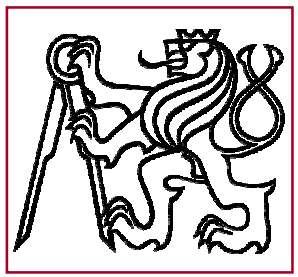
Y35PES

Programming for Embedded Systems

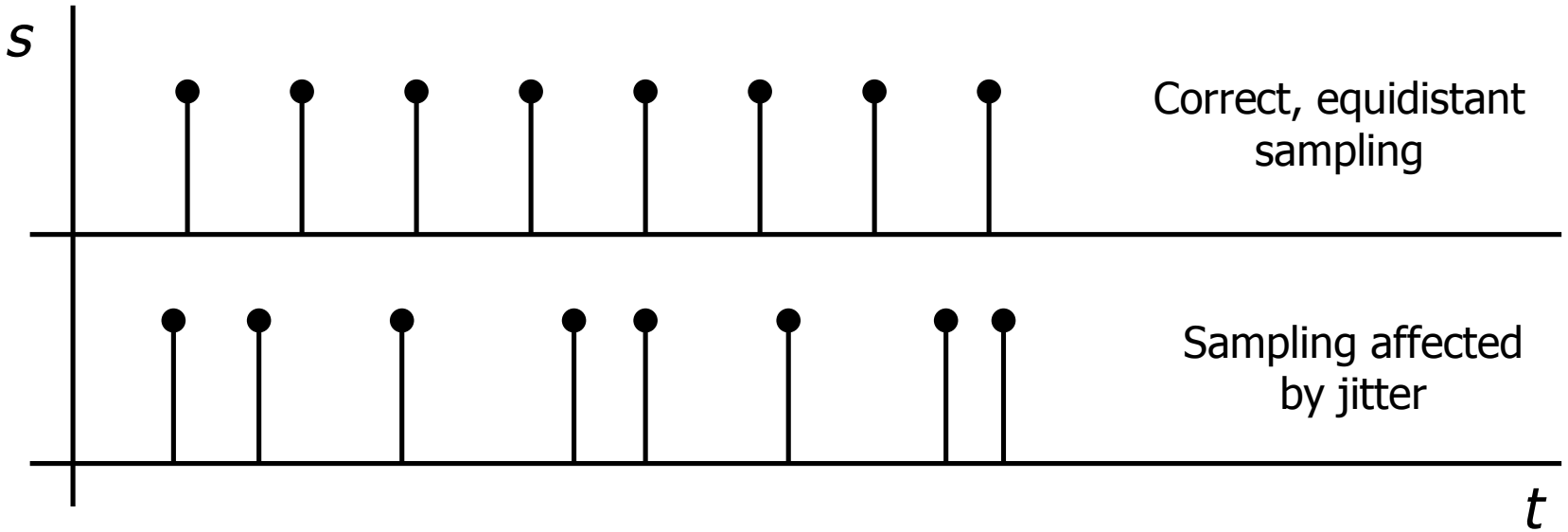
Ondřej Špinka, Pavel Němeček
spinkao@fel.cvut.cz nemecp1@fel.cvut.cz
http://dce.felk.cvut.cz/pes



Real-Time Operations (1)

By the term **real-time response** of a system we mean that the response of the system to an event is **guaranteed** to occur within a certain time.

This is often a natural demand for embedded systems, especially for control and measurement systems. A control/measurement system must ensure to maintain a steady **sampling period**, without variations of any kind (called **jitter**), as any discrepancy in the consecutive sampling times would affect the control algorithm performance.



Real-Time Operations (2)

In other words, correctly designed real-time system should guarantee a response/action time to lie in a relatively tight region. The response/action should be not early, nor late.

Often, the real-time constraints put to certain operations vary. Some of them are very strict (**hard**), requiring a very precise timing, on the other hand some of them might be **soft**, requesting only vague timing demands (for example that the response of the system should not be later than a specified time, without any request regarding the earliness).

Usually, the timing demands on sampling period / control action delivery are hard. On the other hand, the timing demands on communication with other systems might be soft (but might be also hard in certain cases).

Real-Time Operations (3)

What can influence the real-time behavior of a computer system:

In this part, we shall restrict ourselves to **system-less** computer systems only. When any kind of Operating System comes into play the things get a lot more complicated. The analysis of a operating system real-time properties exceeds the scope of this course.

The first natural demand is that the system must have enough computational power to be capable to perform all required operations within the period of a hard real-time task.

Then the interrupts must be taken into account. The interrupts inherently prolong the "normal" program run, but also other interrupts (with lower priority). A thorough analysis of the worst-case event sequence, generating the longest delay, must be performed in order to evaluate the worst-case response of the system. On the other hand, a certain mechanism must also be implemented to prevent excessive earliness (if required), to ensure correct measurement/action delivery.

Real-Time Operations (4)

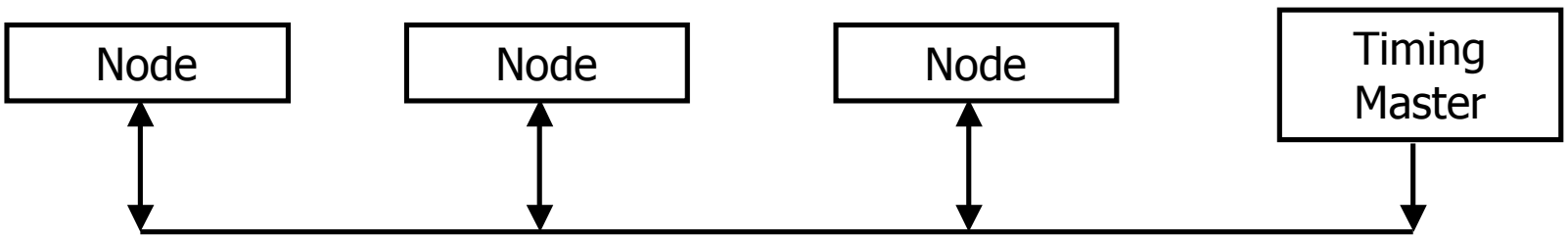
Sufficient programming techniques:

Because real-time operations are mostly interrupt-driven, we shall talk mainly about interrupts.

- The interrupt handlers should be kept as simple and fast as possible to prevent excessive interrupt latency.
- Interrupt occurrence must be carefully evaluated, worst-case scenarios estimated and consequently correct priorities must be assigned to respective interrupts.
- Most time-critical and jitter-sensitive operations shall be implemented directly using the TPU hardware if possible – i.e. PWM generation, pulse counting / duty cycle / frequency measurement etc.
- If possible, measurement and action computation / delivery should be synchronized by using mutual time source in control systems (timer driven, either external or internal)
- If it is necessary to use multiple timers, those timers should be synchronized
- DMA channels should be used for data transfers whenever possible

Timing Synchronization

- The easiest way to achieve synchronous timing is to use a single timer and employ frequency multipliers for differently timed actions
- If this is not possible and separate timers must be used, a PID controller might be employed to synchronize them
- If the system is distributed, it is essential to use a common periodic time synchronization message. Latencies induced by excessive bus length, routers and/or repeaters, etc. must be taken into account



Data Sampling

- Could be synchronous (time-driven) or asynchronous (event-driven). In most cases, synchronous sampling is used.
- Sampling points should be equidistant in time (jitter-less), since jitter can trigger mayhem in the dependent control loops for many reasons (numerical integration and derivation, control algorithm designed for certain fixed delays etc.)
- Sampling frequency must be adequate; According the Shannon theorem, it must be at least twice as much as the highest frequency present in the sampled signal (neglecting this theorem results in aliasing).
- Sampling could be driven by timer interrupt, or handled completely by a certain peripheral, avoiding any CPU interference.

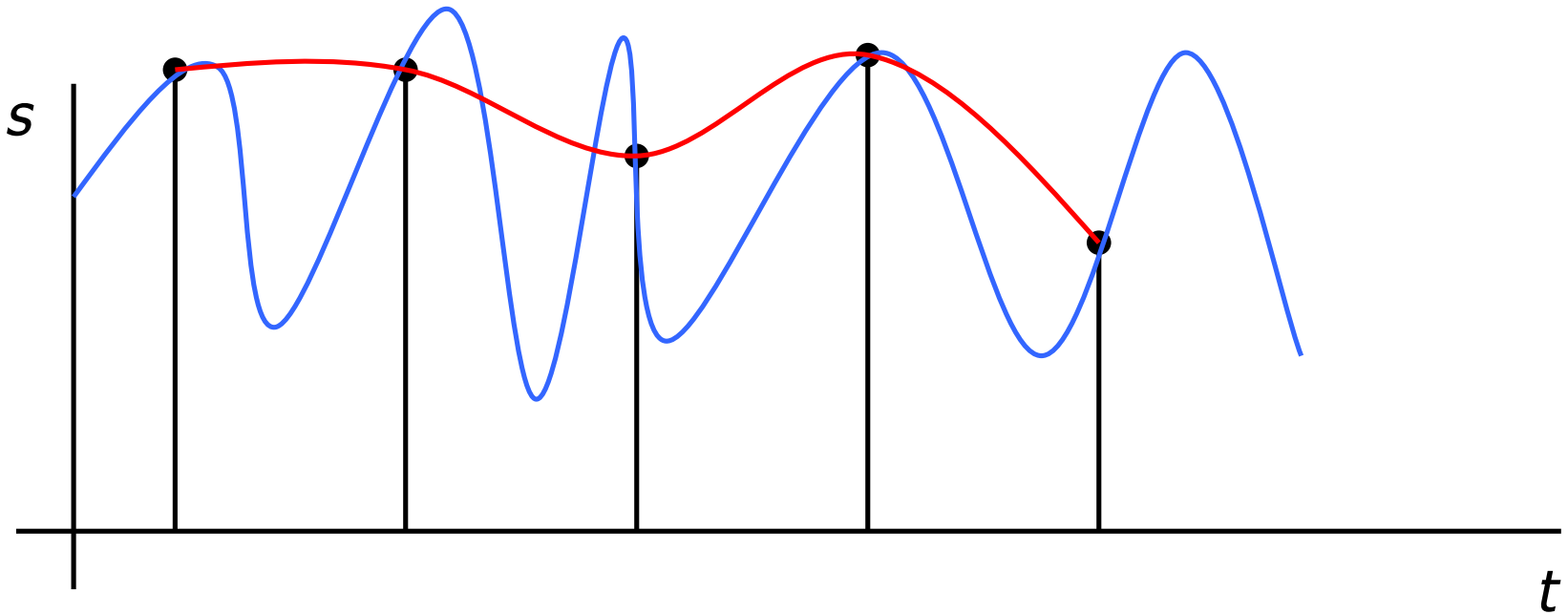
Example – Motor Control

Frequency Spectrum Aliasing (1)

Improper sampling may result in so-called **frequency spectrum aliasing.** This phenomenon occurs when the Shannon theorem boundary is not met, i.e. when

$$f_s < 2 * f_{max}$$

It is always wise to sample as fast as the computing power allows, even much faster than required by Shannon theorem, since it could simplify the controller design and make the system more robust.



Frequency Spectrum Aliasing (2)

Behold what aliasing can do in a real-life application…

Frequency Spectrum Aliasing (3)

Remember, frequency aliasing may lead to grave consequences…

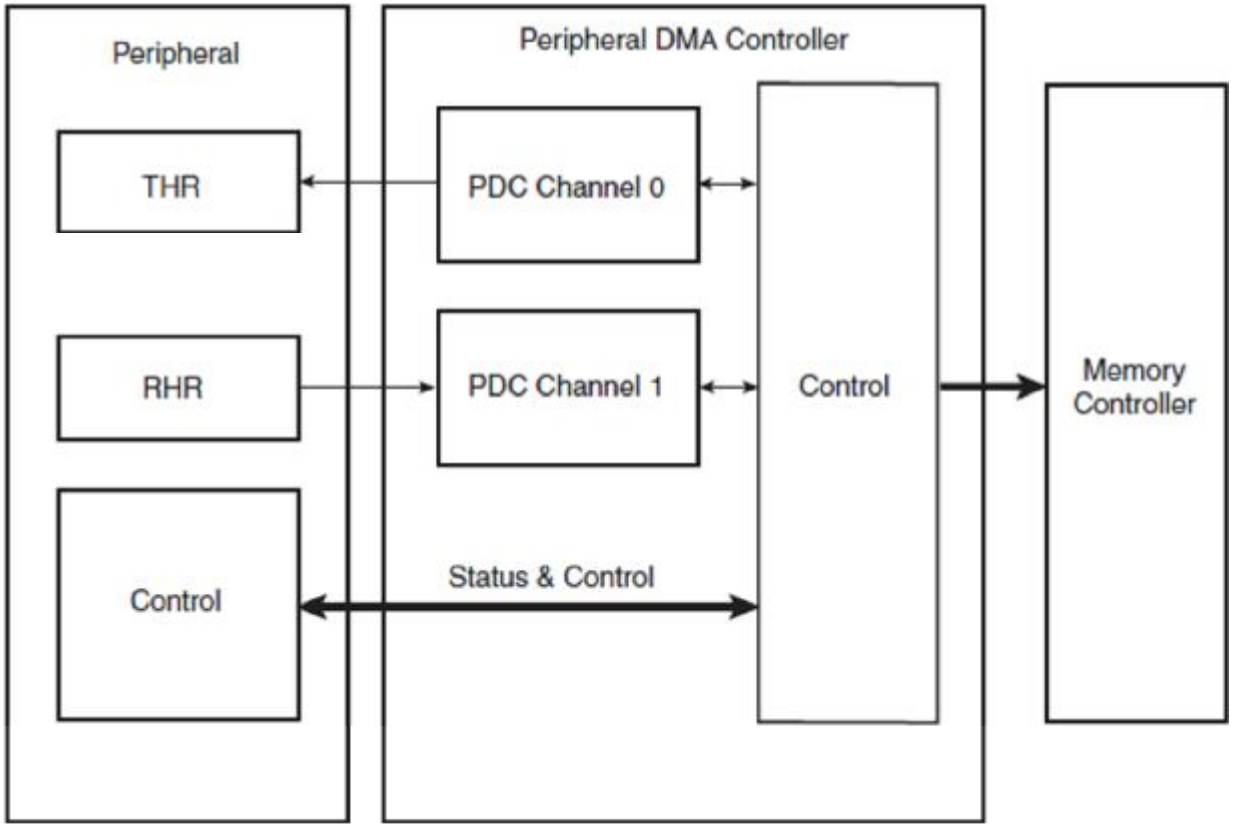
DMA channels on AT91SAM7 (1)

Using DMAs avoids the CPU messing with the data transfers and removes the correlated interrupt handling overhead, thus significantly reducing processor load and improving determinism of the operation. In AT91SAM7 MCU family, several peripherals support DMA (for example UART, USART, SPI, ADC etc.) The Peripheral DMA Controller (PDC) is very simple and easy to use. It has eight registers (four for transmit and four for receive), mapped to the memory area of each peripheral:

- Two 32-bit memory pointer registers (transmit and receive)
- Two 32-bit next transfer memory pointer registers
- Two 16-bit data counter registers
- Two 16-bit next data counter registers

DMA channels on AT91SAM7 (2)

PDC Block Diagram (picture property of Atmel Corporation)



Example - DMA transfers for SPI (1)

```
void xmit_datablock (  unsigned char *buff, unsigned int btr ) {
    unsigned int rdr;

    while ( AT91C_BASE_SPI0->SPI_TCR > 0 );

    rdr = AT91C_BASE_SPI0->SPI_RDR;

    AT91C_BASE_SPI0->SPI_PTCR = 0x200;
    AT91C_BASE_SPI0->SPI_TNPR = ( unsigned int ) ( buff );
    AT91C_BASE_SPI0->SPI_TNCR = btr;
    AT91C_BASE_SPI0->SPI_PTCR = 0x100;
}
```

Example - DMA transfers for SPI (2)

```
void rcvr_datablock (  unsigned char *buff, unsigned int btr ) {
    unsigned int rdr;

    while ( AT91C_BASE_SPI0->SPI_TCR > 0 );

    rdr = AT91C_BASE_SPI0->SPI_RDR;
    memset ( buff, 0xFF, btr );

    AT91C_BASE_SPI0->SPI_PTCR = 0x202;
    AT91C_BASE_SPI0->SPI_RNPR = ( unsigned int ) ( buff );
    AT91C_BASE_SPI0->SPI_RNCR = btr;
    AT91C_BASE_SPI0->SPI_TNPR = ( unsigned int ) ( buff );
    AT91C_BASE_SPI0->SPI_TNCR = btr;
    AT91C_BASE_SPI0->SPI_PTCR = 0x101;
}
```

Lecture Summary – Essential Things to Remember

- By the term **real-time response** of a system we mean that the response of the system to an event is **guaranteed** to occur within a certain time. Real-time constraints might be **soft** or **hard**.
- Unwanted variations in the **sampling period** are called **jitter**.
- It is highly desirable to use hardware peripherals of the MCU whenever possible, in order to assure precise timing and reduce CPU load.
- When multiple timing sources are used, it is highly desirable to have them synchronized.
- Frequency aliasing occurs when Shannon's condition $f_s > 2*f_{max}$ is not met.