poskytuje mnoho funkcí, jako je aktualizace firmwaru OTA, stav webového rozhraní online / offline a metadata (verze firmwaru, doba provozu atd.).
- K dispozici je také mobilní aplikace pro iOS i Android.
- Web: `https://mongoose-os.com/`

Contiki OS

- Operační systém s otevřeným zdrojovým kódem (BSD licence)
- Poskytuje multitasking, jádro 10kB RAM + 30kB ROM, jazyk C
- Simulátor Cooja umožňuje emulovat celou Contiki síť
- Web: `https://github.com/contiki-ng/contiki-ng/wiki`

RTOS – Projekt Zephyr a Nucleus

Projekt Zephyr

- Malý škálovatelný RTOS, původně pod patronací Linux Foundation (Apache licence)
- Jádro Zephyr je odvozeno od komerčního VxWorks Microkernel Profile.
- MCU: `https://docs.zephyrproject.org/latest/boards/index.html`
- Web: `https://zephyrproject.org/`

Nucleus

- OS od divize Embedded Software společnosti Mentor Graphics
- Honeywell: systém varování před přiblížením k zemi v aplikaci pro letecký průmysl.
- Garmin: Avionics Navigator CNX80
- ZOLL: automatizovaný externí defibrilátor AED Plus
- MCU: `https://www.mentor.com/embedded-software/nucleus/processor-support(linkisexternal)`

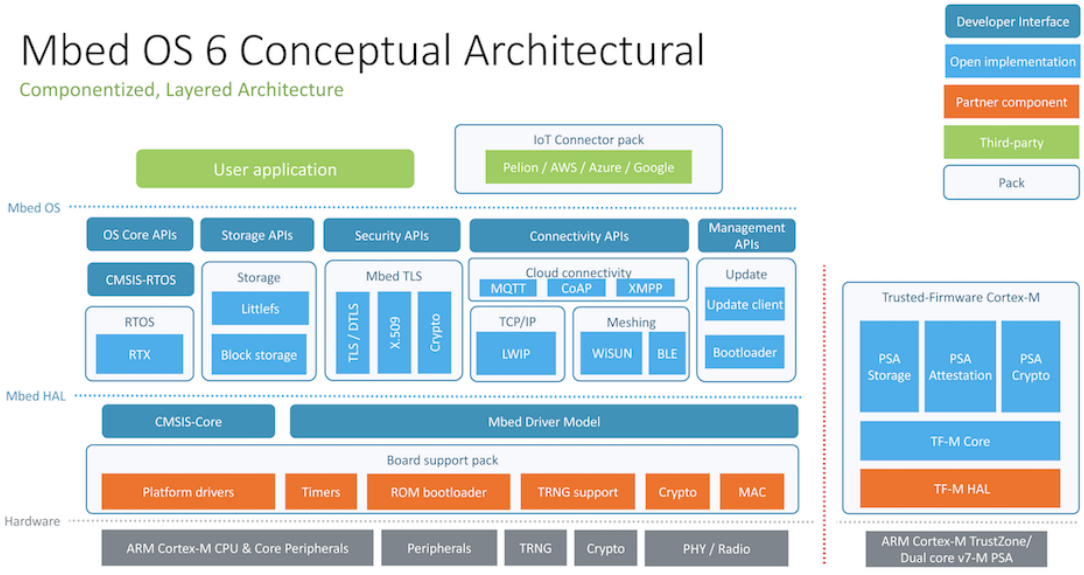
Mbed OS

- RTOS určený primárně pro procesory ARM (licence Apache 2.0)
- Dostupný zdrojový kód: `https://github.com/ARMmbed/mbed-os`
- Podporuje BLE, NFC, RFID, LoRa, 6LoWPAN-ND, Thread, Wi-SUN, Ethernet, Wi-Fi, LPWA
- Propojení s platformou Pelion IoT
- Nástroje
  - Mbed CLI – příkazový řádek, založeno na Pythonu
  - Mbed Online Compiler
  - Mbed Studio
- Desky: `https://os.mbed.com/platforms/`
- Komponenty: `https://os.mbed.com/components/`
- Web: `https://os.mbed.com/`

Mbed OS – Hello world!

```
1  #include "mbed.h"
3  #define WAIT_TIME_MS 500
5  DigitalOut led1(LED1);
7  int main()
8  {
9    while (true)
10   {
11     led1 = !led1;
12     thread_sleep_for(WAIT_TIME_MS);
13   }
14 }
15
```

Mbed OS – architektura



Mbed OS – API

- Scheduling: RTOS and event handling
- Drivers
  - UART, SPI, Input/Output, USB, ...
- Platform
  - Time, Power, Memory, ...
- Data Storage
  - File system, Block device
- Connectivity
  - Network socket, Network interface (cellular, ethernet, wi-fi), DNS, BLE, NFC, LoRaWAN
- Security

Mbed OS – RTOS

- Zahrnuje správu objektů, jako jsou vlákna, synchronizační primitiva a časovače.
- Poskytuje rozhraní pro připojení idle funkce specifické pro danou aplikaci a čtení stavu RTOS časovače (tickeru)
  - generuje periodicky (1ms) přerušení
  - používá se SysTick (součást Cortex-M jádra)
- Funkce `main()` je je speciální s funkce vlákna, které se vytvoří při inicializaci systému
- Vlákno může v být v některém z následujících stavů
  - Běžící: Aktuálně běžící vlákno. V tomto stavu může být vždy pouze jedno vlákno.
  - Připravené: Vlákna, která jsou připravena ke spuštění. Jakmile se běžící vlákno ukončí nebo čeká, stane se běžícím vláknem připravené vlákno s nejvyšší prioritou.
  - Čekající: Vlákna, která čekají na událost.
  - Neaktivní: Vlákna, která nejsou vytvořena nebo ukončena. Tato vlákna obvykle nespotřebovávají žádné systémové prostředky.
- Každé vlákno může čekat na signál nebo být notifikováno událostí

Mbed OS – RTOS – vlákna

- Základní pojmy: atomická operace, kritická sekce
- Sychnronizační mechanismy: mutex, signál, fronta
- Různé komponenty Mbed OS poskytují různé úrovně synchronizace vláken
  - Interrupt safe – bezpečné pro použití z více vláken a přerušení; operace se provádí atomicky nebo v kritické sekci. Chování je dobře definováno při použití z přerušení i z vláken.
  - Thread safe – bezpečný pro použití z více vláken; operace je chráněna primitivem RTOS a může být použita z více vláken, ale způsobí problémy, pokud je použita z rutiny obsluhy přerušení.
  - Nechráněno – operace není chráněna proti souběžnému přístupu a musí být synchronizována externě. Při volání z více vláken bez jiné formy synchronizace může dojít k poškození dat a chování je nedefinované.
- Většina komponent je thread safe
- Interrupt safe komponenty: DigitalIn, DigitalOut, DigitalInOut, InterruptIn, PortIn, PortOut, PortInOut, PwmOut, Ticker, TimerEvent, Timeout, Timer
- Nechráněné komponenty: SPISlave, I2CSlave

Mbed OS – RTOS – příklad 1/2

```
1  #include "mbed.h"
2
3  DigitalOut  led1(LED1);
4  InterruptIn sw(SW2);
5  EventQueue  queue(32 * EVENTS_EVENT_SIZE);
6
7  Thread t;
8
9  void rise_handler(void)
10 {
11   led1 = !led1;
12 }
13
14 void fall_handler(void)
15 {
16   led1 = !led1;
17 }
```

Mbed OS – RTOS – příklad 1/2

```
1  int main()
2  {
3    // Start the event queue
4    t.start(callback(&queue, &EventQueue::dispatch_forever));
5
6    // The 'rise' handler will execute in IRQ context
7    sw.rise(rise_handler);
8
9    // The 'fall' handler will execute in the context of thread 't'
10   sw.fall(queue.event(fall_handler));
11 }
12
```

Mbed OS – ovladače

- Ovladače poskytují přístup k periferiím procesoru, v podobě C++ třídy
- Pokrývají I/O (analogový i digitální) a komunikační rozhraní (UART, SPI, USB)
- Defaultně jsou ovladače blokující
  - vlákno, ve kterém je volána instance ovladače čeká na dokončení operace
  - pokud je vlákno blokováno, RTOS přepne do neblokokového vlákna nebo procesor uspí
  - instance lze volat paralelně v různých vláknech – thread safe
- Neblokující ovladač běží na pozadí a informuje o dokončení prostřednictvím propojené callback funkce
- Propojení s callback funkcemi je součást API
  - ovladače volají callback funkce při různých příležitostech
  - volání callback funkce vyvolá přerušení, které má vyšší prioritu než všechna vlákna
  - veškerý kód, který je v rutině přerušení musí být interrupt safe – to vylučuje blokující ovladače, mutexy a např. malloc
  - lepší je využít semafor nebo frontu událostí .

Mbed OS – ovladač UART

- `BufferedSerial` – Třída pro sériové přenosy dat, zařízení se může spojit s jiným zařízením (například senzory, PC nebo jinou deskou) a vyměňovat si data nebo posílat text, který se zobrazí na textovém rozhraní počítače.
- `UnbufferedSerial` – Podobná třída jako `BufferedSerial`, ale je určena pro aplikace, které mají málo paměti RAM a nemohou si dovolit vyrovnávací paměť nebo které potřebují větší kontrolu nad sériovým portem a využívají jej z IRQ.
- Používají se dva piny: TX a RX
- Parametry spojení (třídy):
  - Baud rate – předdefinovaná rychlost, typicky 9200, 19200 nebo 115200
  - Datový rámec – může mít 5 až 8 (nebo 9, pokud není použit paritní bit) bitů pro aktuální přenášená data.
  - Paritní bit – volitelný bit, který se používá pro detekci chyb v datech.
  - Stop bity – 1-2 bity pro signalizaci konce datového paketu.

Mbed OS – BufferedSerial

```
1  BufferedSerial serial_port;
2  // ---
3    // nastaveni pozadovanych vlastnosti (9600-8-N-1).
4    serial_port.set_baud(9600);
5    serial_port.set_format(
6      /* bits */ 8,
7      /* parity */ BufferedSerial::None,
8      /* stop bit */ 1
9    );
11   char buf[MAXIMUM_BUFFER_SIZE] = {0};
13   while (1) {
14     if (uint32_t num = serial_port.read(buf, sizeof(buf)))
15       serial_port.write(buf, num); // echo zpet do terminalu
16   }
```

Mbed OS – DigitalOut

```
DigitalOut myled(LED1);
// ---
  if (myled.is_connected()) {
    printf("myled is initialized and connected!\n\r");
  }

  while (1) {
    myled = 1;           // set LED1 pin to high
    printf("myled = %d \n\r", (uint8_t)myled);
    ThisThread::sleep_for(500);
    myled.write(0);      // set LED1 pin to low
    printf("myled = %d \n\r", myled.read());
    ThisThread::sleep_for(500);
  }
```

Mbed OS – DigitalIn

```
DigitalIn  mypin(BUTTON1);
DigitalOut myled(LED1);
// ---
  // Optional:  set mode as PullUp/PullDown/PullNone/OpenDrain
  mypin.mode(PullNone);

  // press the button and see the console / led change
  while (1) {
    printf("mypin: %d \n\r", mypin.read());
    myled = mypin; // toggle led based on value of button
    ThisThread::sleep_for(250);
  }
```

Mbed OS – InterruptIn – 1. příklad

```
1  InterruptIn button(BUTTON1);
2  DigitalOut  led(LED1);
3  DigitalOut  flash(LED2);
5  void flip()
6  {
7    led = !led;
8  }
10 int main()
11 {
12   button.rise(&flip);  // callback function to the rising edge
13   while (1) {          // wait around
14     flash = !flash;
15     ThisThread::sleep_for(250);
16   }
17 }
```

Mbed OS – InterruptIn – 2. příklad 1/2

```
1  class Counter {
2    InterruptIn  interrupt;
3    volatile int count;
4  public:
5    Counter(PinName pin) : interrupt(pin) {
6      interrupt.rise(callback(this, &Counter::increment));
7    }
9    void increment() {
10     count++;
11   }
13   int read() {
14     return count;
15   }
16 };
```

Mbed OS – InterruptIn – 2. příklad 2/2

```
1  Counter counter(BUTTON1);
3  int main()
4  {
5    while (1) {
6      printf("Count so far: %d\n", counter.read());
7      ThisThread::sleep_for(2000);
8    }
9  }
```

Mbed OS – další užitečné ovladače

- AnalogIn – čtení externího napětí přivedeného na analogový vstupní pin
- AnalogOut – nastavení výstupního napětí analogového výstupního pinu, zadané v procentech
- BusIn – sloučí několik pinů `DigitalIn` a čte je jako jediné rozhraní
- BusOut – sloučí několik pinů `DigitalOut`, zápis na jedno rozhraní
- BusInOut – obousměrná sběrnice, která podporuje až 16 pinů `DigitalInOut`
- DigitalInOut – obousměrný digitální pin
- PwmOut – řízení frekvence a pracovního cyklu signálu PWM
- ResetReason – zjistí důvod restartu systému
- Watchdog – nastaví hardwarový časovač, který resetuje systém, pokud není pravidelně obnovován

Mbed OS – Platform – Time API – `Ticker` 1/2

- Rozhraní pro volání funkce (přerušení) v zadaném intervalu opakování
- Lze vytvořit libovolný počet instancí
- Volat lze statickou funkci, členskou funkci třídy, funktor nebo instanci `Callback`
- Omezení pro ISR
  - Žádný blokující kód v ISR: vyhněte se jakémukoli volání wait, nekonečnému cyklu while nebo obecně blokujícím voláním.
  - Žádný printf, malloc nebo new v ISR: vyhněte se volání objemných knihovních funkcí. Zejména některé knihovní funkce (jako printf, malloc a new) nejsou re-entrantní a jejich chování by mohlo být při volání z ISR poškozeno.
  - Zatímco je událost připojena k Tickeru, je blokován hluboký spánek, aby bylo zachováno přesné časování. Pokud nepotřebujete mikrosekundovou přesnost, zvažte místo toho použití třídy LowPowerTicker, protože ta režim hlubokého spánku neblokuje.

```
1  Ticker flipper;
2  DigitalOut led1(LED1);
3  DigitalOut led2(LED2);
5  void flip()
6  {
7    led2 = !led2;
8  }
9  // ---
10   led2 = 1;
11   flipper.attach(&flip, 2.0); // ISR a interval (2 seconds)
13   while (1) {
14     led1 = !led1;
15     ThisThread::sleep_for(200);
16   }
```

Mbed OS – Platform – Time API – Timer

- Rozraní pro vytvoření, spuštění a zastavení časovače s lepším než ms rozlišením
- Lze nezávisle vytvářet, spouštět a zastavovat libovolný počet instancí
- Když je časovač spuštěn, je blokován hluboký spánek, aby bylo zachováno přesné časování. Pokud nepotřebujete mikrosekundovou přesnost, zvažte místo toho použití tříd `LowPowerTimer` nebo `Kernel::Clock`, protože ty režim hlubokého spánku neblokují.

```
1  using namespace std::chrono;
3  Timer t;
4  // ---
5    t.start();
6    printf("Hello World!\n");
7    t.stop();
8    printf("The time taken was %llu milliseconds\n", duration_cast<
       milliseconds>(t.elapsed_time()).count());
```

Mbed OS – Platform – Power management

- Low-power verze časovačů, neblokují sleep módy – `LowPowerTimer`, `LowPowerTicker` a `LowPowerTimeout`
- V Mbed OS je funkce `sleep()`, která automaticky vybere vhodný režim spánku

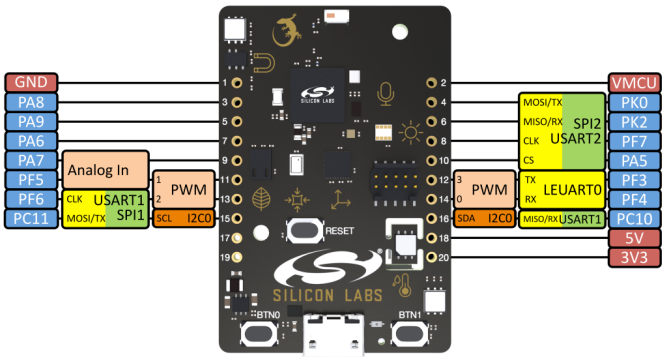
  Není třeba ji volat explicitně, Mbed OS volí režim spánku v případě nečinnosti.
- K dispozici jsou dva režimy spánku:
  1. Režim spánku – Systémové hodiny jádra se zastaví, dokud nedojde k resetu nebo přerušení. Tím se eliminuje dynamické napájení, které využívá procesor, paměťové systémy a sběrnice. Tento režim zachovává stav procesoru, periferií a paměti, periferie nadále pracují a mohou generovat přerušení. Procesor lze probudit libovolným interním přerušením periferie nebo přerušením externího pinu.
  2. Režim hlubokého spánku – Tento režim je podobný režimu spánku, ale šetří více energie a má delší dobu probuzení. Další energii šetří vypnutím vysokorychlostních hodin. Z tohoto důvodu můžete do tohoto režimu přejít pouze tehdy, když se nepoužívají periferní zařízení závislá na vysokorychlostních hodinách. Mezi periferie, které nespoléhají na vysokorychlostní hodiny, patří rozhraní API LowPowerTicker, RTC a InterruptIn. Tento režim zachovává všechny stavy.

Silicon Labs EFR32 1/2

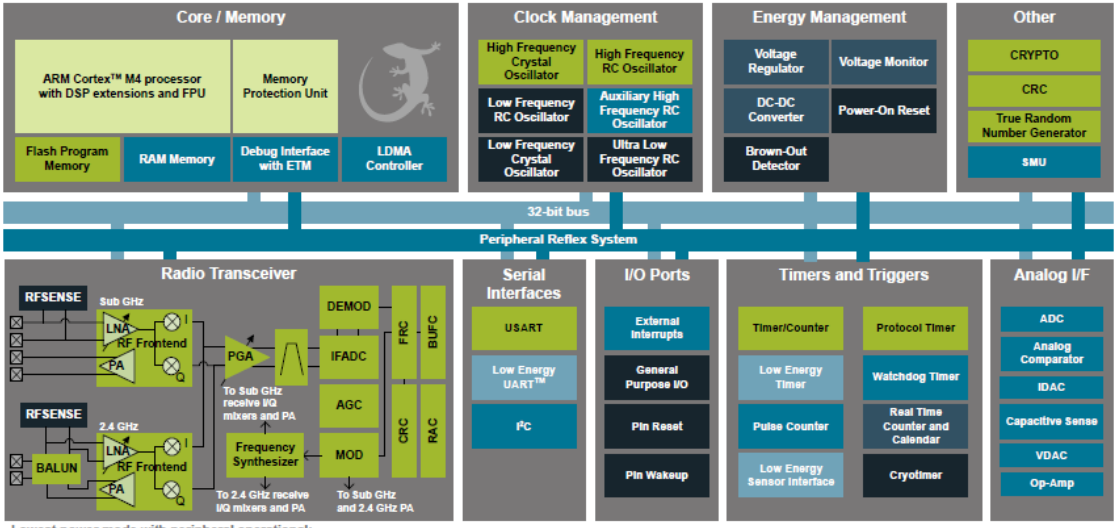
- ARMCortex-M4, 32bitový procesor @ až 40 MHz
- Flexibilní systém managementu spotřeby (5 módů)
- 1024 kB Flash, 256 kB RAM
- Až 65 GPIO pinů (konfigurovatelné push-pull, open-drain, pull-up/down, ...)
- 16 asynchronních externích přerušení, 8 DMA kanálů
- 12 kanálový Peripheral Reflex System – autonomní systém signalizace mezi periferiemi pro operace v low-energy módech
- Vysoká podpora šifrovacích metod
- Komunikační rozhraní: 4x USART, 1x low-energy USART, 2x I^2C
- Čítače/časovače: 2x 16bitový, 2x 32bitový (všechny 3-4 compare/capture/PWM kanály), 16bitový LE, 32bitový ultra LE pro periodické probouzení z různých energetických módů, 32bitový RT, 3kanálový 16bitový čítač pulsů, 2x watchdog s dedikovaný RC oscilátorem

Silicon Labs EFR32 2/2

- Analogové periferie: 12bitový ADC (1Ms/s), 12bitový DAC (500 kbps), až tři operační zesilovače, proudový DAC, 2x analogový komparátor
- LE senzor interface: autonomní monitorování periferií v deep sleep módu
- Debug rozhraní: JTAG, serial-wire (SWD)



Silicon Labs EFR32 – architektura



Silicon Labs EFR32 – oscilátory

název	frekvence	popis
HFXO	38–40 MHz	vysoká přesnost, minimální jitter, vyžadován pro všechny RF aplikace
HFRCO	1–38 MHz	středně přesný, typicky používán během startu HFXO a jako zdroj hodin, pokud není aktivní RF komunikace
AUXHFRCO	1–38 MHz	středně přesný, alternativní zdroj hodin pro ADC a debugovací rozhraní
LFRCO	32768 Hz	středně přesný zroj hodin pro RT aplikace
ULFRCO	1000 Hz	watchdog časovač

- RC oscilátory mohou být kalibrovány oproti krystalovému – kompenzace změn napájecího napětí nebo teploty
- Oscilátory lze konfigurova prostřednictvím Clock Management Unit

Silicon Labs EFR32 – čítače/časovače

název	instance	zdroj hodin	popis
RTCC	1	LFXO / LFRCO	32bitový čítač reálného času a kalendář, který se obvykle používá k přesnému neaktivní období v protokolu rádiové komunikace a umožňují probuzení při porovnání
TIMER	2	HFXO / HFRCO	16bitový čítač
WTIMER	2	HFXO / HFRCO	32bitový čítač
SysTick	1	podle konfigurace systému	32bitový systémový čítač, integrovaný v Cortex-M4 jádře
WDOG	2	LFXO, (U)LFRCO	Pokud je watchdog aktivní, musí být pravidelně dotazován, jinak způsobí reset
LETIMER	1	LFXO, (U)LFRCO	Low energy čítač

Silicon Labs EFR32 – jádro procesoru