Continental
The Future in Motion

You Choose the Way.
We Make the Most of it.

SensePlanAct

Analysis of safety and real time behavior using cycle accurate ARM models

Agenda

1 Objective

2 Motivation

3 Why cycle accurate models

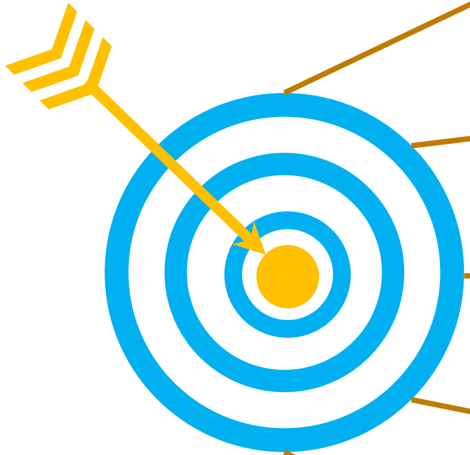
4 Platform setup

5 Block diagram

6 COSIDE® support

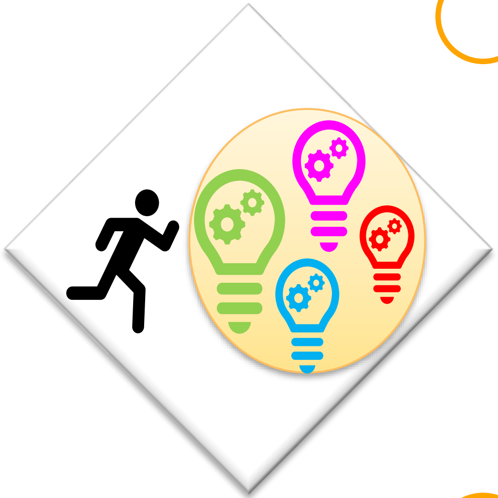
7 Outcome

Objective



- Perform analysis of Arm® Cortex®-M7 core-based SOC.
- Rebuild Cortex®-M7 based virtual SOC.
- Port a matured software into Cortex®-M7 platform.
- Run Software with different hardware and compiler configurations.
- Perform analysis on Safety and real time behavior using cycle accurate models.

Motivation



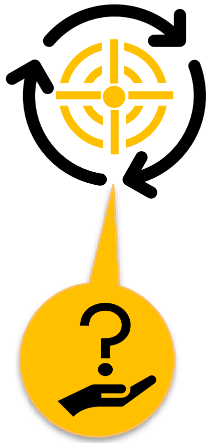
In the past, we realized that the hardware performance did not suit requirements at the last stage.

Memory latency caused issues with performance.

Memory bottleneck and problem with system integration, resulted in a bad hit on the performance.

To discover these problems early, we wanted to analyze safety and real time behavior on cycle accurate models.

Why cycle accurate models



As a part of safety requirement, we have some timing requirements that are too stringent.

To study how quick the code would execute if it is configured in either TCM or memory.

To measure and check the hardware requirements using software and identify where exactly we need higher accuracy.

We wanted to transfer huge data from traffic generator to the core and measure the time consumed.

In fast models, we cannot validate these kind of scenarios and there are high chances that we might get FALSE PASS.

Platform Setup

Platform Configuration:

- ❖ Core: Arm® Cortex®-M7
- ❖ SystemC models : Cycle accurate models from Arm®
- ❖ Interconnect : NIC 400
- ❖ SRAM : BP140
- ❖ Traffic Generator : Conti specific
- ❖ TCM : Both ITCM and DTCM are used
- ❖ Cache : Both Instruction and Data Cache are used
- ❖ Floating point : Single precision floating point

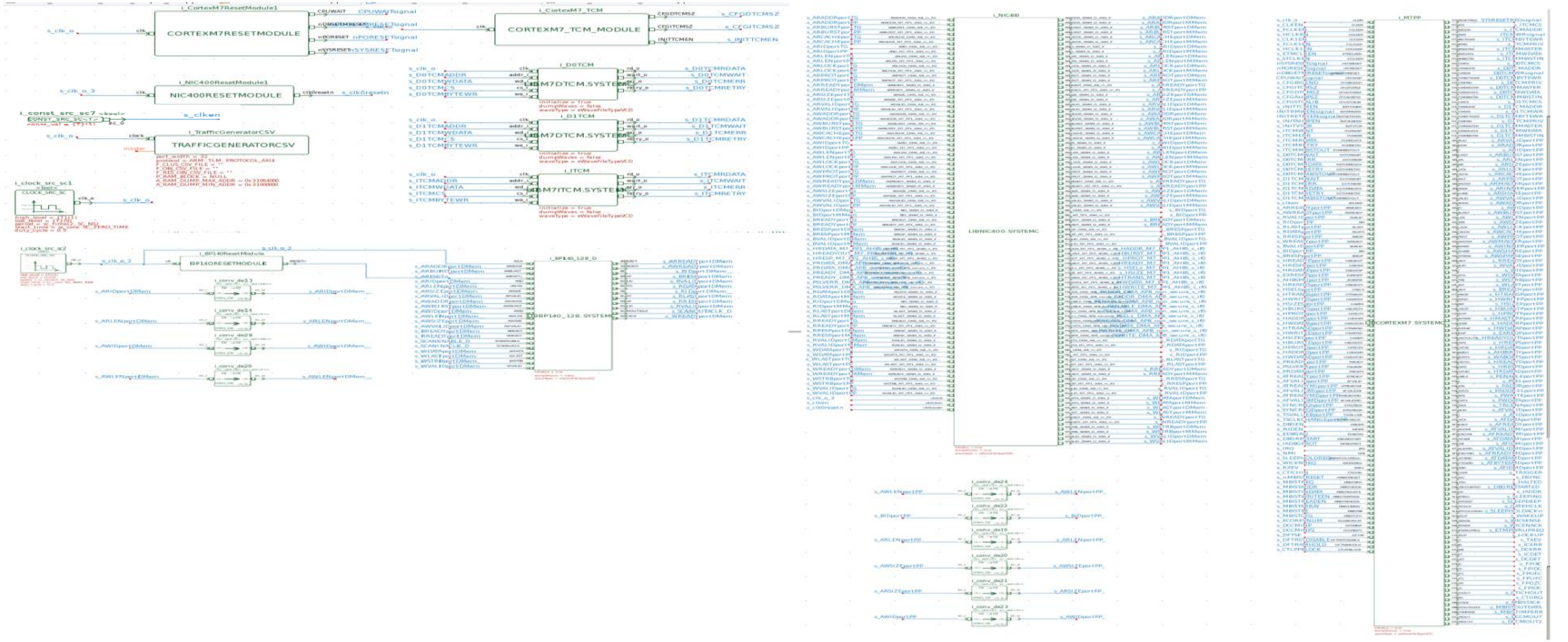
Memory Map:

Address	Module	Remarks
0	I TCM (private)	Exception vectors
20000000	D TCM (private)	Stack and heap
30000000	Global RAM0 (BP140)	Application code
31000000	Global RAM1 (BP140)	Application Data

Tool chain:

- ❖ IDE used : COSIDE® 2.7 RC
- ❖ Compiler : GNU Arm Embedded Toolchain

Block diagram

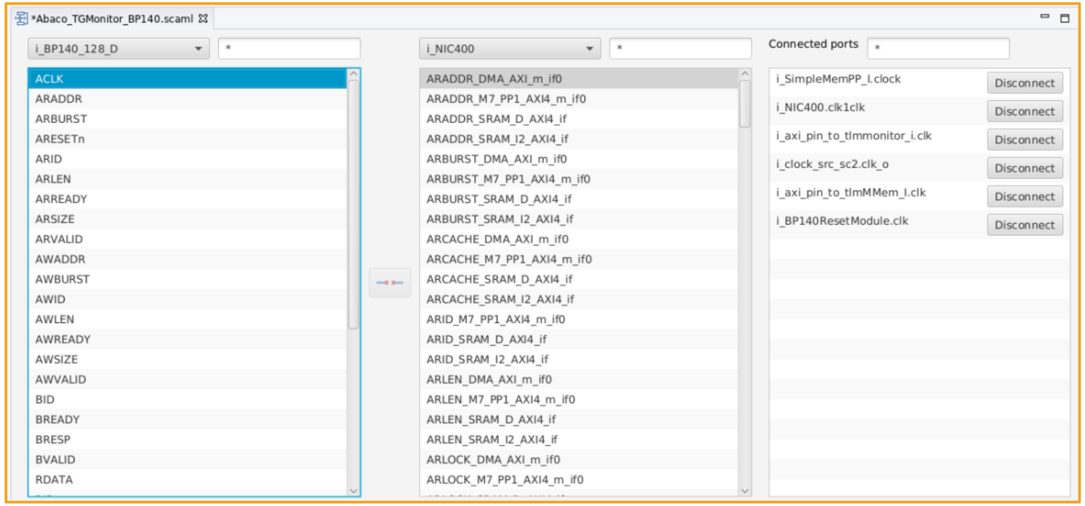


Business Area Autonomous Mobility and Safety (AMS) /
Business Unit ADAS / IC Solutions
Public

November 4, 2020
Chethan Muralidhara, Dr.-Ing. Sacha Loitz

COSIDE® support

- Most of the models were pin level models. So we had to do several connections, re-integrations when we receive the updated models. Interface definitions and wiring editor feature helped us overcome the connection issues.



*Abaco_TGMonitor_BP140.scaml

i_BP140_128_D | *

ACLK
ARADDR
ARBURST
ARESETn
ARID
ARLEN
ARREADY
ARSIZE
ARVALID
AWADDR
AWBURST
AWID
AWLEN
AWREADY
AWSIZE
AWVALID
BID
BREADY
BRESP
BVALID
RDATA

i_NIC400 | *

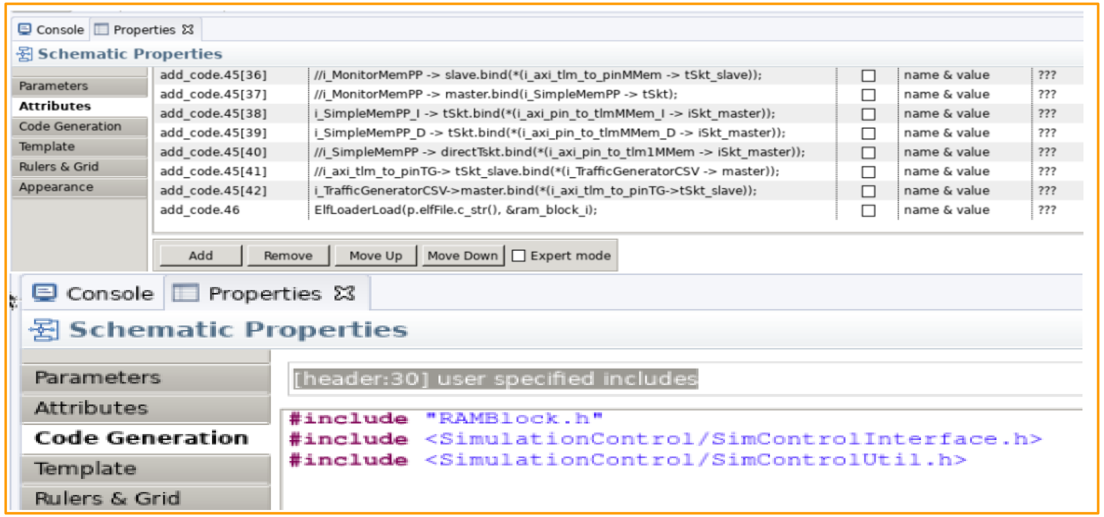
ARADDR_DMA_AXI_m_if0
ARADDR_M7_PP1_AXI4_m_if0
ARADDR_SRAM_D_AXI4_if
ARADDR_SRAM_I2_AXI4_if
ARBURST_DMA_AXI_m_if0
ARBURST_M7_PP1_AXI4_m_if0
ARBURST_SRAM_D_AXI4_if
ARBURST_SRAM_I2_AXI4_if
ARCACHE_DMA_AXI_m_if0
ARCACHE_M7_PP1_AXI4_m_if0
ARCACHE_SRAM_D_AXI4_if
ARCACHE_SRAM_I2_AXI4_if
ARID_M7_PP1_AXI4_m_if0
ARID_SRAM_D_AXI4_if
ARID_SRAM_I2_AXI4_if
ARLEN_DMA_AXI_m_if0
ARLEN_M7_PP1_AXI4_m_if0
ARLEN_SRAM_D_AXI4_if
ARLEN_SRAM_I2_AXI4_if
ARLOCK_DMA_AXI_m_if0
ARLOCK_M7_PP1_AXI4_m_if0

Connected ports | *

i_SimpleMemPP_I.clock	Disconnect
i_NIC400.clk1clk	Disconnect
i_axi_pin_to_tlmmonitor_i.clk	Disconnect
i_clock_src_sc2.clk_o	Disconnect
i_axi_pin_to_tlmMMem_I.clk	Disconnect
i_BP140ResetModule.clk	Disconnect

COSIDE® support (Contd..)

- We were able to generate the manual code in few places by using the ‘attributes’ and ‘code generation’ feature. Using this feature, we were able to successfully bind few TLM ports.



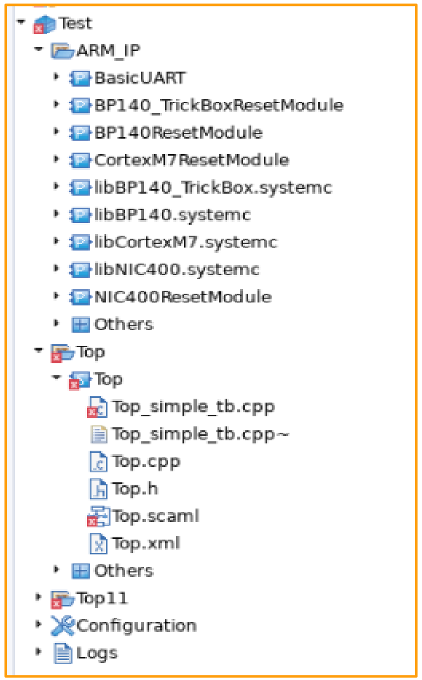
add_code.45[36]	//i_MonitorMemPP -> slave.bind(*(i_axi_tlm_to_pinMMem -> tSkt_slave));	☐	name & value	???
add_code.45[37]	//i_MonitorMemPP -> master.bind(i_SimpleMemPP -> tSkt);	☐	name & value	???
add_code.45[38]	i_SimpleMemPP_I -> tSkt.bind(*(i_axi_pin_to_tlmMMem_I -> iSkt_master));	☐	name & value	???
add_code.45[39]	i_SimpleMemPP_D -> tSkt.bind(*(i_axi_pin_to_tlmMMem_D -> iSkt_master));	☐	name & value	???
add_code.45[40]	//i_SimpleMemPP -> directTskt.bind(*(i_axi_pin_to_tlm1MMem -> iSkt_master));	☐	name & value	???
add_code.45[41]	//i_axi_tlm_to_pinTG-> tSkt_slave.bind(*(i_TrafficGeneratorCSV -> master));	☐	name & value	???
add_code.45[42]	i_TrafficGeneratorCSV->master.bind(*(i_axi_tlm_to_pinTG->tSkt_slave));	☐	name & value	???
add_code.46	ElfLoaderLoad(p.elfFile.c_str(), &ram_block_i);	☐	name & value	???

[header:30] user specified includes

```
#include "RAMBlock.h"
#include <SimulationControl/SimControlInterface.h>
#include <SimulationControl/SimControlUtil.h>
```

COSIDE® support (Contd..)

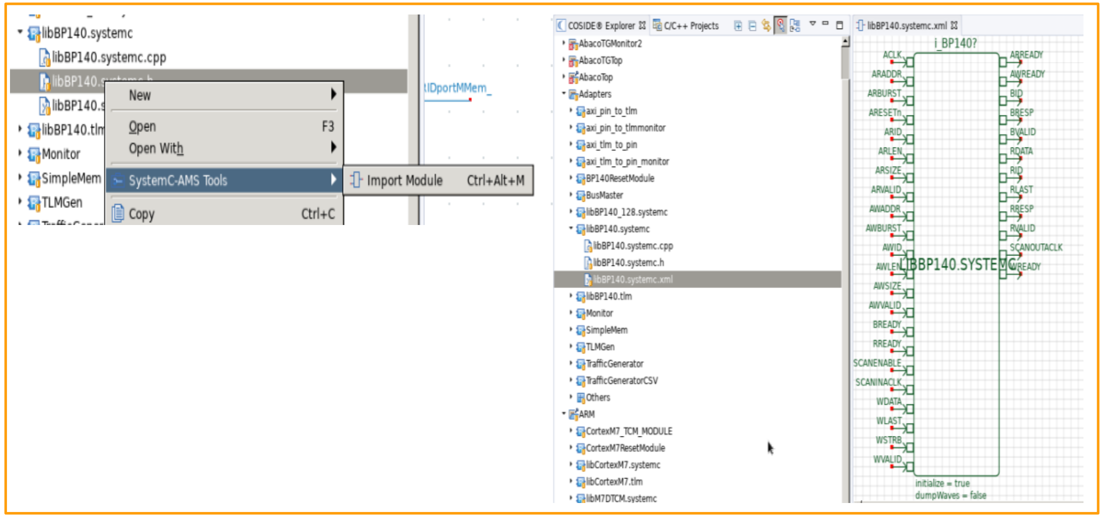
- With many basic options that are provided, we were able to import models from Arm®, create custom models, integrate all of those and create our custom SOC platform.



```
▾ Test
  ▾ ARM_IP
    ▸ BasicUART
    ▸ BP140_TrickBoxResetModule
    ▸ BP140ResetModule
    ▸ CortexM7ResetModule
    ▸ libBP140_TrickBox.systemc
    ▸ libBP140.systemc
    ▸ libCortexM7.systemc
    ▸ libNIC400.systemc
    ▸ NIC400ResetModule
    ▸ Others
  ▾ Top
    ▾ Top
        Top_simple_tb.cpp
        Top_simple_tb.cpp~
        Top.cpp
        Top.h
        Top.scaml
        Top.xml
    ▸ Others
  ▸ Top11
  ▸ Configuration
  ▸ Logs
```

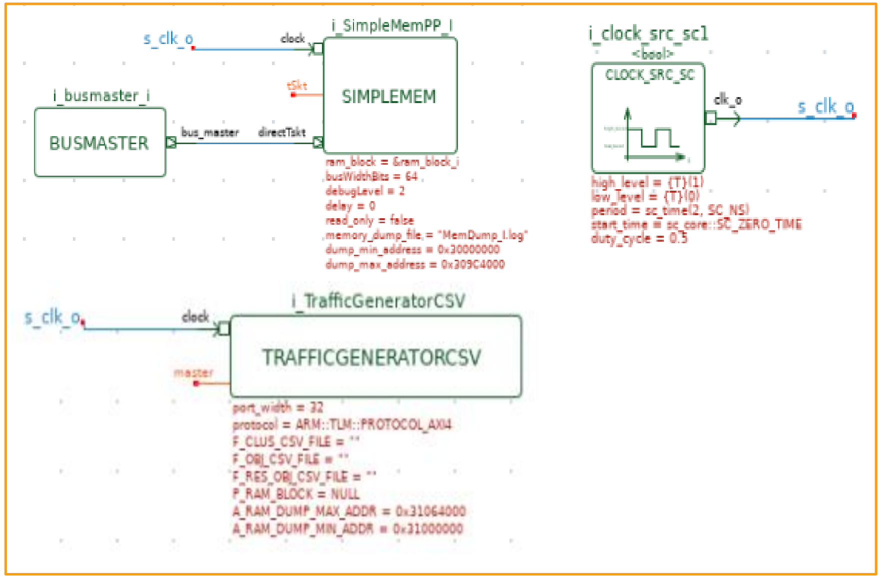
COSIDE® support (Contd..)

- Automatic generation of models(xml and symbols) from the header files, i.e. SystemC projects were automatically created from the header files provided by Arm®.



COSIDE® support (Contd..)

- Templated module and parameters support the configuration of models with different bus width, protocol selection, port width, different clock configurations, start and end address of memory, VCD generation control, log file generations etc.



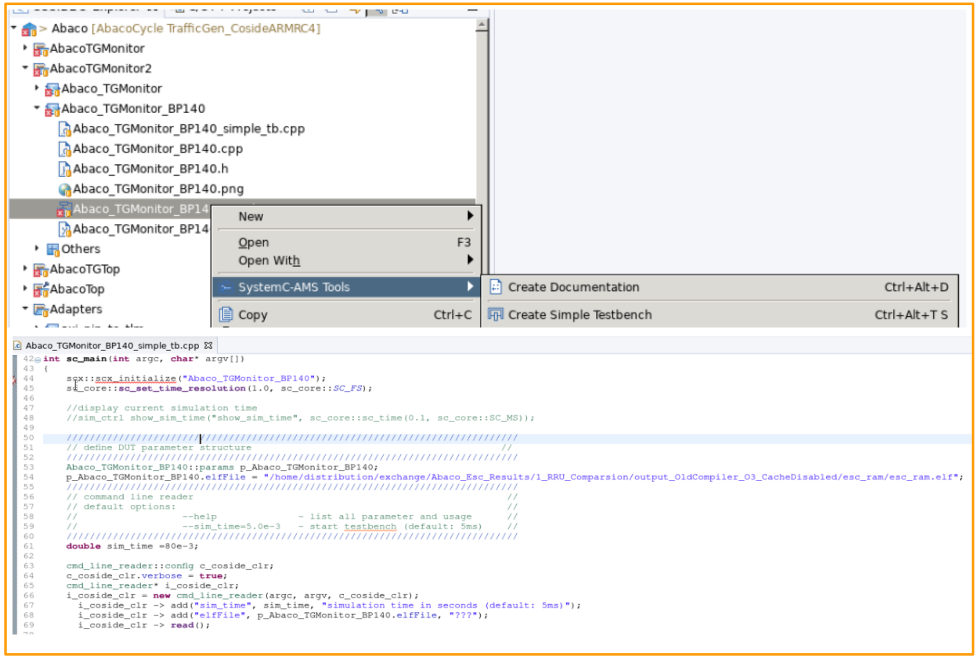
```
ram_block = &ram_block_i
busWidthBits = 64
debugLevel = 2
delay = 0
read_only = false
memory_dump_file = "MemDump_l.log"
dump_min_address = 0x30000000
dump_max_address = 0x309C4000
```

```
high_level = (T)(1)
low_level = (T)(0)
period = sc_time(2, SC_NS)
start_time = sc_core::SC_ZERO_TIME
duty_cycle = 0.5
```

```
port_width = 32
protocol = ARM::TLM::PROTOCOL_AXI4
F_CLUS_CSV_FILE = ""
F_OBJ_CSV_FILE = ""
F_RES_OBJ_CSV_FILE = ""
P_RAM_BLOCK = NULL
A_RAM_DUMP_MAX_ADDR = 0x31064000
A_RAM_DUMP_MIN_ADDR = 0x31000000
```

COSIDE® support (Contd..)

- Automatic test bench generation with support of parameters lead to run various software without rebuilding the platform.



Abaco [AbacoCycle TrafficGen_CosideARMRC4]
AbacoTGMonitor
AbacoTGMonitor2
Abaco_TGMonitor
Abaco_TGMonitor_BP140
Abaco_TGMonitor_BP140_simple_tb.cpp
Abaco_TGMonitor_BP140.cpp
Abaco_TGMonitor_BP140.h
Abaco_TGMonitor_BP140.png
Abaco_TGMonitor_BP14
Abaco_TGMonitor_BP14
Others
AbacoTGTop
AbacoTop
Adapters
New
Open F3
Open With
SystemC-AMS Tools
Copy Ctrl+C
Create Documentation Ctrl+Alt+D
Create Simple Testbench Ctrl+Alt+T S
Abaco_TGMonitor_BP140_simple_tb.cpp

```cpp
int sc_main(int argc, char* argv[])
{
    scx::scx_initialize("Abaco_TGMonitor_BP140");
    sc_core::sc_set_time_resolution(1.0, sc_core::SC_FS);

    //display current simulation time
    //sim_ctrl show_sim_time("show_sim_time", sc_core::sc_time(0.1, sc_core::SC_MS));

    ////////////////////////////////////////////////////////////////////////////////
    // define DUT parameter structure                                            //
    ////////////////////////////////////////////////////////////////////////////////
    Abaco_TGMonitor_BP140::params p_Abaco_TGMonitor_BP140;
    p_Abaco_TGMonitor_BP140.elfFile = "/home/distribution/exchange/Abaco_Esc_Results/1_RRU_Comparsion/output_OldCompiler_O3_CacheDisabled/esc_ram/esc_ram.elf";
    ////////////////////////////////////////////////////////////////////////////////
    // command line reader                                                        //
    // default options:                                                           //
    //                   --help              - list all parameter and usage      //
    //                   --sim_time=5.0e-3   - start testbench (default: 5ms)    //
    ////////////////////////////////////////////////////////////////////////////////
    double sim_time =80e-3;

    cmd_line_reader::config c_coside_clr;
    c_coside_clr.verbose = true;
    cmd_line_reader* i_coside_clr;
    i_coside_clr = new cmd_line_reader(argc, argv, c_coside_clr);
      i_coside_clr -> add("sim_time", sim_time, "simulation time in seconds (default: 5ms)");
      i_coside_clr -> add("elfFile", p_Abaco_TGMonitor_BP140.elfFile, "???");
      i_coside_clr -> read();
```

Outcome

✓Analyzed whether Arm® Cortex®-M7 features fit our needs.

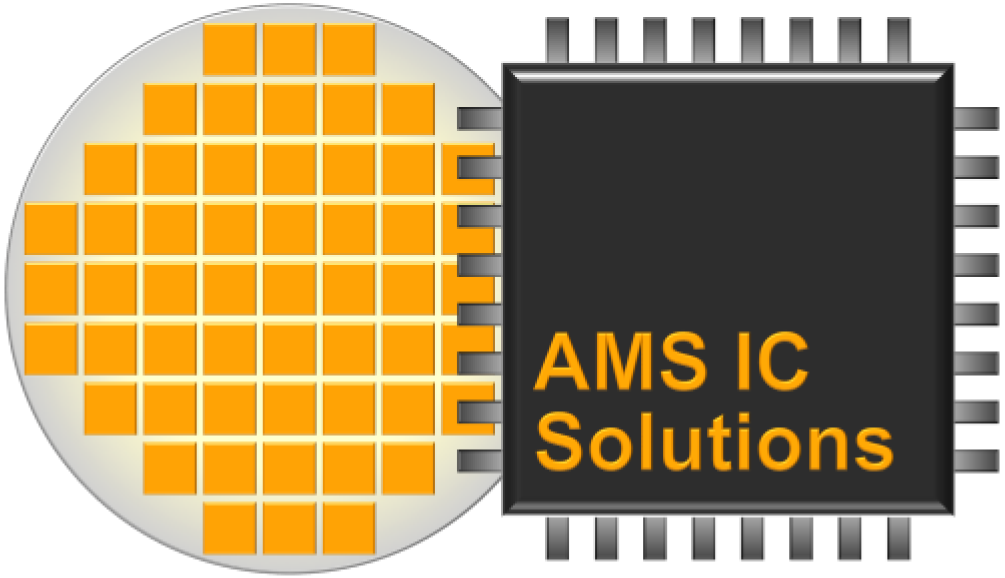
✓Observed the behavior of software at various memory speeds and different hardware configuration.

✓Studied behavior of software with different compilers, various compiler and linker options.

✓Approximately 50KB of data is transferred between Core, RAM and Traffic generator for every few ms and we got the expected response out of it.

✓Compared the results with the hardware implementation.

Thank you
for your attention!



Safe and Dynamic Driving
towards Vision Zero

SensePlanAct