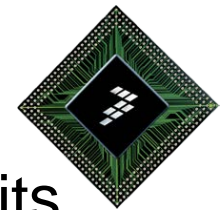




DwF 2011



Energy Efficient Operation of Automotive Control Units

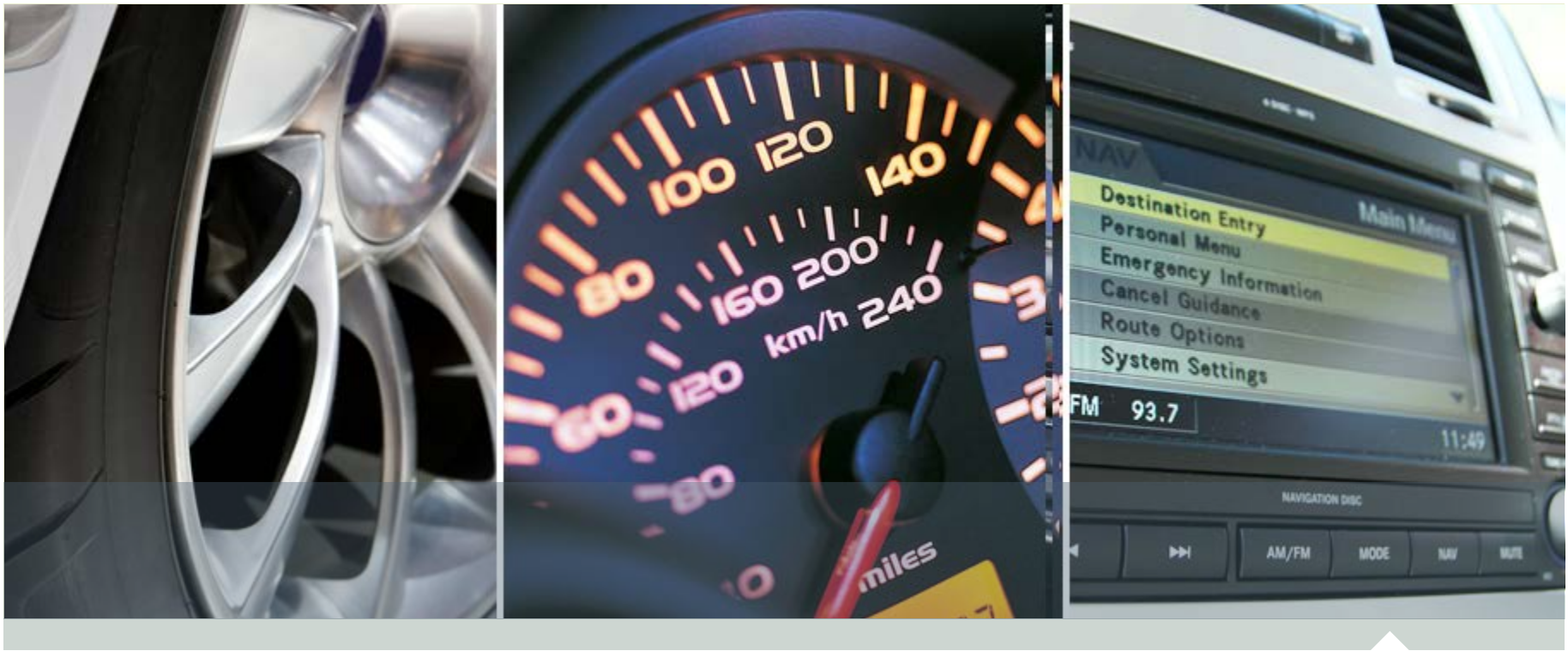
Stefan Imrich

Field Application Engineer

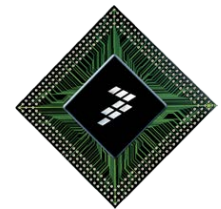


▶ The lecture will:

- Architecture evolution over the years
- Low power challenges starting on a typical body controller architecture
- MCU Low Power Mechanisms to address Low Power challenges
- Describe techniques to optimize their usage and concepts to address the challenges identified
- Software configuration example



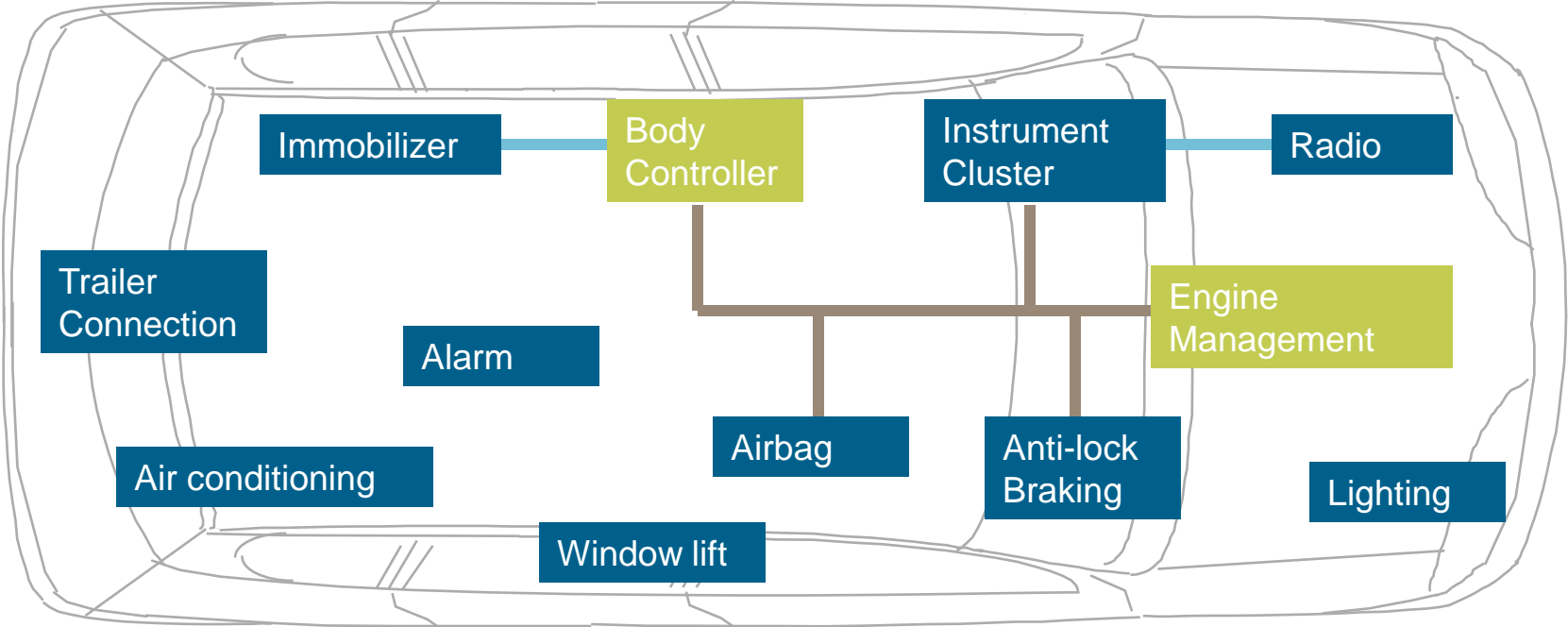
Architecture Evolution over the Years



1990's : The Birth of Networking into Cars

- CAN, VAN or J1850 depending on car makers
- Proprietary communication, K-line

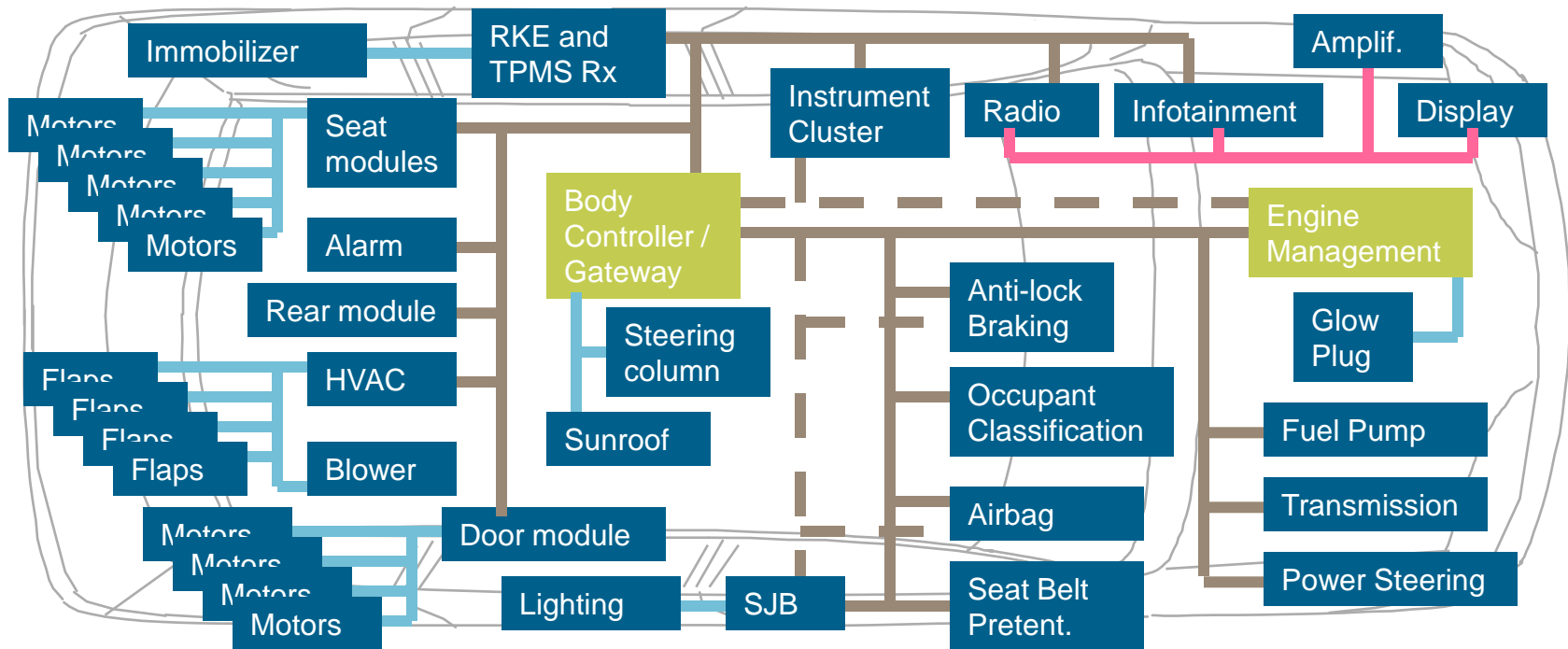
The Challenge: Get to some consistency, Reuse



2000's : CAN and LIN Standards Dominate






- CAN
- Diagnostic CAN
- LIN
- MOST

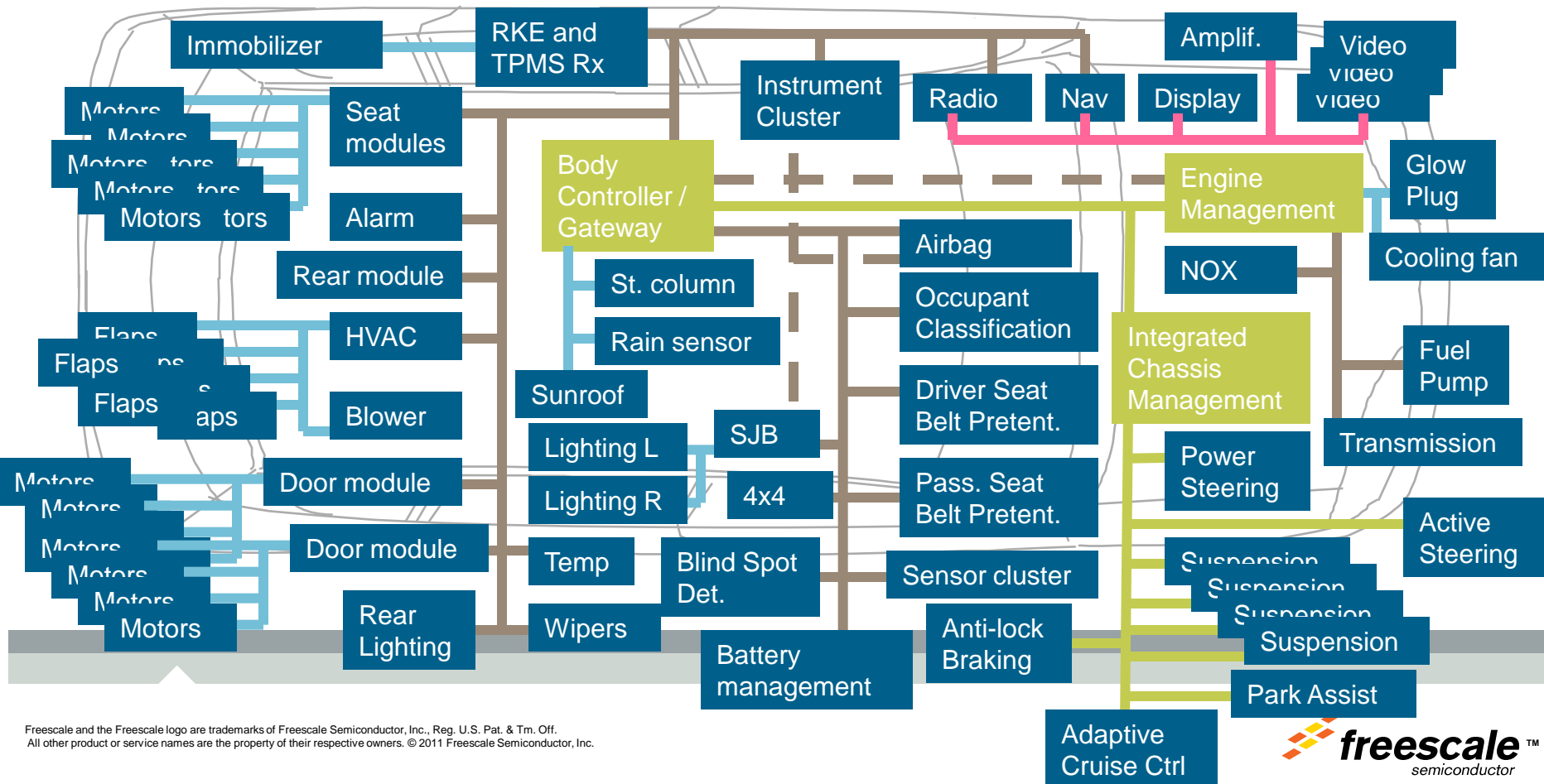
The Challenge: Manage the complexity, limited bandwidth and non-determinism



2010's : Hierarchical Networks

The Challenge : Overcome the complexity of “super nodes” through new design methodologies, standard software...

-  CAN
-  Diagnostic CAN
-  LIN
-  MOST
-  FlexRay



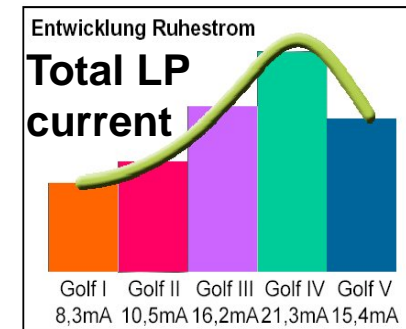
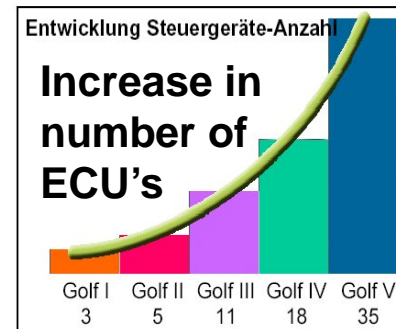
Low Power Challenge – Automotive

► Why has the number of ECUs increased?

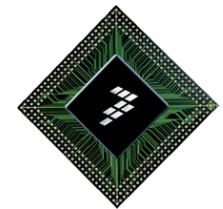
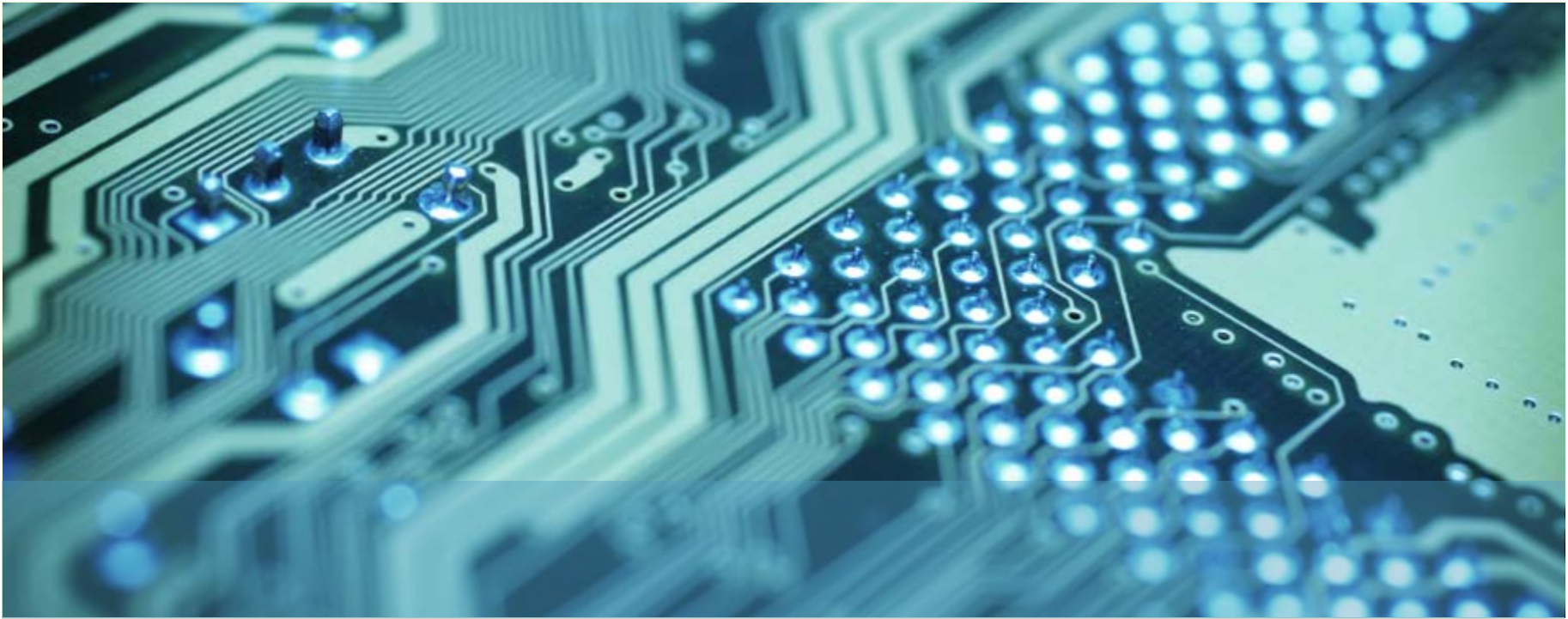
- Nodes requiring MCU functionality are replacing passive and mechanical systems throughout the vehicle.
- Nodes include measurement points, actuation points, or control.
- Often the vehicle architecture distributes some of the processing to local distributed nodes.

► As a consequence:

- Global Power budget requirement is either flat or decreasing then the power allowed per Electronic Control Unit (ECU) decreases.

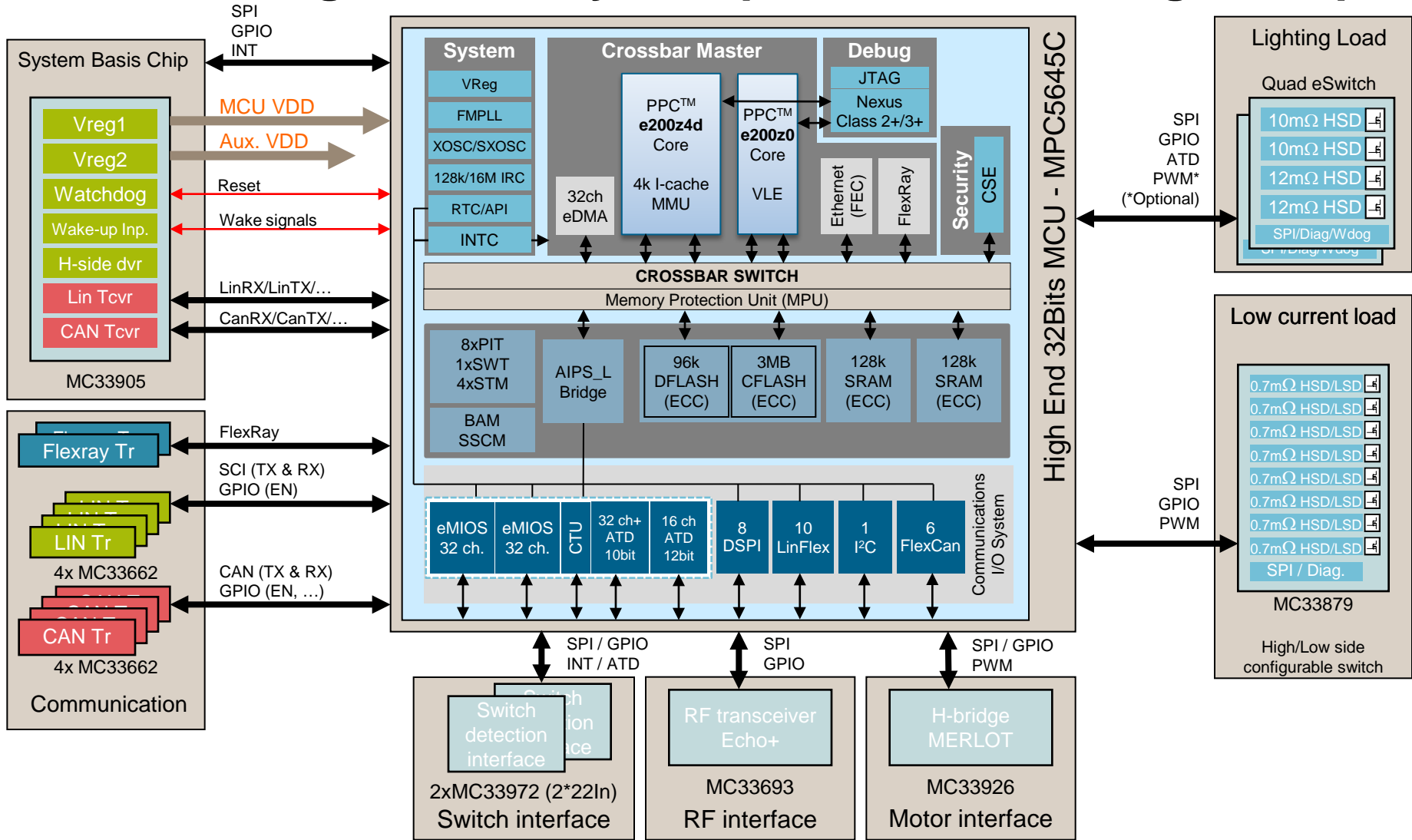


Source: J.Leohold, VW, 9th International Automobile Electronics Conference, June 05
trotz gesteigerter Komplexität ist beim Ruhestrom die Trendwende geschafft



Low Power Challenges

High End Body Computer – Partitioning Example



► Power

- Dynamic Power – MCU Run Current
- Static Power – MCU Stop Current

$$P_{\text{TOTAL}} = P_{\text{DYNAMIC}} + P_{\text{STATIC}}$$

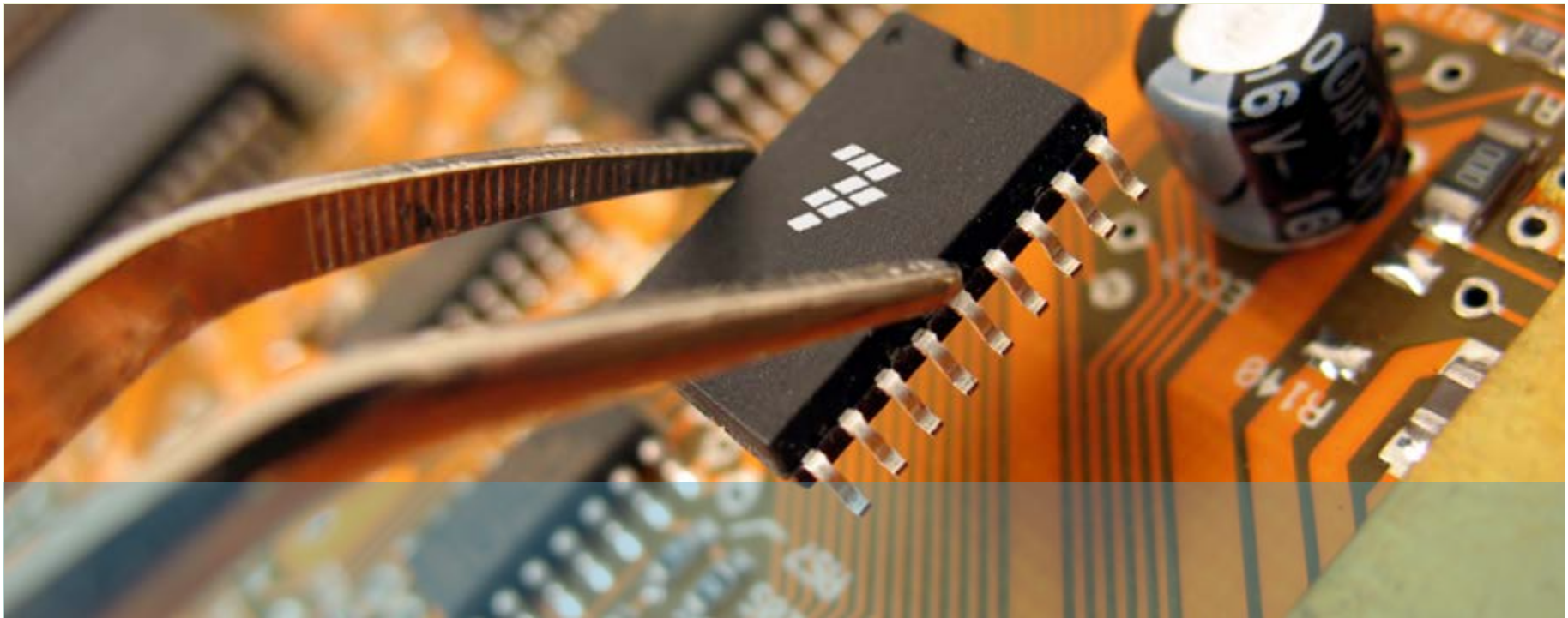
Where

$$P_{\text{DYNAMIC}} = \alpha \cdot CV^2f$$

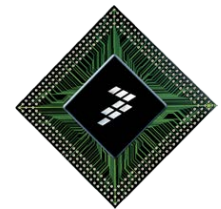
Computational Speed

$$P_{\text{STATIC}} = I_{\text{STATIC}} V$$

Supply Voltage



MCU Low Power Mechanisms to address Low Power Challenges



Power Segmentation – MCU Structure

▶ Power Domains

- Individually disconnected from Power
- Eliminate leakage from areas that are turned off

▶ Power Domain PD0:

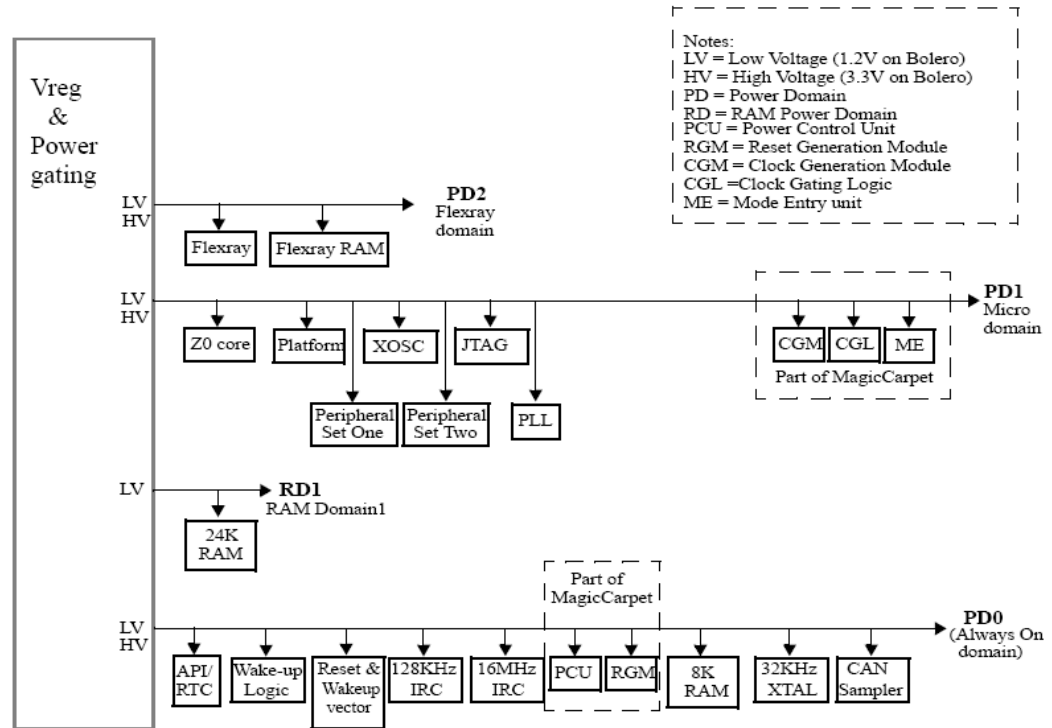
- Always on
- Wakeup peripheral, e.g., CAN sampler, RTC, API, etc.
- Minimum RAM segment

▶ Power Domain RD1

- Contains an additional RAM segment
- Remainder of the RAM is in the PD1 domain

▶ Power Domain PD1:

- Contains all cores and the majority of peripherals
- Can be turned off in STOP or STANDBY modes
- Must be turned on in RUN modes



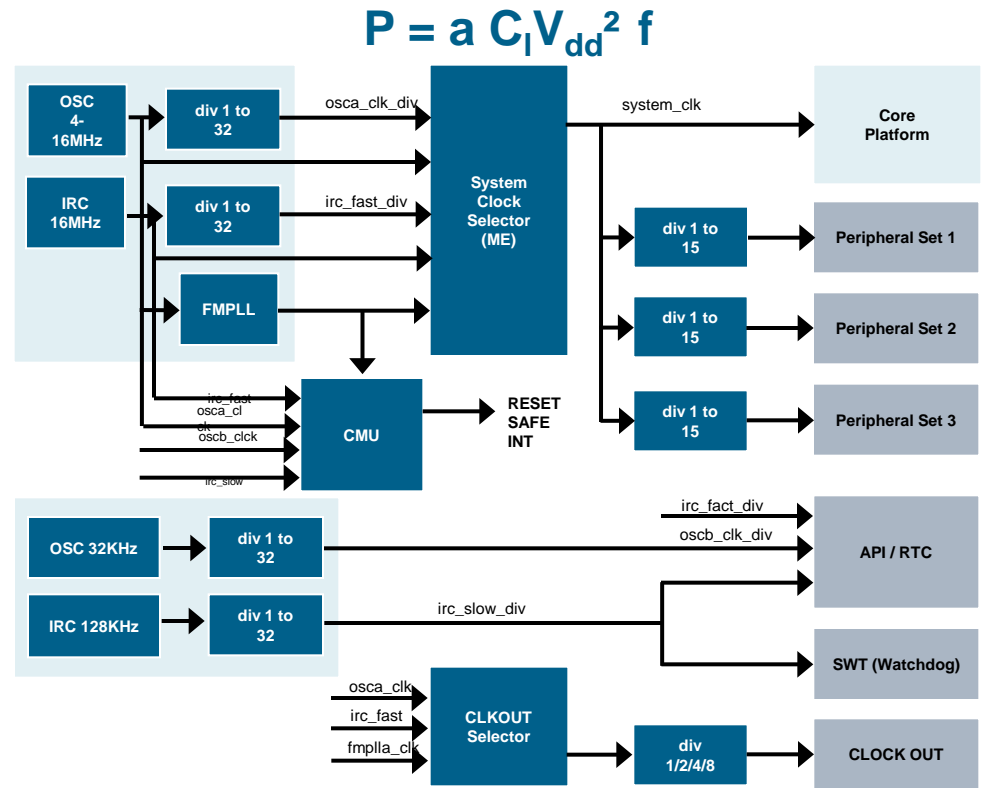
MPC5604B Power Domain Structure

Managing Clock Distribution

- ▶ Power consumption is directly linked to clock signal switching
- ▶ Reducing the number of clocked lines directly reduces current consumption
- ▶ Several methods are employed to avoid wasting power in clock edges,

for example:

- Clock freeze mode
 - When CPU activity can be temporarily halted, e.g., while waiting for an Analog-to-Digital (ADC) conversion to complete
- Peripheral bus division
 - Reduced local clock rates, e.g., for ADC and communication ports
- Clock gating
 - Applied wherever possible and at the entry to each sub-module



MPC5603B Intelligent Clock Tree architecture

Intelligent Clocking – Typical Clock Sources

► Fast Wake-Up:

- Increase the speed at which the device can recover from low power modes and start execution.
- Reconfiguration and non-timing critical operations can start as soon as a clock is present.
 - main RAM/module registers can be re-initialized whilst the external (accurate) OSC clock is stabilizing
- Very fast Wake-Up requires an on-chip RC oscillator
 - e.g., a 16 MHz Internal RC can provide Bus Clock in <5 cycles
- Allows near instant operation

► Periodic Wake-Up:

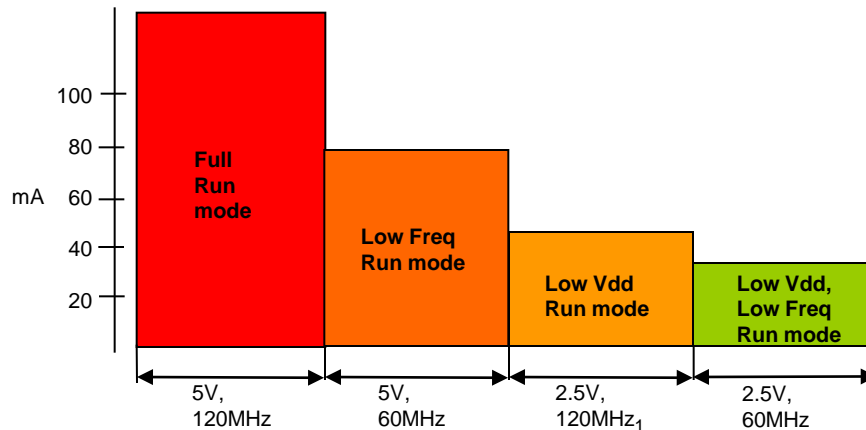
- Allows a reliable recovery in low power mode
- Minimizes average current

Clock Source	Start-up time	Power consumption	Accuracy
16 MHz Internal RC Osc.	L (<1us)	M (<50uA)	L (>10%)
128 KHZ Internal RC Osc.	L (<1us)	L (<1uA)	L (>10%)
32 KHz Ext. Crystal Osc.	H (ms)	L (<10uA)	H (<1%)
16 MHz Ext. Crystal Osc.	H (ms)	H (>100uA)	H (<1%)
Internal PLL	H (ms)	H (>1mA)	H (<1%)

←
Optimum
wake-up
clock

Dynamic Voltage Frequency Scaling (DVFS)

- ▶ Devices typically do not require full performance all of the time
 - Reducing frequency directly reduces power consumption (already discussed)
 - But reducing frequency also allows the voltage to be reduced, further reducing power
- ▶ Solutions can be via hardware, software or combination of both
 - System sophistication increases with DVFS
 - Typically PLL allows frequency scaling
 - Need to balance savings with complexity:
 - Scalable regulator design, with fast switching behaviour
 - Peripherals require full scaling capability to ensure seamless switching
 - Synchronization handling during DVFS changes



Note 1: Assumption is that maximum frequency also achievable at lowest voltage, but this will not always be the case

Low Power Modes

▶ STANDBY – lowest power mode

- Power supply is cut off from most devices
- All clocks disabled
- Longest wakeup time and some reconfiguration required
- Most pins not powered (high impedance)

▶ STOP

- Advanced low-power mode during which the clock to the core and the PLL are disabled
- Optionally switch off most peripherals
- State of the output pins is kept

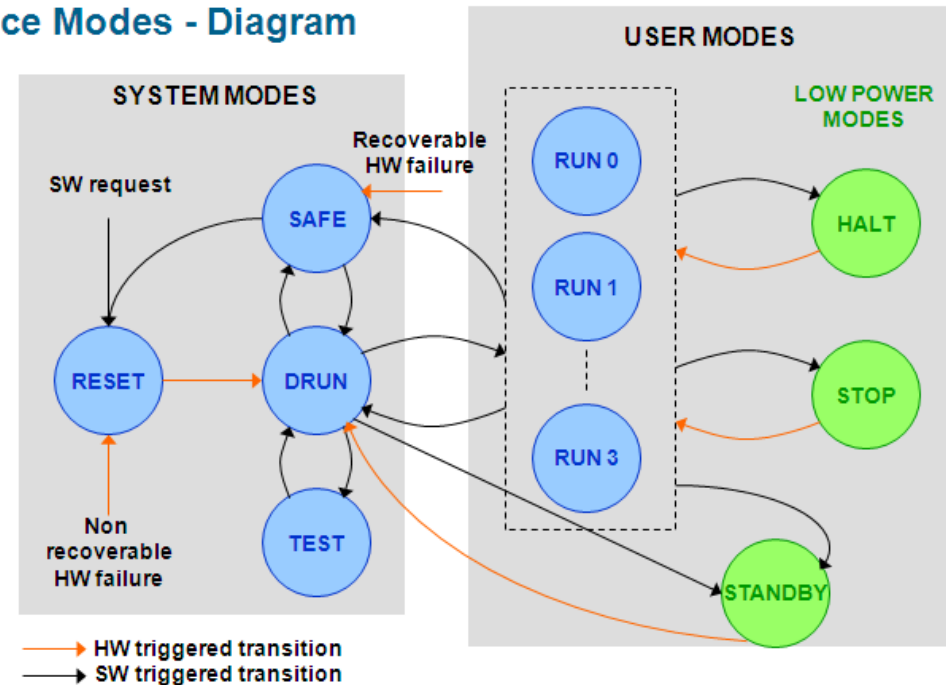
▶ HALT – disable core clock

- Reduced-activity mode during which the clock to the core is disabled
- Optionally switch off analog peripherals (PLL, flash, main regulator, etc.)

▶ RUN0-3

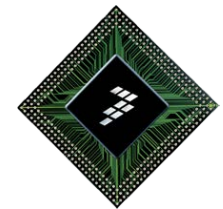
- Software execution modes where most processing activity occurs.
- Allows run-time customization of different clock & power configurations of the system

Device Modes - Diagram



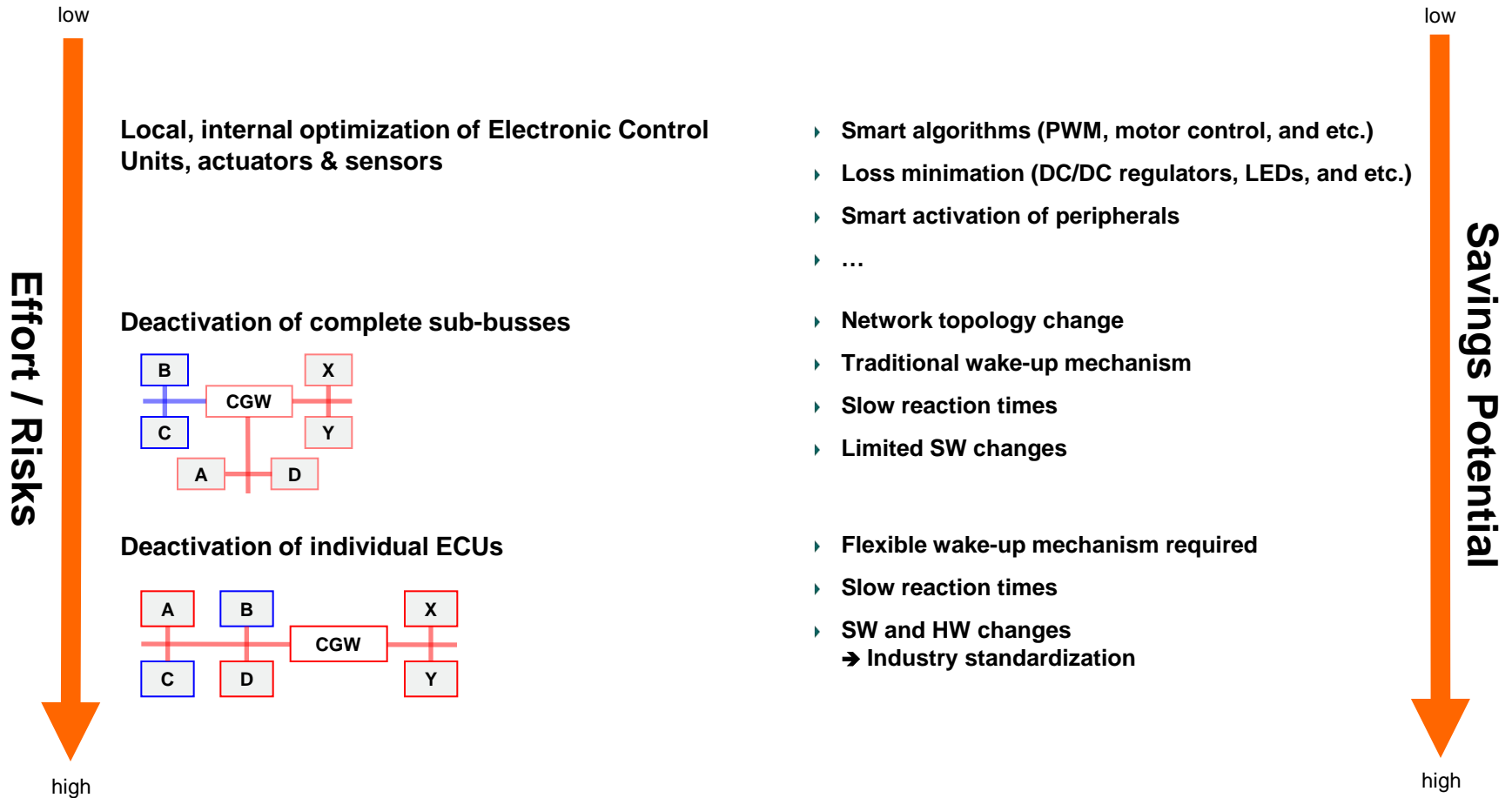
Autonomous Operation

- ▶ General theme is to switch on as little as possible:
 - CPU is the most power hungry module on an MCU
 - Need to switch to OFF when possible
 - Put more intelligence into peripherals
- ▶ Autonomous peripherals can help achieve this.
 - Typical Autonomous peripherals include:
 - API – Autonomous Periodic Interrupt
 - Allows device to recover from very low power state at selectable time intervals
 - RTC – Real Time Clock
 - Offers time keeping functionality in very low power states
 - DMA – Direct Memory Access
 - Allows data transfer between peripherals minimizing CPU activity
 - ADC – Analog Digital Converter
 - Continual conversion while running in low power
 - Triggers wake-up when signal reaches certain level
 - LINFlex – Intelligent LIN management, minimizing CPU interrupts

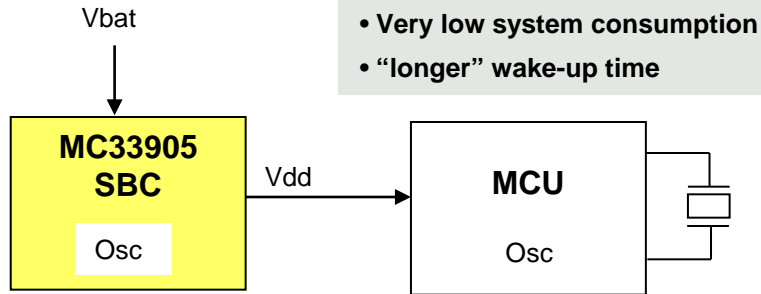


Low Power System Techniques

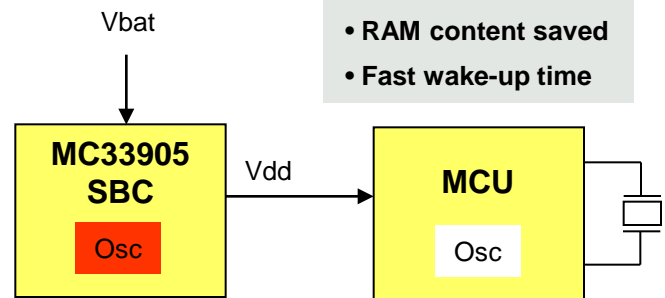
Energy Saving Potential on the System Level



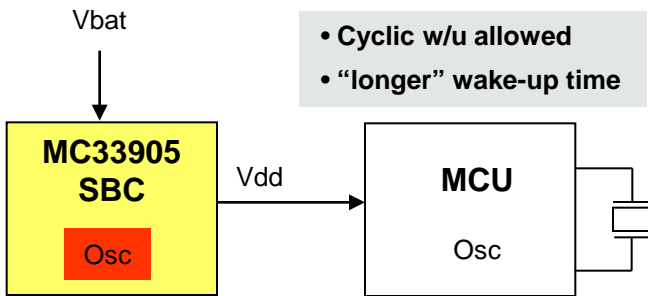
Electronic Control Unit Low Power Modes



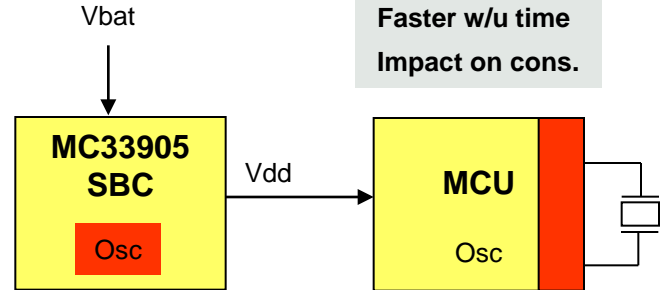
- **SBC LPM** VDD OFF, no osc no cyclic wake-up
- **MCU off**
 $I_{SBC} = 15 \mu A$
 Wake up time: 2-5ms + MCU S/W start



- **SBC LPM** VDDON with osc
- **MCU powered, STDBY mode no osc.**
 $I_{SBC} = 40 \mu A, I_{MCU} = 30 \mu A$



- **SBC LPM** VDDOFF + osc (cyclic wake-up)
- **MCU off**
 $I_{SBC} = 25 \mu A$



- **SBC LPM** VDDON with osc
- **MCU powered, HALT mode with osc.**
 $I_{SBC} = 40 \mu A, I_{MCU} < 350 \mu A$

Low Power System Techniques – Using the Silicon

▶ System solutions to achieve low power are based on several key principles:

Reduce average power

- Sleep as much as possible
- Minimize RUN execution
- Match speed against requirements



Only power what is needed

- Only switch on silicon portions
- Completely power gate unused portions in many power modes

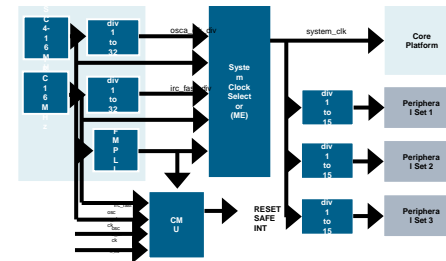
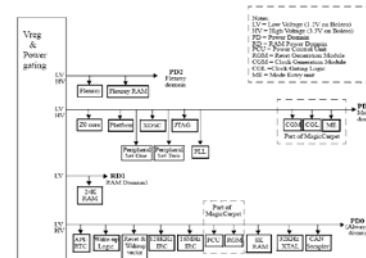
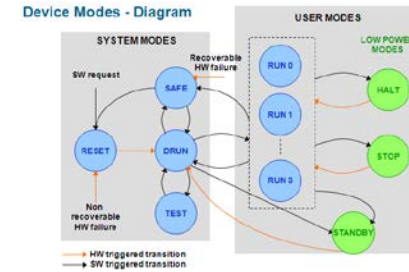


Only clock what is required

- Clock gating
- Clock tree management
- Peripheral grouping



Employ intelligent autonomous operation

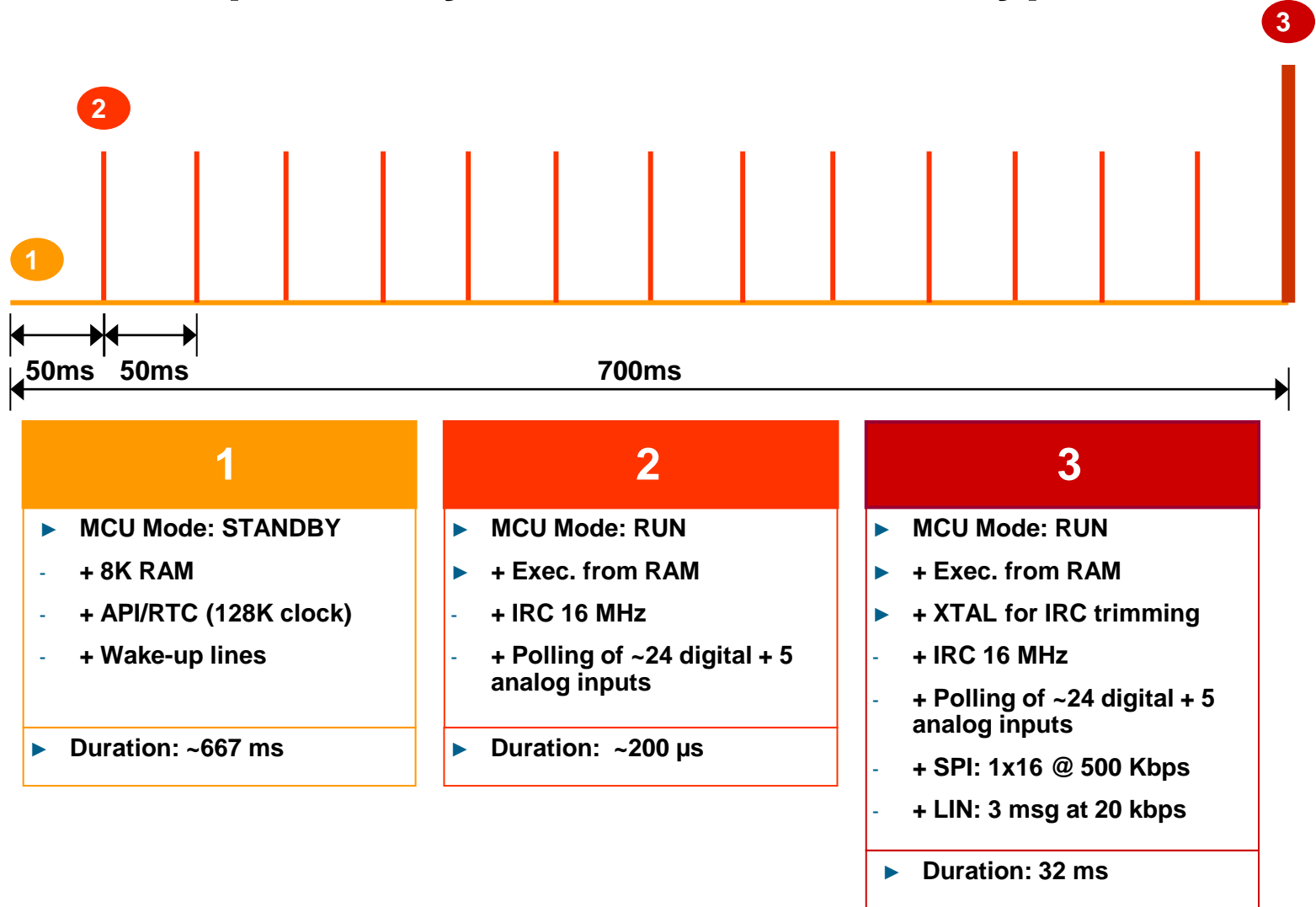


MPC5603B Intelligent Clock Tree architecture

Design autonomous peripherals

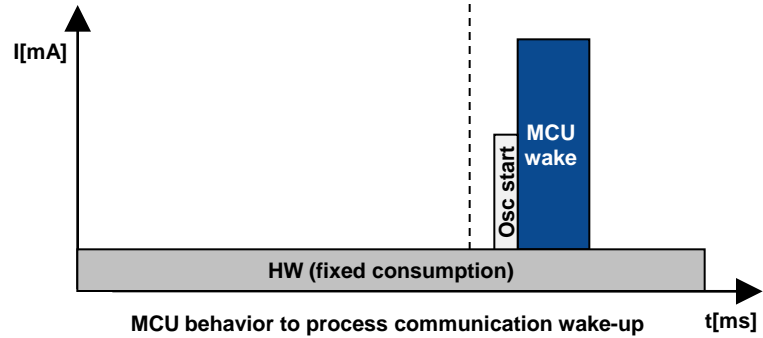
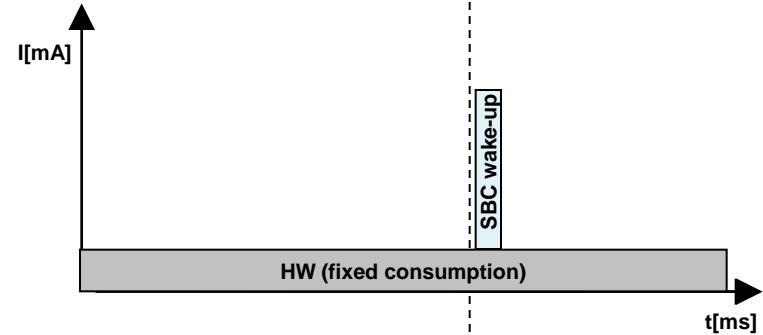
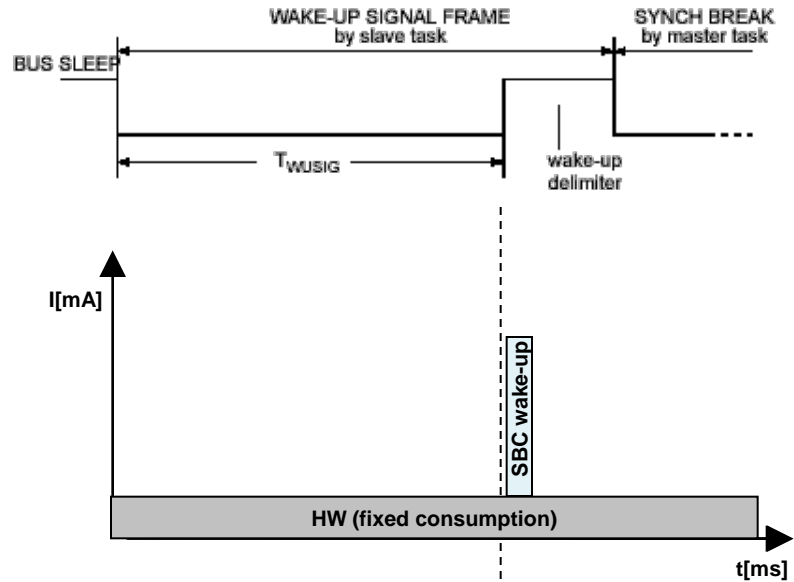
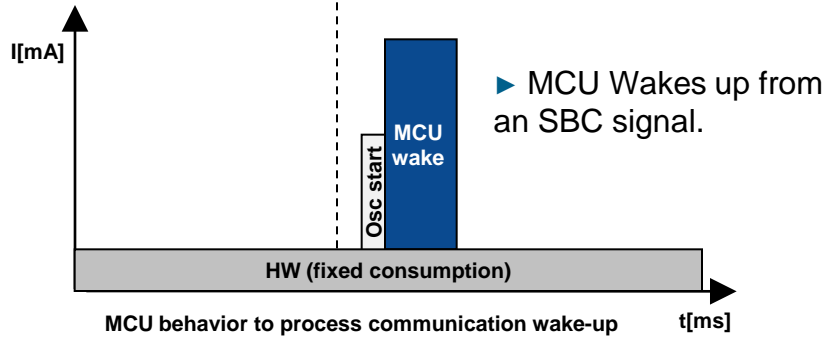
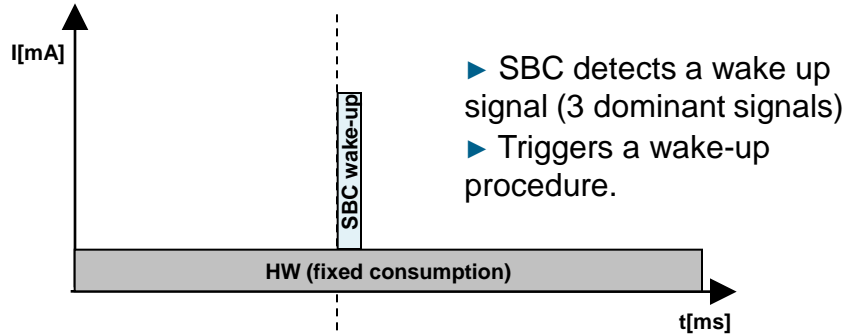
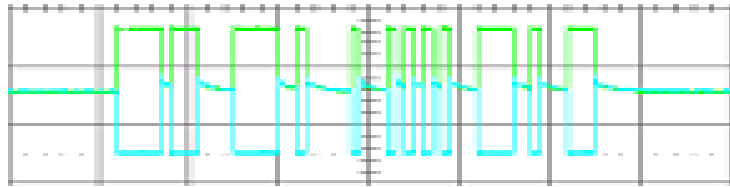
- e.g. DMA, RTC, API, ADC, LINFlex

Example: Body Controller Module Typical Scenario



Managing Communication Wake-Ups

High speed CAN



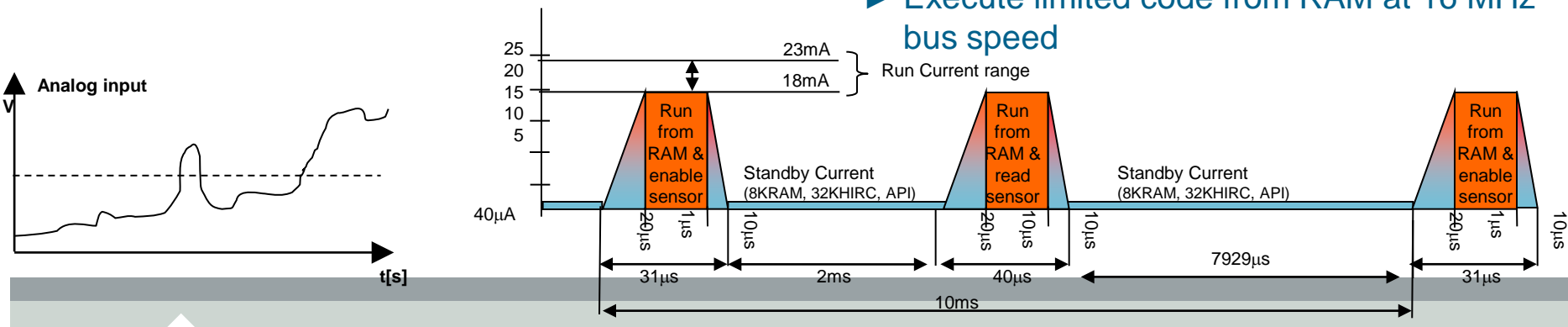
Monitoring Fast Analog Signals

Scenario:

- Measure 3 'fast' analog inputs and check values (e.g., 'fast' temperature sensors)
- read the state of 5 port inputs and check values
- Only continue full power-up if values breach pre-defined conditions
- ▶ Timing and absolute accuracy of initial measurements is not critical
- ▶ ADC function is active for approximately 9 us (3 us per conversion) in every 10 ms

Proposed solution:

- ▶ Use approach: *STANDBY* → *RUN* → *STANDBY* → *RUN*
- ▶ Utilize *STANDBY* and retain 8K of RAM
- ▶ Utilize an API (Application Programming Interface) to wake-up periodically every 10ms and transition into *RUN*
- ▶ Clock the API with the on-chip 128 KHz IRC (very low power Internal RC oscillator)
- ▶ Use a 16 MHz IRC for fast execution, accurate enough for this example application
- ▶ Execute limited code from RAM at 16 MHz bus speed



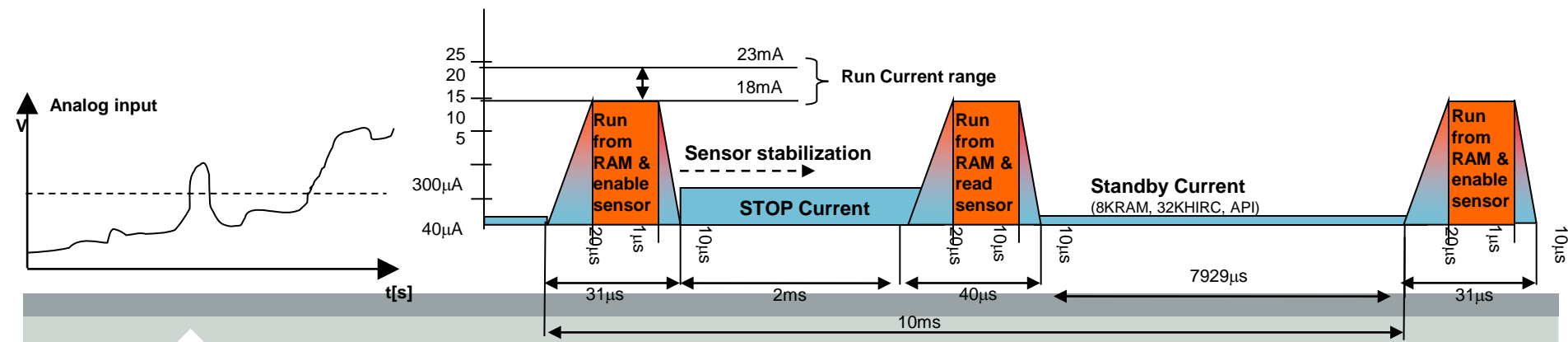
Monitoring Slow Analog Signals

► Scenario:

- Measure 3 analog inputs and check values (for example, 'slow' temperature sensors)
- Sensors are **not** instantly available for reading
- Sensor settling time: 2 ms
- Timing and absolute accuracy of initial measurements is not critical
- ADC function is active for approximately 9 μ s (3 μ s per conversion) in every 10 ms

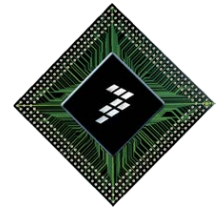
Proposed solution:

- This solution is identical as previous one, except the *STOP* state will be used instead of *STANDBY* during the 'sensor stabilization' period
- Use approach: *STANDBY* \rightarrow *RUN* \rightarrow *STOP* \rightarrow *STANDBY* \rightarrow *RUN*
- Return to *STANDBY* unless pre-defined conditions are exceeded





Software Configuration Example



Code Example – Peripherals Enabled in LPM / RUN Mode

```

// RunPC0 will be use to disable all peripherals in 'high' power mode
ME_RUNPC[0].B.RUN3=0;
ME_RUNPC[0].B.RUN2=0;
ME_RUNPC[0].B.RUN1=0;
ME_RUNPC[0].B.RUN0=0;
ME_RUNPC[0].B.DRUN=1; // all peripherals on in DRUN to execute Peripherals setup
ME_RUNPC[0].B.SAFE=0;
ME_RUNPC[0].B.TEST=0;

// RunPC1 will be use to enable peripherals in 'high' power mode
ME_RUNPC[1].B.RUN3=1;
ME_RUNPC[1].B.RUN2=1;
ME_RUNPC[1].B.RUN1=1;
ME_RUNPC[1].B.RUN0=1;
ME_RUNPC[1].B.DRUN=1; // all peripherals on in DRUN to execute Peripherals setup
ME_RUNPC[1].B.SAFE=0;
ME_RUNPC[1].B.TEST=0;

// LPPC0 will be use to disable all peripherals in low power mode
ME_LPPC[0].B.STANDBY0=0;
ME_LPPC[0].B.STOP0=0;
ME_LPPC[0].B.HALTO=0;

// LPPC1 will be use to enable all peripherals in power mode
ME_LPPC[1].B.STANDBY0=1;
ME_LPPC[1].B.STOP0=1;
ME_LPPC[1].B.HALTO=1;

//Peripherals mode cfg
ME_PCTL[0].R =DBGGr|LPPC0|RUNPC0; //DBG_F:1|LP_CFG:3|RUN_CFG:3
ME_PCTL[1].R =DBGGr|LPPC0|RUNPC0; //
ME_PCTL[2].R =DBGGr|LPPC0|RUNPC0; //
ME_PCTL[3].R =DBGGr|LPPC0|RUNPC0; //
ME_PCTL[4].R =DBGGr|LPPC0|RUNPC1; //DSPIO
ME_PCTL[5].R =DBGGr|LPPC0|RUNPC0; //DSPI1
ME_PCTL[6].R =DBGGr|LPPC0|RUNPC0; //DSPI2
ME_PCTL[7].R =DBGGr|LPPC0|RUNPC0; //DSPI3
ME_PCTL[8].R =DBGGr|LPPC0|RUNPC0; //DSPI4
ME_PCTL[9].R =DBGGr|LPPC0|RUNPC0; //DSPI5
ME_PCTL[10].R =DBGGr|LPPC0|RUNPC0; //
ME_PCTL[11].R =DBGGr|LPPC0|RUNPC0; //
ME_PCTL[12].R =DBGGr|LPPC0|RUNPC0; //LINFlex8
ME_PCTL[13].R =DBGGr|LPPC0|RUNPC0; //LINFlex9
ME_PCTL[14].R =DBGGr|LPPC0|RUNPC0; //
ME_PCTL[15].R =DBGGr|LPPC0|RUNPC0; //
ME_PCTL[16].R =DBGGr|LPPC0|RUNPC0; //FlexCAN0
ME_PCTL[17].R =DBGGr|LPPC0|RUNPC0; //FlexCAN1
ME_PCTL[18].R =DBGGr|LPPC0|RUNPC0; //FlexCAN2
ME_PCTL[19].R =DBGGr|LPPC0|RUNPC0; //FlexCAN3
ME_PCTL[20].R =DBGGr|LPPC0|RUNPC0; //FlexCAN4
ME_PCTL[21].R =DBGGr|LPPC0|RUNPC0; //FlexCAN5
ME_PCTL[22].R =DBGGr|LPPC0|RUNPC0; //
ME_PCTL[23].R =DBGGr|LPPC0|RUNPC1; //DMA_CH_MUX
ME_PCTL[24].R =DBGGr|LPPC0|RUNPC0; //
ME_PCTL[25].R =DBGGr|LPPC0|RUNPC0; //
ME_PCTL[26].R =DBGGr|LPPC0|RUNPC0; //
ME_PCTL[27].R =DBGGr|LPPC0|RUNPC0; //
ME_PCTL[28].R =DBGGr|LPPC0|RUNPC0; //
ME_PCTL[29].R =DBGGr|LPPC0|RUNPC0; //
ME_PCTL[30].R =DBGGr|LPPC0|RUNPC0; //
ME_PCTL[31].R =DBGGr|LPPC0|RUNPC0; //
ME_PCTL[32].R =DBGGr|LPPC0|RUNPC1; //ADC0
ME_PCTL[33].R =DBGGr|LPPC0|RUNPC1; //ADC1
ME_PCTL[34].R =DBGGr|LPPC0|RUNPC0; //
    
```

- ▶ Each peripheral can be enable/disable per mode (done at init)
- ▶ Auto trun off on selected Peripheral when MODE are called during runtime

Session Summary

▶ Low Power Challenges

- ▶ Automotive market demands for low power are increasing:
 - ▶ More body nodes are appearing in every new generation of cars
 - ▶ Each body node has more functionality and features
- ▶ More functionality demands more from the silicon:
 - Smaller geometries to meet increasing demands
 - Smaller technology drives the power curve

▶ MCU Low Power Mechanisms

- ▶ Are there to make it easy
- ▶ Software support

▶ System Low Power Techniques

- ▶ Power up fast
- ▶ Standby as much as possible
- ▶ Intelligent use of available resources

Attention to power is needed
from the **beginning** of system design

