AURIX 2G Standby Controller(SCR) Overview

Thomas
IFCN ATV SMD GC SAE MC
2018/5/2

History List / Changes

- Module Name: Standby Controller (SCR)
- Module Owner: Angel Berrio Moreno
- Slide Target: SCR Overview
- Slide Version: 3.0
- Slide Status: Released
- Last Modification: 23/06/2017
- Revision History:
  - 1.0 First version
  - 2.0 Added specific features of A2G and peripheral description
  - 3.0 Added changes available in the latest version of the IP

Agenda

1. Overview
2. System Architecture : Standby Domain
3. SCR Architecture
4. Debug system
5. SCR SW Framework

Agenda

1. Overview
2. System Architecture : Standby Domain
3. SCR Architecture
4. Debug system
5. SCR SW Framework

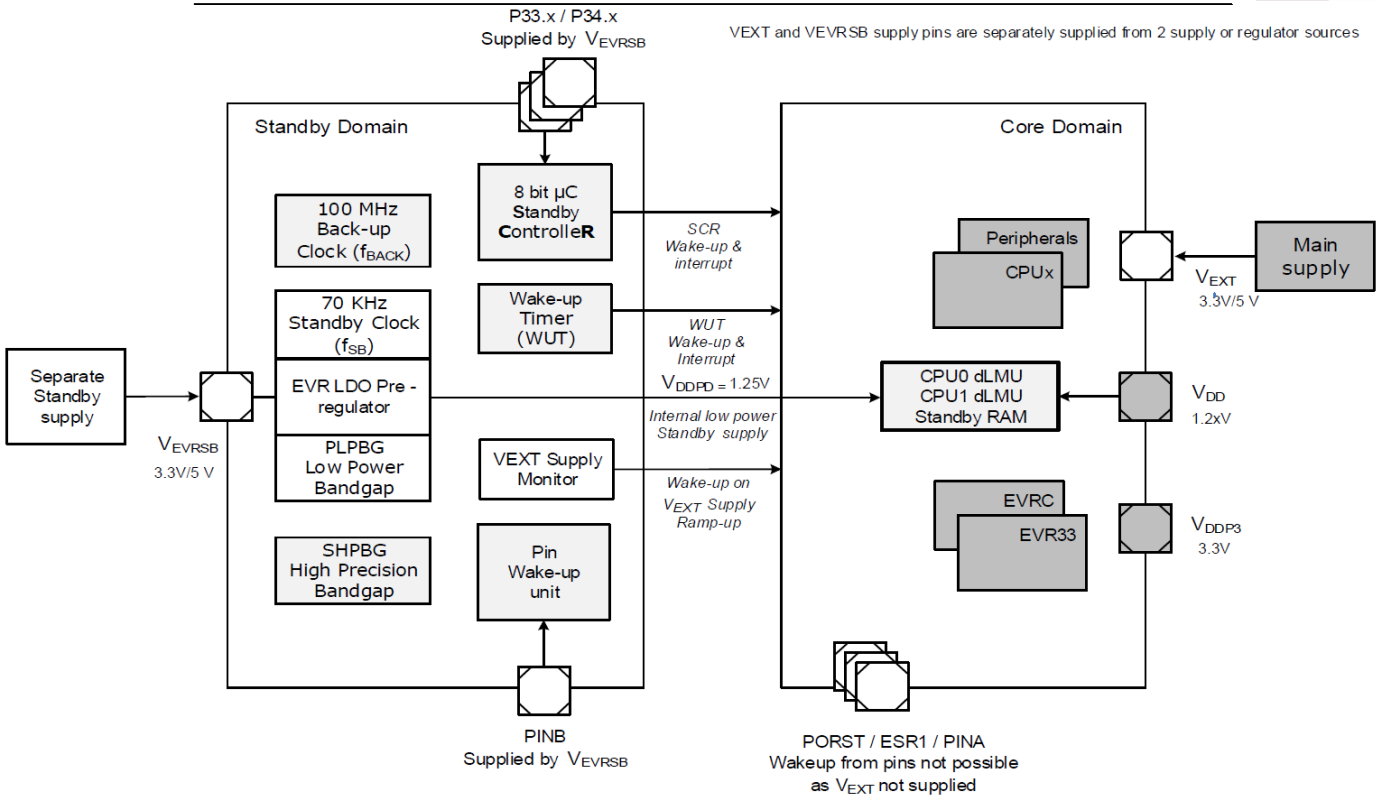
Overview

- Support High Performance and low power standby using one silicon
- Two separate core domains
  - A “high performance” domain (Tricore)
    - only needed if engine is on, otherwise, switched off
  - A “low power” domain (SCR)
    - Permanently on
    - All features available to do supervising tasks.
    - Wakeup of high performance domain only needed, when the car is started
- Two separate IO domains
  - A domain is powered during standby mode while all other pins are switched off

Agenda

1. Overview
2. System Architecture : Standby Domain
3. SCR Architecture
4. Debug system
5. SCR SW Framework

System Architecture : Standby Domain



- Standby Mode: Low Power Mode
- Cores, Peripherals powered off
- Only Standby Domain is powered
- Wake-up possible from different sources. Eg: Ports/ESR, SCR etc.

- Standby RAM:
  - CPU0 and CPU1 dLMU can be kept powered on
  - Data retained during standby
- Pads: Pad control possible via SCR
- From 100MHz to 70kHz clock

Agenda

1. Overview
2. System Architecture : Standby Domain
3. SCR Architecture
4. Debug system
5. SCR SW Framework

SCR Architecture : SCR in the System

- Standby Controller: 8-bit Controller
  - XC800 Core + peripherals
  - 8KB XRAM for code / data accesible via FPI slave interface

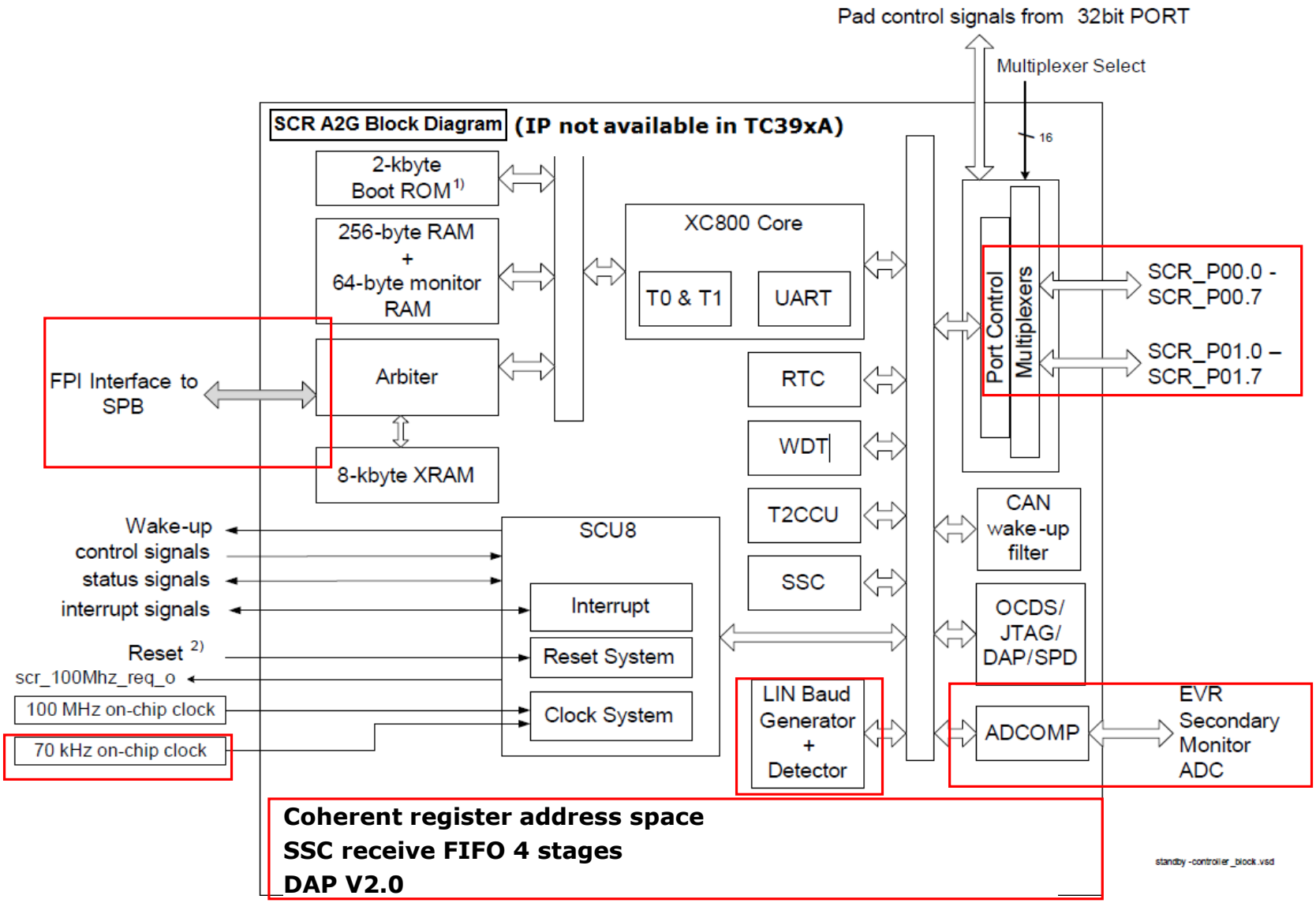
- Some signals to/from SCU
- Interrupts to/from main system

- Ports shared between Tricore & SCR
  - Can select the owner of the port

SCR Architecture

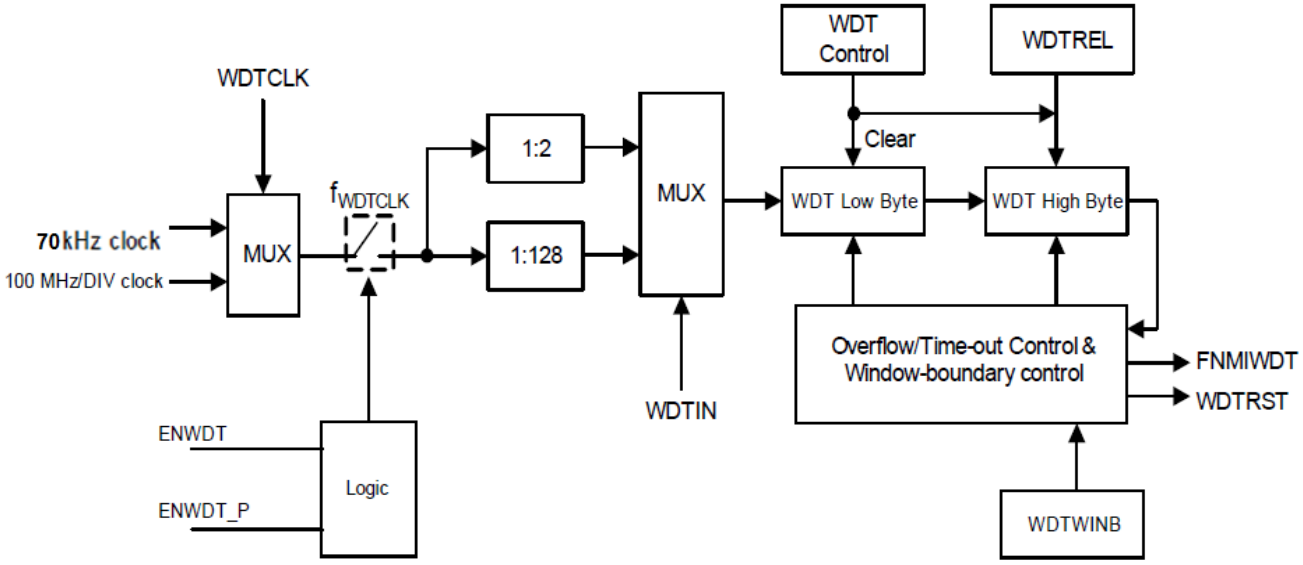
- Supports various supervising tasks
  - Real-time clock for periodic wake-up
  - LIN possible via UART
  - SPI via SSC
  - Shared I/O pins including wake-up pins
  - Wakeup-CAN filter
  - On-chip SRAM for data and code
- Very low current consumption ~µA range (70kHz)
- Existing proven solution: XC800 based 8-Bit controller

SCR Architecture : A1G vs A2G



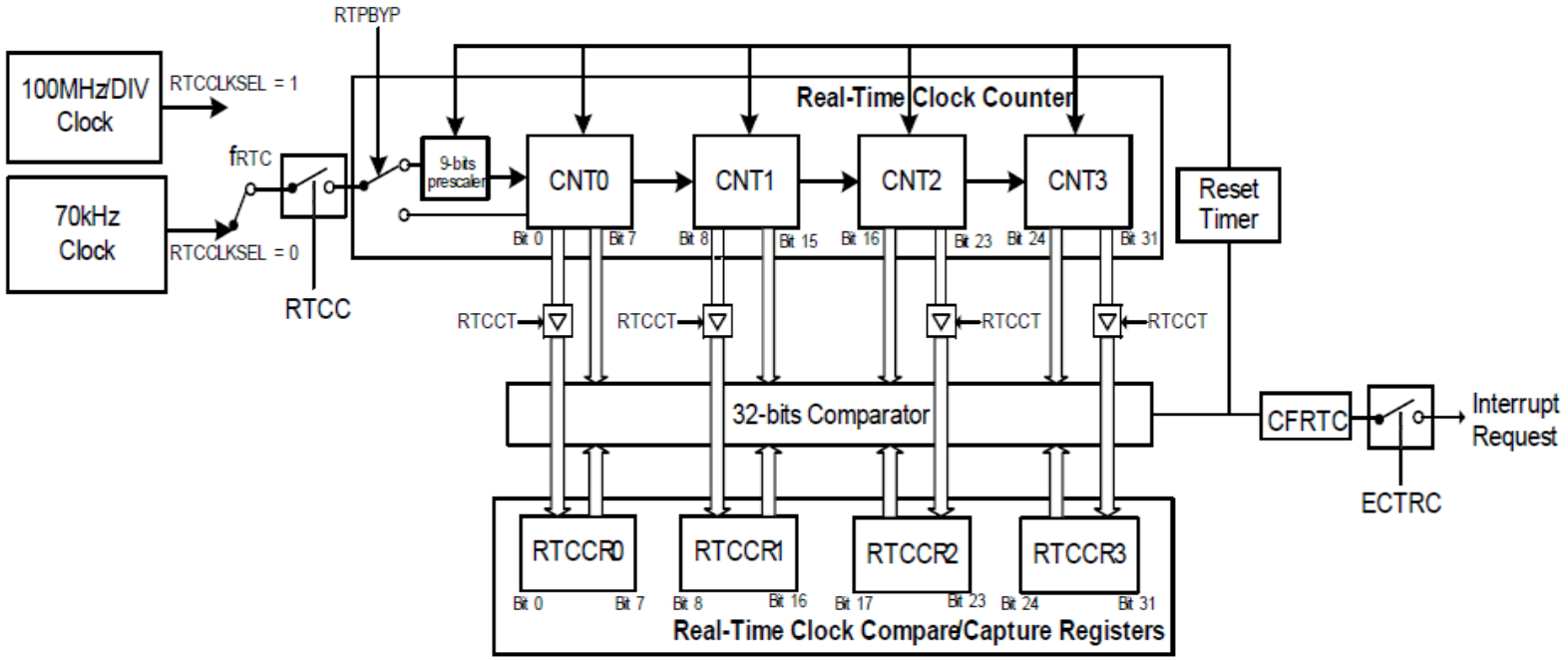
SCR Architecture – Watchdog Timer

- 16-bit Watchdog Timer
- Programmable reload value for upper 8 bits of timer and window boundary
- Clock source from either the 70kHz clock or the 100MHz/DIV clock
- Selectable input frequency of fWDTCLK/2 or fWDTCLK/128



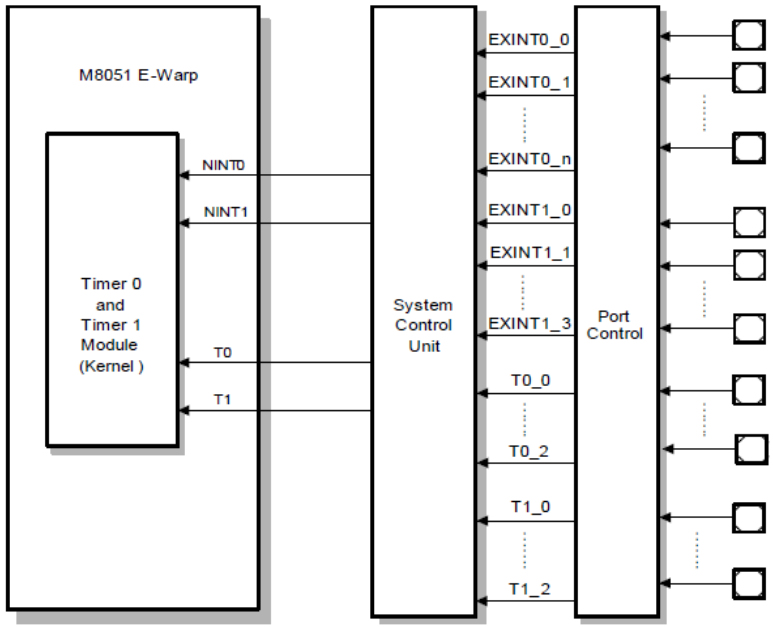
SCR Architecture – Real-Time Clock

- Periodic Wake-up Mode using either the 70kHz clock or the 100MHz/DIV clock
- Wake-up source during standby mode

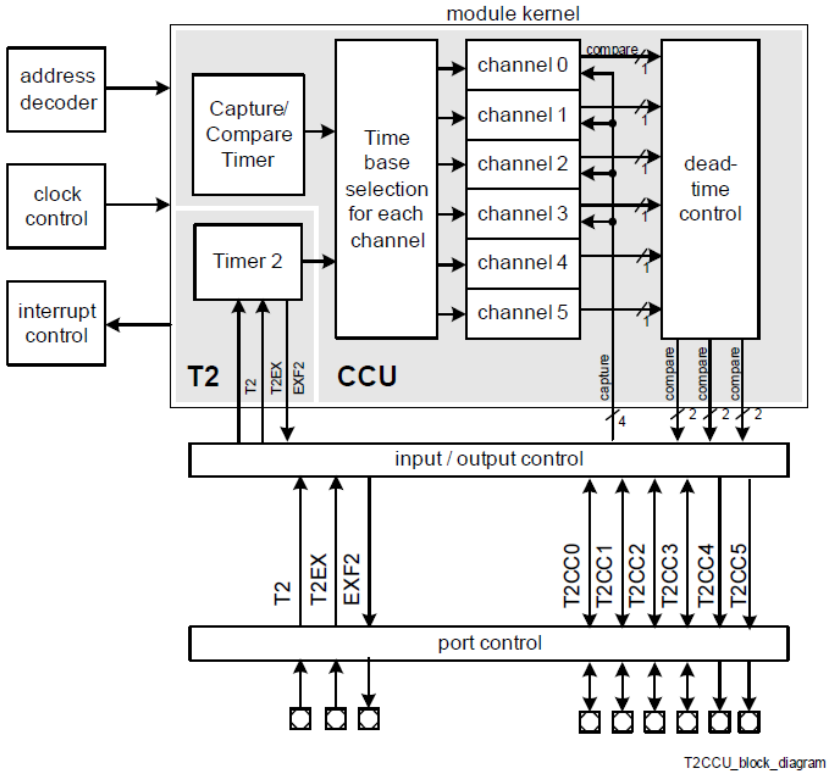


SCR Architecture – Timer & Capture

- Timer 0, Timer 1 and Timer 2
  - 8-bit(with auto reload), two 8-bit timers, 13-bit timer and 16-bit timer modes

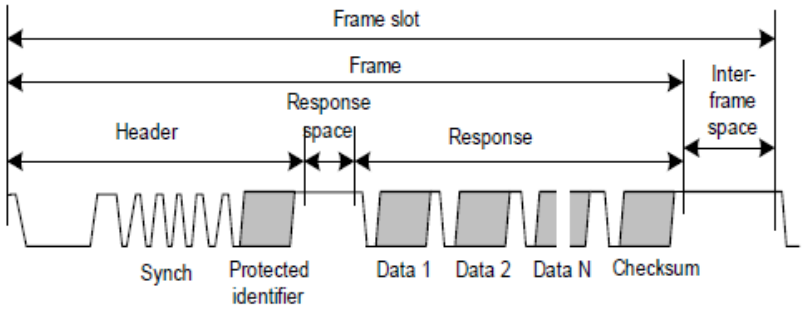


- T2CCU
  - 16-bit resolution, 6 compare channels, shadow register for each compare register

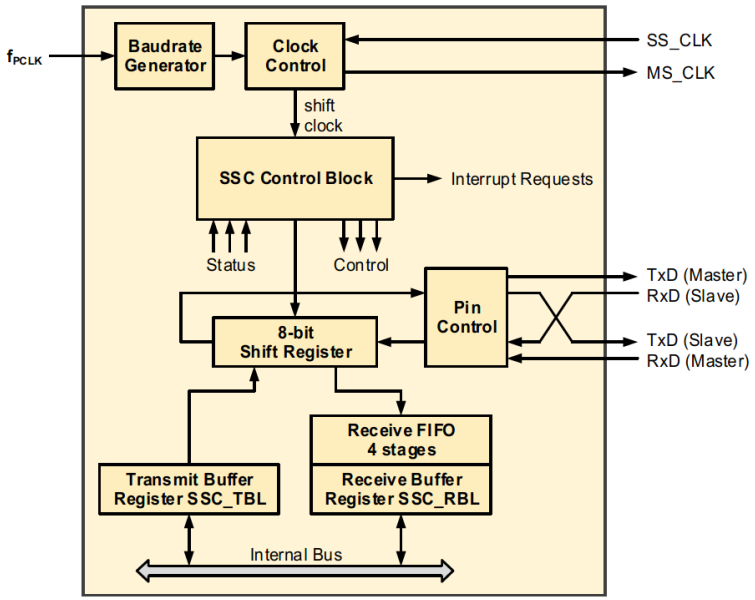


SCR Architecture – Serial Communication

- UART/LIN
  - 8 and 9 bits UART with configurable Baud Rate
  - LIN Automatic Synchronization to the Host
  - LIN Baud Rate Detection

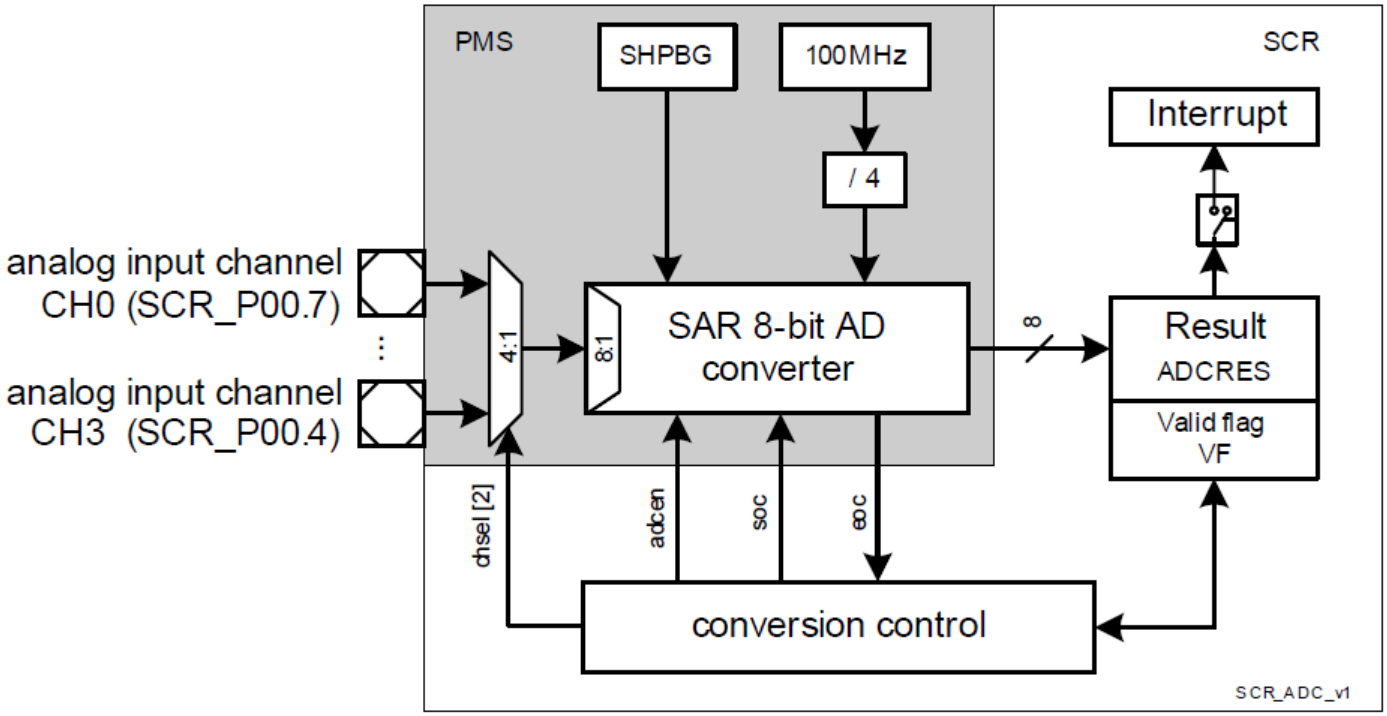


- SSC (SPI)
  - Full-duplex and half-duplex sync. communication up to 2MB
  - Master and Slave support, Flexible data format
  - Receive FIFO (4 stages)



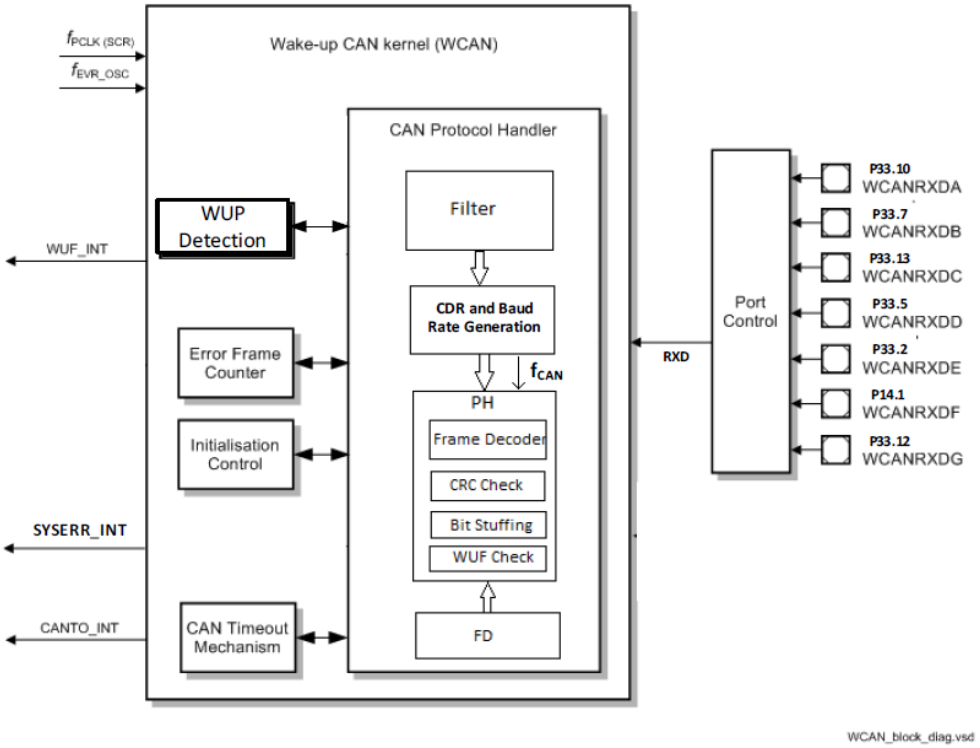
SCR Architecture – ADC Comparator Unit

- 8-bit resolution, LSB = 23,077mV, Range = 0 - 5861,54mV.
- Accuracy / TUE = +-3LSB
- 4 ADC channels
- 200ns sampling time + 400ns conversion time.



SCR Architecture – Wake-up CAN Filter

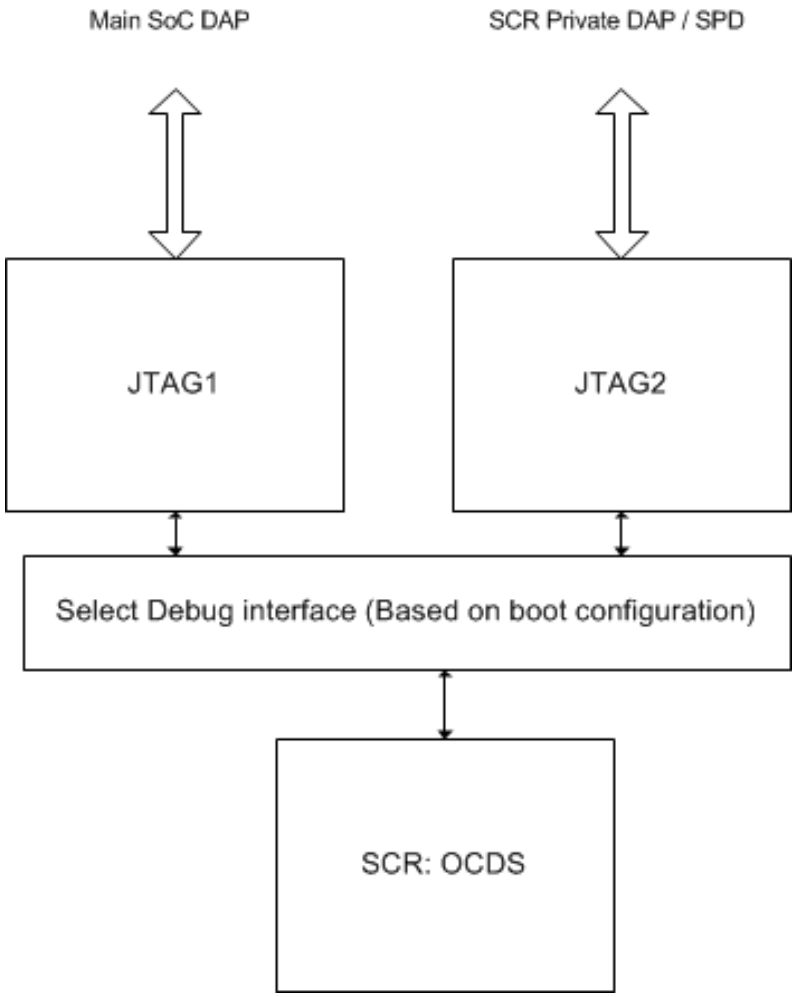
- Receive and Wake-up CAN operation according to ISO11898-6
- 1 Wake-up CAN node mapped to CAN 0
- Tolerant to CAN FD frames
- Data transfer rates guaranteed until 500kBaud



Agenda

1. Overview
2. System Architecture : Standby Domain
3. SCR Architecture
4. Debug system
5. SCR SW Framework

Debug system



Debug system: Normal running mode

- Debug both TriCore and SCR **in parallel**.
  - TriCore running some code – All normal debug functions available.
  - SCR:
    - Can program XRAM with some code.
    - Read/write XRAM + direct / indirect addressed SFRs
    - Read/write 8051 CPU registers.
    - Access SCR peripherals.
    - Normal debug functions: Breakpoints etc.
    - Access via SPD/DAP0 or DAP1

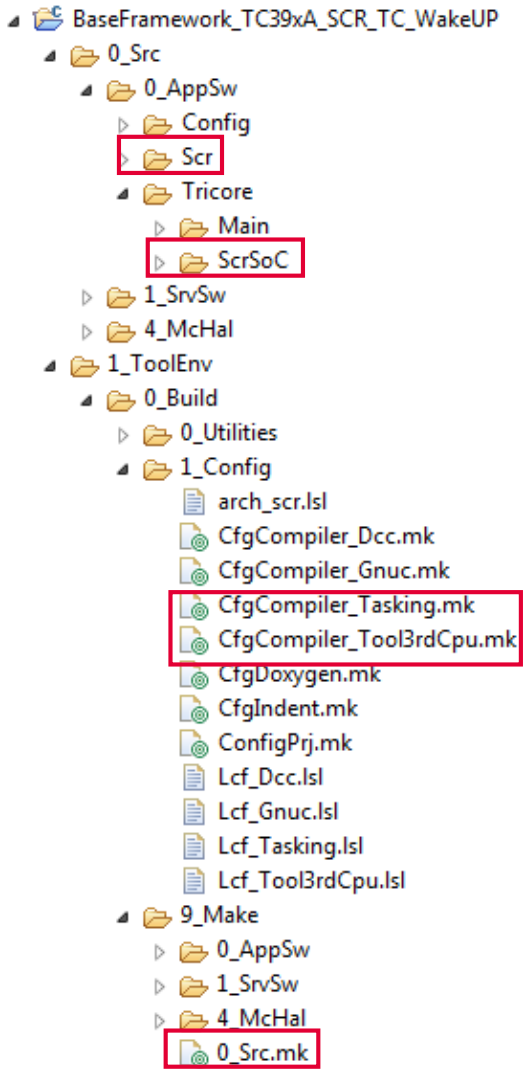
Debug system: Standby mode

- In standby mode, interface via main SoC DAP is not available (Main system in standby)
- Therefore, debug access via SCR private DAP pins.
  - Similar to previous:
    - Can program XRAM with some code.
    - Read/write XRAM + direct / indirect addressed SFRS
    - Read/write 8051 CPU registers.
    - Access SCR peripherals.
    - Normal debug functions: Breakpoints etc.
    - Access via SPD/DAP0 or DAP1

Agenda

1. Overview
2. System Architecture : Standby Domain
3. SCR Architecture
4. Debug system
5. SCR SW Framework

SCR SW Framework

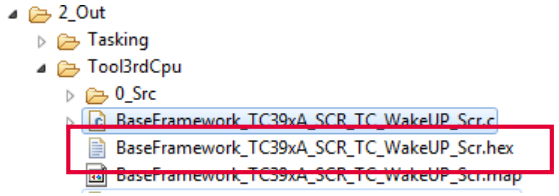


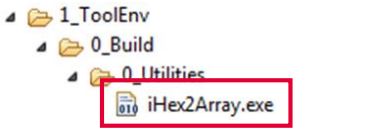
- SCR integrated as 3rd-CPU in SW Framework (Scr)
- Tricore/ScrSoC: SoC side of the code
- Tasking compiler support (Both Tricore and 8051)
  - 0_Src.mk including both make files:
    - `TOOLCHAIN_DIR_MAIN:= …\TASKING\TriCore v6.0r1\ctc`
    - `TOOLLIB_DIR_THIRD:= …\TASKING\TriCore v6.0r1\c51\lib`
  - CfgCompiler_Tool3rdCpu.mk:

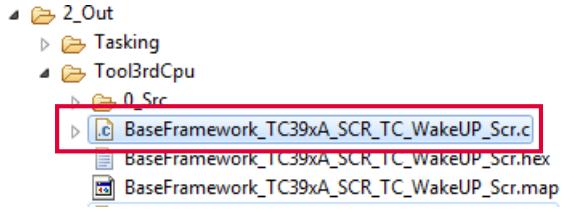
```
CARRAY="1_ToolEnv/0_Build/0_Utilities/iHex2Array.exe“
$(SCR_ARRAY): $(ELF_BIN_THIRD)
$(CARRAY) $(<:.elf=.hex) $(@:.o=.c)
$(CC) -o $(@:.o=.src) $(@:.o=.c) $(CC_OPTS)
$(AS) -o $@ $(@:.o=.src) $(ASM_OPTS)
```

SCR SW Framework: Compilation (SCR code)

- Generate intel-hex file in Out folder
- Simple Parser converts iHex to C-array
- C-array compiled into Tricore code
  - Included from
    0_Src/0_AppSw/Tricore/ScrSoC/IfxScr_main.c







BaseFramework_TC39xA_SCR_TC_WakeUP_Scr.c

```
/* Automatically generated array containing the SCR program */
unsigned int scrProgramSize = 385;
unsigned char scrProgram[385] = {

    0x2,    0x0,    0x6,    0x80,   0xfe,   0x22,   0x75,   0xd0,   0x0,    0x78,   0x0,    0xe8,
    0xf5,   0xa0,   0x75,   0x81,   0xd,    0x75,   0x9,    0xff,   0x75,   0x8,    0x1,    0x12,
    0x0,    0x5,    0xe4,   0xff,   0xfe,   0x12,   0x0,    0x24,   0x12,   0x0,    0x3,    0x32,
    0x12,   0x0,    0x76,   0x80,   0xfe,   0xc0,   0xe0,   0xc0,   0xd0,   0x75,   0xd0,   0x0,
    0xe4,   0xf5,   0xf5,   0x75,   0xfe,   0xdd,   0x75,   0xf6,   0x80,   0xd0,   0xd0,   0xd0,
    0xe0,   0x32,   0xc0,   0xd0,   0x75,   0xd0,   0x0,    0x85,   0xfe,   0xa,    0x5,    0xa,
    0x85,   0xa,    0xfe,   0x75,   0xf6,   0x80,   0xd0,   0xd0,   0x32,   0xe4,   0xf5,   0xeb,
    0x75,   0xea,   0xf,    0x75,   0xe9,   0x42,   0x75,   0xe7,   0x40,   0x75,   0x95,   0x17,
    0x80,   0xfe,   0x0,    0x0,    0x0,    0x0,    0x0,    0x0,    0x0,    0x0,    0x0,    0x2,
    0x0,    0x3e,   0x0,    0x0,    0x0,    0x0,    0x0,    0x2,    0x0,    0x29,   0xe4,   0xf5,
    0xb,    0x75,   0xc,    0x40,   0xf5,   0x8f,   0xf5,   0xf1,   0xf5,   0xf5,   0x75,   0xfe,
    0xac,   0x74,   0x80,   0xf5,   0xf6,   0xf5,   0xa8,   0xf5,   0xe8,   0x75,   0xfe,   0xa1,
    0xf5,   0xf6,   0x75,   0xf1,   0x1,    0x75,   0xfa,   0x2,    0xe4,   0xf5,   0xf1,   0x12,
```

```
void IfxScr_copyProgram()
{
    uint16 addrOffset = 0;
    uint16 data = 0;

    uint8* addr = ((volatile uint8*)(0xF0240000));
    uint8 i = 0;
    uint16* addr16 = ((volatile uint16*)(0xF0241FF8));

    //SCR program transfer
    for (addrOffset = 0; addrOffset < scrProgramSize; addrOffset++)
    {
        *addr = scrProgram[addrOffset];
        addr++;
    }

    for(i = 0; i < 4; i++){
        *(addr16 + i) = 0xAA55;
    }
}
```

Part of your life. Part of tomorrow.

Infineon