System Overview

SC14CVMDECT

SC14CVMDECT System Overview

Abstract

This document contains an introduction to the software background of the SC14CVMDECT development. It is divided into: overview, host-based and target-based chapters.

Contents

Figures

Revision History

Version	Date	Description
1.0.0	25-Sep-2012	Initial version.
1.0.1	26-Oct-2012	Reviewed documentation with some small changes as a result
2.0	26-Feb-2013	Revised to new release (v2.0)
2.1	24-Jan-2022	Updated logo, disclaimer, copyright.

1. Terms and definitions

See SC14DECT_Terms and Definitions

2. References

1. SC14DECT_Terms and Definitions, Dialog Semiconductor
2. SC14CVMDECTDEVKT_Example Hosted Common Functionality, Dialog Semiconductor
3. API_PPCVMDECT_v0808.pdf, Dialog Semiconductor
4. API_FPCVMDECT_v0808.pdf, Dialog Semiconductor

3. Introduction

This is the complete overview of possible systems that can be made based on SC14CVMDECT, there are 2 methods to implement a system: hosted and on target (application in the module itself). The same cola application sources can be used in both cases (platform independent). This way development and debugging can be done on PC. Hosted use requires a separate bus mail library. This library provides the same API as used by internal cola application (on target).

4. Overview

The user should be aware that only one cola application can be active at a time (either in a host or in the module). In case of a hosted application the module cannot contain an active cola application (if there is a cola application present in the module it must be deactivated).

The Natalie stack has three ways to communicate:

- (bus) mail via UART (PC host or Embedded host)
- Internal mails. (on target)

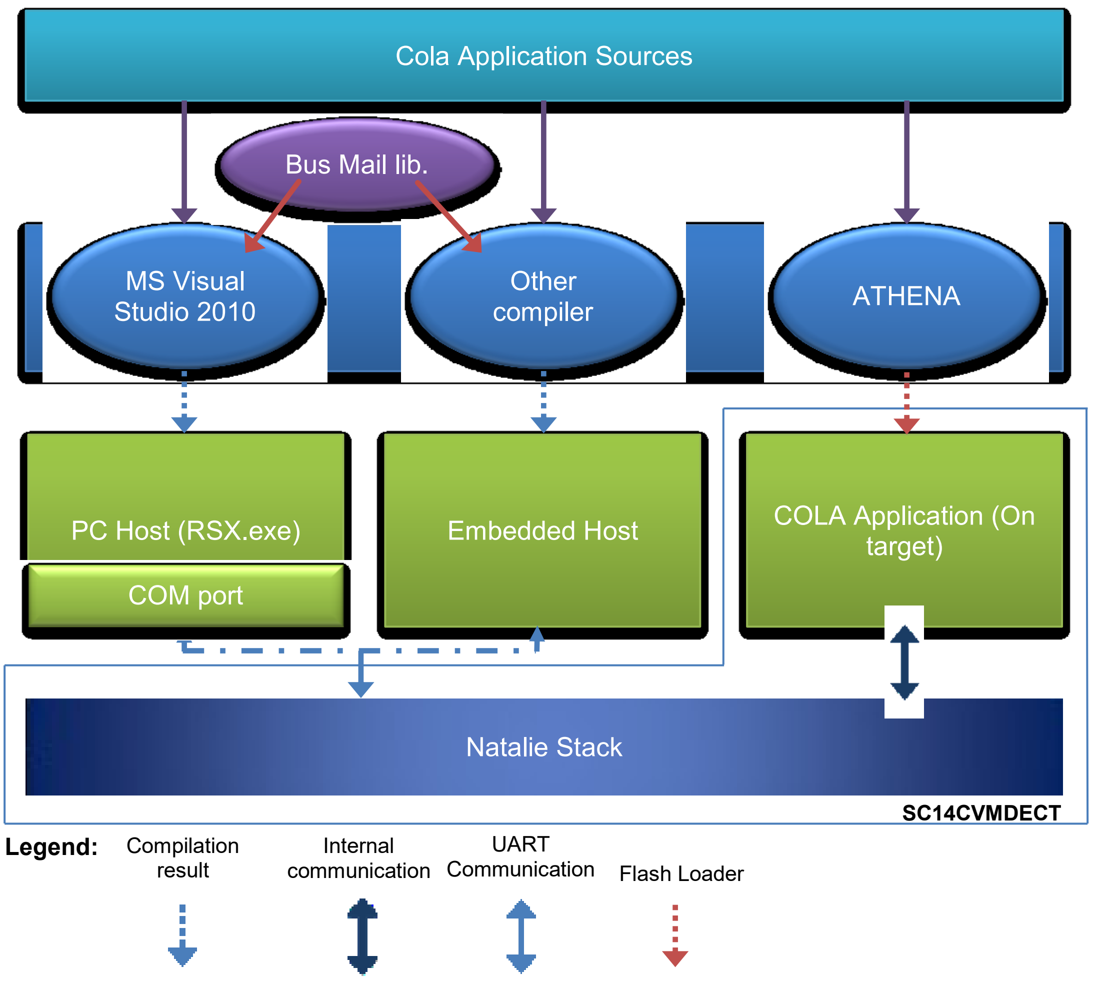

Figure 1 System overview

5. PC based development

PC based development can in principle be done with any compiler. The examples that come with the SC14DECTDEVKT are developed with MS Visual Studio 2010 express. MS Visual Studio 2010 compiles the written code into an executable. When executed it communicates with the module via a COM port on the PC. All provided host-code is portable, so any platform can be used as host as long as it has a UART port and an operating system. *(OS is required to support the bus mail library)

When the application is executed on a host the user must make sure that the SC14CVMDECT module contains only the Natalie stack and no active COLA application.

1.1 PC Host Application

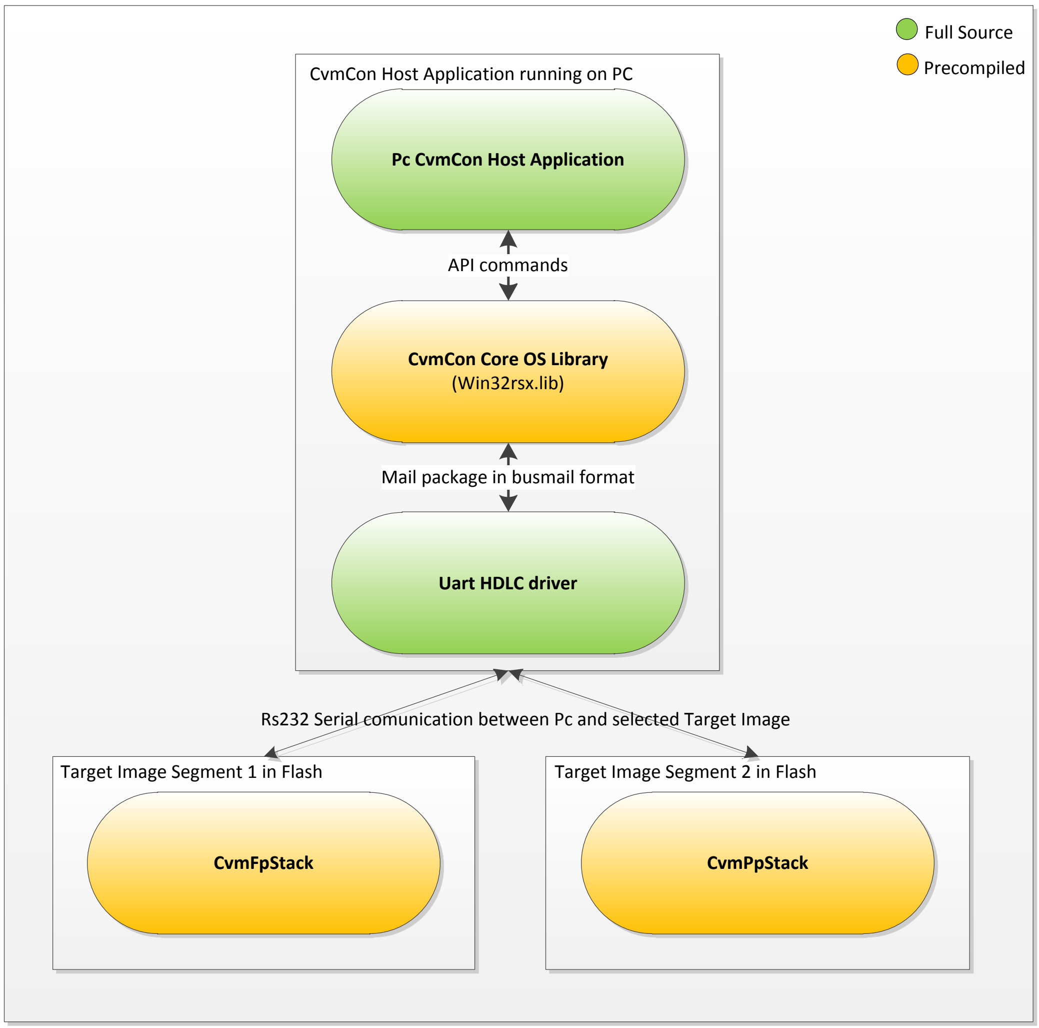
Full Source
Precompiled
CvmCon Host Application running on PC
Pc CvmCon Host Application
API commands
CvmCon Core OS Library
(Win32rsx.lib)
Mail package in busmail format
Uart HDLC driver
Rs232 Serial comunication between Pc and selected Target Image
Target Image Segment 1 in Flash
CvmFpStack
Target Image Segment 2 in Flash
CvmPpStack

Figure 2 Host Application

SC14CVMDECT System Overview

The Con Host applications are applications running externally on a PC, controlling the Stack through the serial port.

When set to External Host mode, the COLA image is disabled and all API communication from the stack is send to the UART instead.

The Con program consists of a user programmable Host application and a library with the API handles and Core functionalities. Built and linked together they result in a PC executable which can communicate to the Target through Bus mail on the serial UART.

For information the Con PC project also holds an example (Send mail Example) with full source code demonstrating how to implement the UART HDLC driver.

The demo projects are arranged in the sub-dirs. (example Con_FPConfDemo)

Cola Demo:

● Debug	The output of the build system: sln Visual C++ PC project file
● Lib	precompiled Con libs (Win32Rsx.lib, RsxAPILog.lib)
● Stack	Includes headers with various needed defines and types from the Stack

6. Third party host based development

6.1 Send mail Example

A Visual C++ PC project is included in the “Sendmail Example” with full source on how to implement a small application (sending one mail and receiving the response) this application can be used as a starting point for porting to another system.

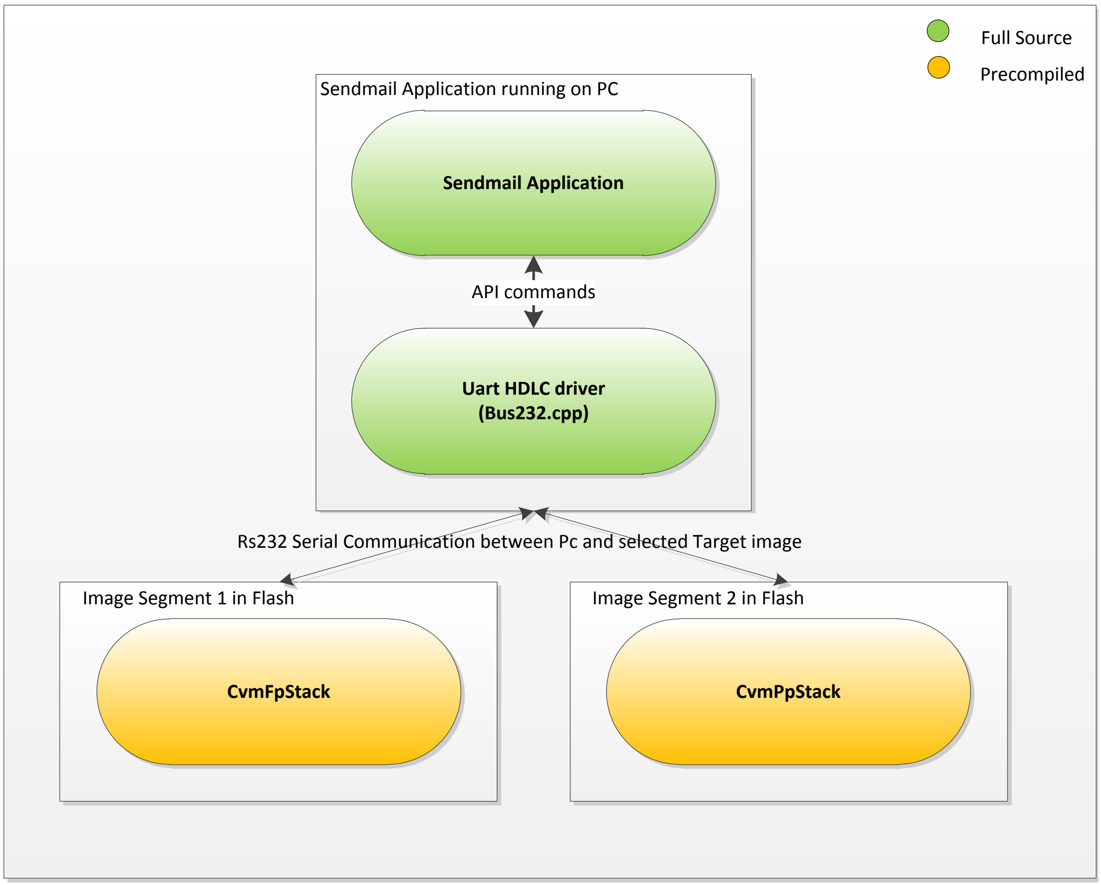
Full Source
Precompiled
Sendmail Application running on PC
Sendmail Application
API commands
Uart HDLC driver (Bus232.cpp)
Rs232 Serial Communication between Pc and selected Target image
Image Segment 1 in Flash
CvmFpStack
Image Segment 2 in Flash
CvmPpStack

Figure 3 Send mail example (Embedded)

It consists of:

• **Bus232.cpp**	The UART HDLC driver for sending and receiving API communication packages. Bus232.cpp automatically manages the HDLC protocol, packet numbering checking and retransmission.
• **Sendmail.cpp**	A small application example taking an API command from the command line and sending it to the Target. After transmission send mail will wait 1sec for reply and print any answer from the Target to screen.
• **Blinky.bat**	Calls the compiled sendmail.exe with a API_HAL_LED_REQ API command flashing a led on the Target.

This is an example of the blinky batch file calling sendmail.exe with API commands and parameters.

```
::=============================================================================
=
:: DESCRIPTION: Sends API_HAL commands to flash the Green LED
::
:: COMMAND LINE: Blinky.bat
::
:: PARAMETER VALUES: ComPort
::
:: RETURN FORMAT: None
::
:: TASK..... : 01       : API_TASK
:: PRIMITIVE : 5902     : API_HAL_LED_REQ
:: Parm[0]   : 02       : LedNr
:: Parm[1]   : 03       : CmdCount
:: Parm[2-4] : 01 1E 00 : First flash sequence  (ALI_LED_ON, 30ms)
:: Parm[5-7] : 00 1E 00 : Second flash sequence (ALI_LED_OFF, 30ms)
:: Parm[8-10]: 02 0A 00 : Third flash sequence  (ALI_REPEAT_SEQUENCE, 10ms)
::
::=============================================================================
=
@if [%1]==[] goto NoComPort

@debug\SENDMAIL %1 1 5902 02 03 01 1e 00 00 1e 00 02 0a 00

@Goto End
:NoComPort
@echo No COMPORT set!
@echo Please call blinky.bat with COMPORT as the command line parameter
@echo.
@Goto End
:NoParm
@echo Parameter missing!
:End
```

The Bus232 source is intended as reference example when porting the UART driver with HDLC to another host platform.

Note: all batch files (in \Examples\<release>\GenericTools\Settings\Bat) work in the same manner as the blinky.bat file.

For an explanation of the parameters see documentation reference: [3] [4]

7. Embedded development

The ATHENA software environment and a flash loader (FL7) are used for this scenario.

1.2 Cola Application

The COLA applications are SC14CVMDECT target applications controlling the Stack.
They are placed in a separate segment in the flash, so they can be programmed and flashed independently from the Stack code.
The Cola Application consists of a user programmable application and a library with the API handles and Core functionalities. Built and linked together they result in a hex file which can be loaded into target in Image segment, controlling the Fp or Pp stack.
For stack selection see section 0

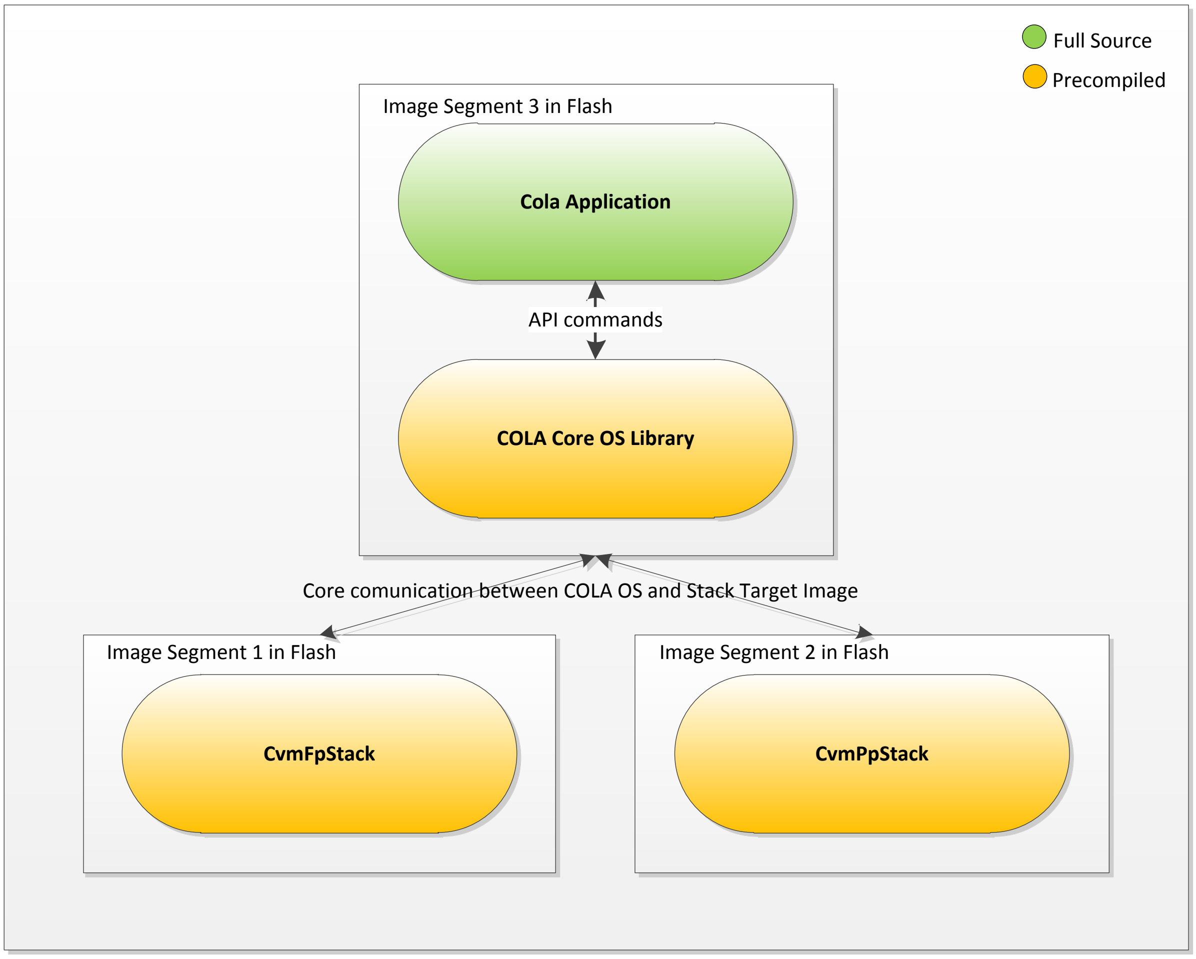

Figure 4 Target based development overview

SC14CVMDECT System Overview

In the Athena Environment there are several Cola demo projects.

The demo projects are arranged in the sub-dirs. (example: CvmCola_FPConfDemo)

Cola Example

- **Src** The actual Cola Application with full source code
- **Debug** The output of the build system:
  .spihex – the actual ColaApplication Hex file
  .map **–** map file with info on segment placement
  .lst – list files for the compilation of the individual source files
  .o – compiled object files for the individual source files

Shared files in main directory:

- **Stack** All stack related files
  - **Lib** The COLA Core OS library (libPhoenix.a)
  - **Tools** Various tools, scripts and definition files for the automated functions, use only if you want to manually executed the same steps

Note: Debugging is only possible in a hosted use scenario.

8. The COLA application functionality

8.1.1 The Cola Application Task

When the Stack has a mail that needs to be delivered to the COLA Task it will either:

Internal Cola Task:

Make a function call directly to the Cola Task. The call will hold the Mail Pointer pointing to the contents of the mail (Primitive and additional parameters)

External Host based Cola Task:

Pack the mail in the Bus Mail format and send it to the Host Processor.

In the Con PC examples the simulated OS/Core will receive the mail and make a function call to the Cola Task with the mail pointer (simulating the original Cola Task call).

8.1.2 Receiving mails in the Cola Task

When receiving a mail in the Cola Task, it receives a function call with the mail pointer. Depending of the Primitive in the mail, a mail type is defined to unpack each mail.

Example: sending a `API_FP_GET_FW_VERSION_REQ` request to the firmware from the stack. Then waiting to receive a `API_FP_GET_FW_VERSION_CFM call` back with the result.

```
void ColaTask(const RosMailType* MailPtr)
{
  switch (MailPtr->Primitive)
  {
    case KEY_MESSAGE:
      SendAPIFpGetFwVersionReq(COLA_TASK);
      break;

    case API_FP_GET_FW_VERSION_CFM:
      Printf("VersionHex=v%x\n", ((APIFpGetFwVersionCfmType*)MailPtr)-
>VersionHex);
      break;
  }
}
```

8.1.3 Mail Pack and Send

Sending mails from the Cola Task can be done in two ways.

Using the MPS functions:

To ease allocating and packing the mails, helping functions are made for each Mail Primitive.

From the example above the `SendAPIFpGetFwVersionReq` function is used to easily pack and send the `API_FP_GET_FW_VERSION_REQ`.

Source code for the MPS functions can be found in the Athena project in the “MPS Source Files” folder.

Manually packaging and sending

It’s also possible to pack and send the mails manually.
If doing so, it’s done with RosMailAllocate and RosMailDeliver. *(See “Mps Source Files”\ directory)*

Example from APIFpGeneralMPS.c

```
void SendAPIFpGetFwVersionReq ( RosTaskIdType Src )
{
  APIFpGetFwVersionReqType*
  m = (APIFpGetFwVersionReqType*)RosMailAllocate( Src,
APIFPGENERAL_TASK,(rsuint16)(sizeof(APIFpGetFwVersionReqType)));

  m->Primitive = API_FP_GET_FW_VERSION_REQ;
  RosMailDeliver((RosMailType *)m);
}
```

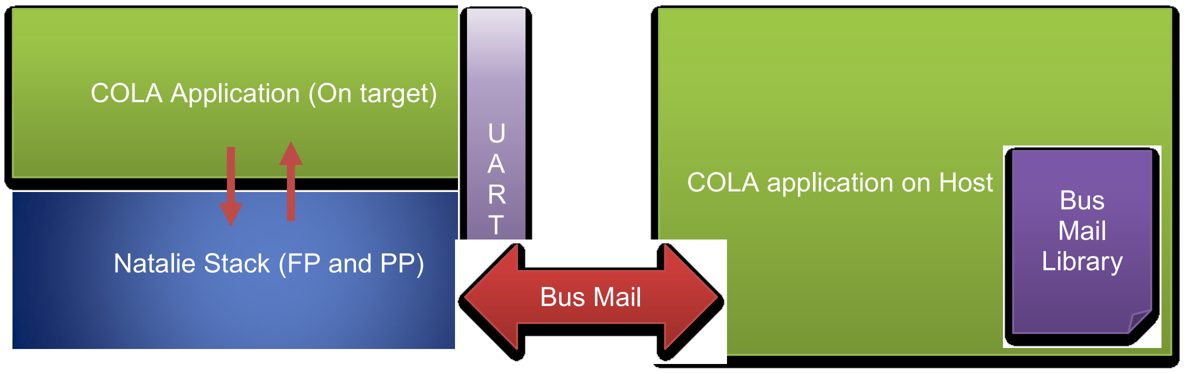

Figure 5 Mail Package system

8.1.4 Mail Timers

The OS/Core supports definitions of mail timers.
Several timers can be defined and set to timeout with individual timing.
At timeout a mail is send to the Cola Task with mail primitive `TIMEOUT` and the timer name as parameter.

Example: Defining and starting the `MMI_DEBUG_TIMER.`
Starting it at key press and waiting for timeout.

```
const RosTimerConfigType MmiDebugTimerConfig = { COLA_TASK, { TIMEOUT,
MMI_DEBUG_TIMER } };

void ColaTask(const RosMailType* MailPtr)
{
  switch (MailPtr->Primitive)
  {
    case KEY_MESSAGE:
      RosTimerStart(MMI_DEBUG_TIMER, RS_T1SEC, &MmiDebugTimerConfig);
      break;

    case TIMEOUT:
      switch (((RosMailType*)MailPtr)->Timeout.Parameter)
      {
        case MMI_DEBUG_TIMER:
          Printf("TIMEOUT\n”);
          //restart timer
          RosTimerStart(MMI_DEBUG_TIMER, RS_T1SEC, &MmiDebugTimerConfig);
          break;
      }
  }
}
```

The timers will only run and timeout once.
If a repeated timer is needed, the timer should be restarted by the Cola application.
A running timer can be stopped by the command `RosTimerStop(MMI_DEBUG_TIMER)` with the selected timer as parameter.

8.1.5 Keyboard and Print simulation

When the Cola Task is run on the PC a simple simulation of keyboard and display is possible.

Keyboard simulation
In the Con environment, when the "Application Console" is selected, any key press on the PC keyboard will be send to the Cola Task as a `KEY_MESSAGE` with the pressed key as the parameter.

In the target examples the integrated keyboard driver is aligned in key definitions also sending `KEY_MESSAGES` to the Cola Task.

Display simulation
As a simple display out from the Cola Task, Print can be used to write text strings to the "Application Console" window on the PC

8.1.6 Mail and memory allocation

In the OS/Core system on target and in the simulation on the PC there are heap files available. To operate it, the following commands are available:

RcHeapAllocEx – used to allocate a memory block
RcHeapReallocEx – used to extend the memory block
RcHeapFreeEx – used to free the memory block

9. Settings

The SC14DECTDEVKT provides a tool that is used to change EEPROM settings of the connected module. This tool is named: Con. For more information please refer to doc. Ref. [2]

Following settings can be changed using Con:

- Change FP/PP operation of the stack
- (De-)Activate COLA task in the module
- DECT mode

The same settings can be changed using the generic tools mentioned in section 10.

ImageCfg

The Cola Task is the single application task placed outside of the Stack.
It can be placed in either the Cola segment of the internal Flash or running on an external processor (host) communicating through the UART.
Internal or external operation is selected in EEPROM “ImageCfg” (EEPROM address: 0x00AB).
Bit 7 of “ImageCfg” determines whether COLA is on or off (value 1 or 0).
Bit 0 of ImageCfg” determines whether the FP or the PP stack is active.

10. Description of the Generic Tools

When using the Athena development environment the user will learn to use the generic tools supplied with the development kit (see Quick Start Guide for a description of how to use the generic tools).

This set of tools has been created to automate certain flash loading and EEPROM-setting related actions. Each of the tools is named after its purpose, for example the **Program_FPConfDemo** starts an interactive procedure to program the FP conferencing application into the flash memory of the selected unit

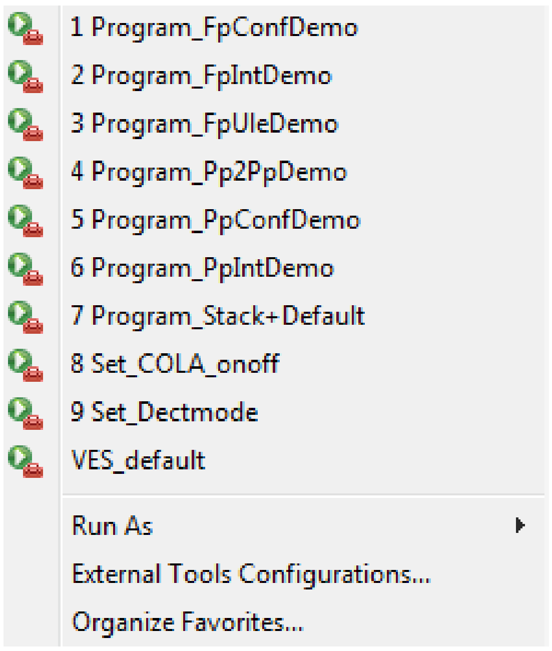
1 Program_FpConfDemo
2 Program_FpIntDemo
3 Program_FpUleDemo
4 Program_Pp2PpDemo
5 Program_PpConfDemo
6 Program_PpIntDemo
7 Program_Stack+Default
8 Set_COLA_onoff
9 Set_Dectmode
VES_default
Run As
External Tools Configurations...
Organize Favorites...

Figure 6 External Tools window in Athena

The Program Launchers of **Error! Reference source not found.** each result in the calling of a batch file that can be found in directory: CvmDectExamples\GenericTools\Tools\.

These batch files can also be called from the command line, so it is not absolutely required to use their functionality from Athena only.

Here is a brief explanation of the included launchers. There are three kinds of launchers:

- Launchers that result in programming a hex file into the module (these start with "**Program_**"). Programming a hex file into the module takes some time.
- Launchers that result in re-configuring the module (change in VES).
- Combination of the above ("**Program_FpPp_Default**")

Program_FPConfDemo

This launcher will automatically program the Fp conference example application onto the module

(note: the stack must be set as FP)

Program_PPConfDemo

This launcher will automatically program the Pp conference example application onto the module

(note: the stack must be set as PP)

Program_FpIntDemo

This launcher will automatically program the Fp intercom example application onto the module

(note: the stack must be set as FP)

Program_PpIntDemo

This launcher will automatically program the Pp intercom example application onto the module *(note: the stack must be set as PP)*

Program_FpULeDemo

This launcher will automatically program the Fp ULE example application onto the module *(note: the stack must be set as FP)*

Program_Pp2PpDemo

This launcher will automatically program the Pp2Pp example application onto the module *(note: the stack must be set as PP)*

Set_COLA_onoff

This launcher will let the user switch between active and disabled COLA

VES_Default

This launcher will let the user select between Fp or Pp operation mode, choose the DECT mode, enable/disable COLA. After the Fp/Pp selection a default will be forced *(note: this is not factory default, it is the same as pressing the PON button)*

Set_DECT mode

This launcher will let the user select between DECT modes: either 0 = EU, 1 = US, A= Japan, B = Japan 5 channel.

Program_stack+default

Same as **VES_Default** but starting with erasing any stack and COLA image in the flash memory and then program the stack again.

Status Definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

RoHS Compliance

Dialog Semiconductor's suppliers certify that its products are in compliance with the requirements of Directive 2011/65/EU of the European Parliament on the restriction of the use of certain hazardous substances in electrical and electronic equipment. RoHS certificates from our suppliers are available on request.