AR_Software_7SR5.docx

Product / Function: **7SR5_Platform**

Author

Name	Department	Version	Date
Neil Dosdale	EM DG SPDL R&D	V1.00	2017-11-20
Neil Dosdale	EM DG SPDL R&D	V1.03	2018-03-01
Neil Dosdale	SI DG SA&P PR GB-R&D 2	V1.04	2019-07-04
Steven Idle	SI EA R&D MP GB 1	V1.05	2022-11-14
Keith Stenson	SI EA R&D MP GB 2	V1.06	2024-04-10

Updates

Chapters / Pages changed	Version	Object and reason of change / Reference to change requirements
-	V1.00	Renamed and setup to the main AR / design reference manual for the firmware on the RN platform.
	V1.03	Added the A9 and KON file programming information. Updated information for debugging and connecting using the SEGGER J-Link.
	V1.04	+ Converted from original project specific file into a generic 7SR5 software platform document. + Added the firmware signing process. + Added the M4 memory map. + Updated J-Link tool version to 6.34h + Added Deliverables, File Formats and Loading Mechanisms. + Added MAC addresses and Firmware Signing.
Table 6 1	V1.05	+ Updated Table 6 1 Slave Card Article Numbers
All	V1.06	+ Updated for V2.50 release. + Updated to latest template

Contents

Figures

Tables

1 Preface

The current 7SR1 platform comprises two architectures:

- E4 with TC1130 or NXP LPC4437 CPU
- E6 with TC1130 CPU, with optional NXP LPC4337 slave processor for protection comms

The TC1130 CPU is obsolete and the NXP is unsuitable for native IEC 61850, due to limited internal Flash memory, and no cache for external memory.

The 7SR2 platform uses a TC1796 processor (formerly a TC1775), and there are no suitable upgrades which have Ethernet capability. Despite a DTC exercise, the 7SR2 platform is still more expensive than the 7SR1 for similar functionality.

It is therefore proposed that a re-design of the platform is required to address these issues. At the same time a DTC exercise will be carried out to reduce the HK cost of IEC61850 versions by at least 15%.

1.1 General notes

1.2 Business case

1.3 Terms and definitions

Term, acronym	Description
7SR5	Reyrolle Native IEC 61850
RC	Reyrolle Compact
RDL	Relay Description Language
RDO	Relay Description Object
RDS	Relay Description Language Script
HWT	Hardware test code
HAL	Hardware Abstraction Layer
HMI	Human Machine Interface
SCL	Substation Configuration Language (IEC 61850)
ICD	IED Capability Description (IEC 61850)
XSL	XML Stylesheet Language
XSLT	XSL Transformations v1.0
RSF	Relay/Reyrolle Setting File/Format

Table 1- Terms, acronyms and definitions

2 Requirements and architectural drivers

2.1 Functional requirements

List the relevant requirements for the architecture from the requirements specification.
Update the list of requirements if necessary when new or changed requirements turn up during the lifetime of the product.

2.2 Non functional requirements (NFR)

Provide a ranked list the most meaningful NFR’s as compliance, efficiency, extensibility, interoperability, maintainability, performance, portability, reliability, resource constraints, safety, security, scalability, testability.
Please refer to (http://en.wikipedia.org/wiki/Non-functional_requirement)

2.3 Further architectural drivers

List further issues which influencing the architecture such as standards, patents, re-used components, skills, time line, budget, multi-side effects.

3 Architecture description

The current design architecture used in the RC family of protection devices with IEC61850 requires a two-processor system. The main processor (Tricore TC1130) handles all functionality except the IEC61850 protocol stack. This includes the protection functionality, the HMI, non IEC61850 comms, and all the low-level drivers for accessing the binary inputs, outputs, keys, and LEDs. To provide IEC61850 comms an EN100(+) module is used and the interface between the TC1130 and the EN100 is through dual-port RAM provided by the EN100(+) through a shared 8K interface.

To minimize risk and delay the new software architecture will follow this same principal. However, instead of using two microcontrollers and communicating through a dual-port RAM interface the system will run on one microcontroller (the NXP iMX6 SX) and utilize both the M4 core and the A9 core of the device. The M4 core will provide the same functionality as the TC1130 processor in the RC platform and the A9 core will provide the same functionality as the EN100(+).

To ensure minimal changes for porting the RC platform the dual-port RAM off the EN100 will be in an area of internal RAM that the two cores can communicate through in the same way as the EN100. In effect, this makes the A9 core look and behave like an EN100 in respect to its memory interface.

The TC1130 port of the RC will not be used in this case. A cost reduced version of the RC platform uses the NXP LPC4337 microcontroller that uses an M4 core and the port for the iMX6 SX will use this port – hopefully the libraries can just be used directly although some investigation will be required. If this is not possible then the libraries will need recompiling for the iMX6 SX.

The IEC61850 stack used by the EN100(+) has already been ported and is used on an A9 core. Similarly, with the M4 port it is hoped that minimal work will be required for the A9 port although the endianness of the port may require some careful consideration.

Access to other cards is performed by the M4 core and uses SPI interfaces to implement a proprietary protocol allowing cards to be controlled and information shared between the main controller and all the slave cards.

The basic architecture for the 7SRn is show in Figure 1.

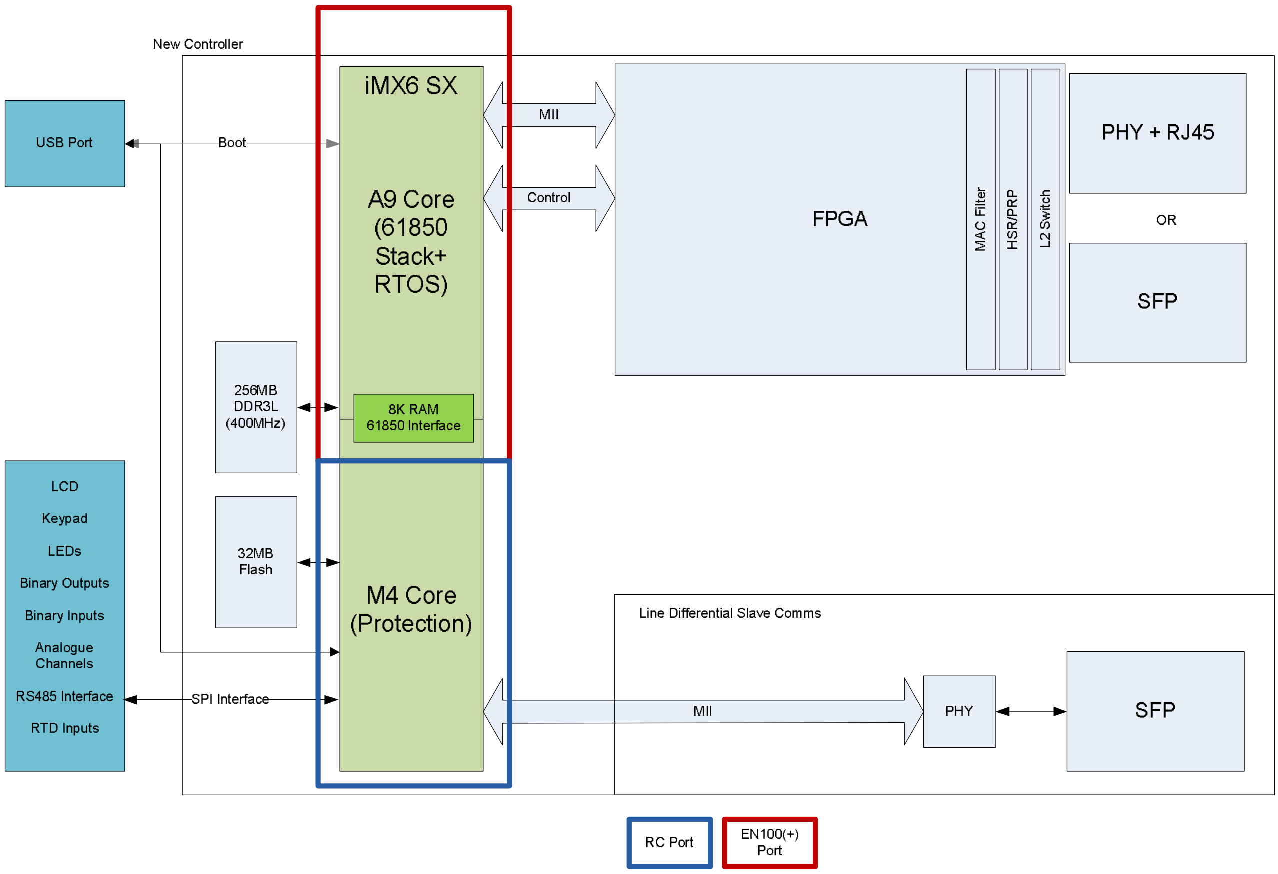


Figure 1 – Basic Architecture

3.1 A9 Architecture

Upon start up the iMX6 SX boots from the A9 core. It must initialize itself and any peripherals it is going to use, including all memories. Once initialized the A9 core will use the USB port to determine if new code is being sent to the device and if so, will then program the code received. Finally, the A9 core will determine if the main application firmware needs to be started or if HWT must be executed and then start the relevant program on the M4 core.

Once the M4 core has started the A9 core will then proceed to start either its own application code or enter HWT mode ready for the M4.

Figure 2 shows the A9 components. The basic architecture is same with EN100. Most modification causes by RTOS which changes from PSOS to Vxworks.

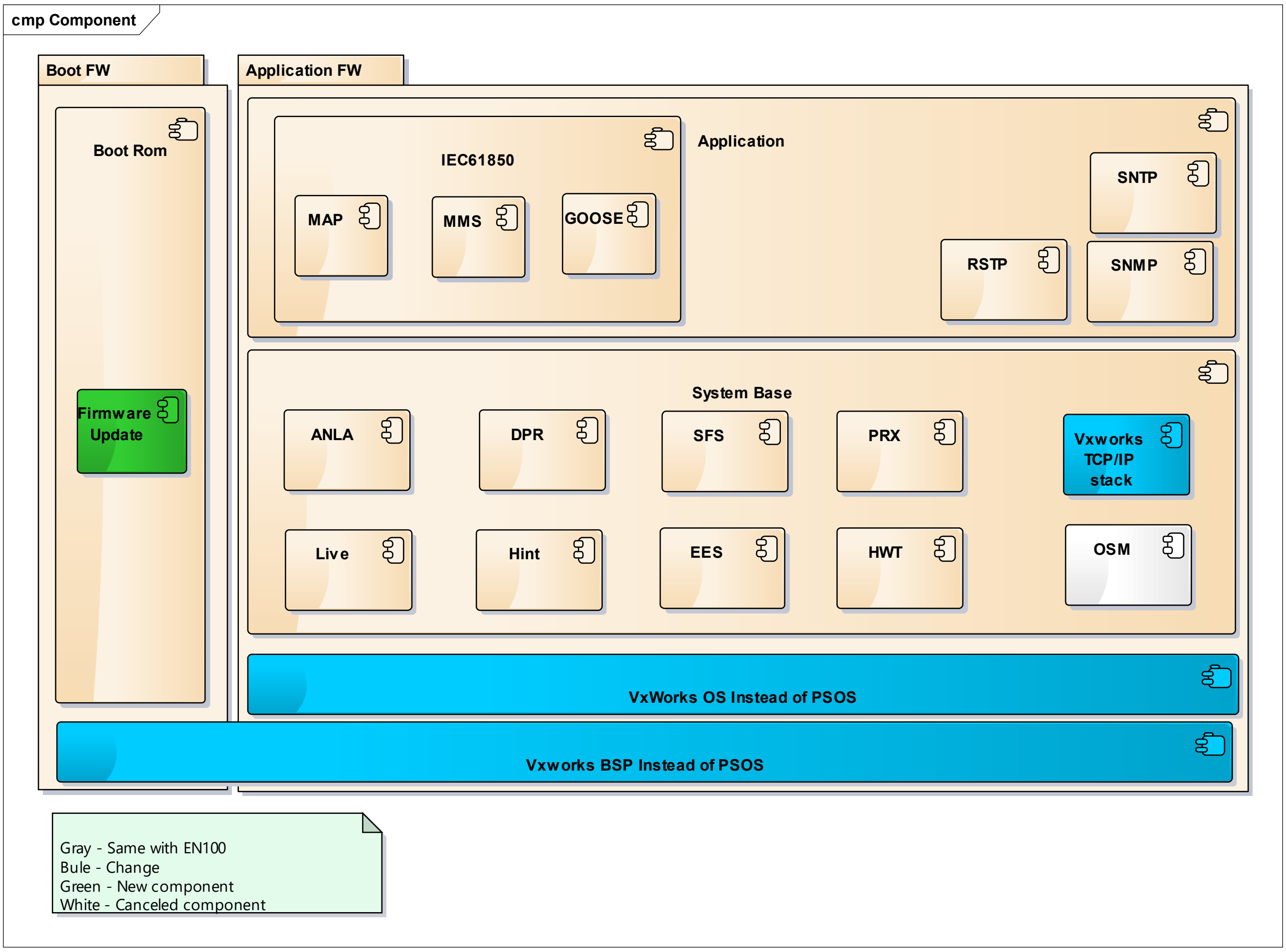


Figure 2 A9 Architecture

3.1.1 Boot-Rom Component

The Boot-Rom based on Vxworks BSP. Main features include:

- Firmware update
- Upload image via network while some key has been pushed.
- Jump to HWT mode while connecting with monitor tool (this function should be further discussion)

3.1.2 System-Base Component

The component defines as below table:

Component Name	Description	Compare with EN100 and Smart-Device
LIVE	Monitor other task and kick off watchdog, task priority is highest	Same with EN100
HINT	Background supervision, task priority is lowest	Same with EN100
ANLA	Start up	Adjusting based on EN100

DPR	Information exchange with M4 core, including three tasks: DPRI, DPRS, DPRR	Adjusting based on EN100
HWT	Hardware-Test	Same with EN100
PRX	Include three tasks: ▪ PRX1 HTTP-Proxi for Stack in device ▪ PRX2 RPC-Proxi for Stack in device ▪ PRX3 DIGSI-Proxi for Stack in device PRX2/3 will be canceled because they are not used for 7SR5.	Adjusting based on EN100
SFS	Siemens Files System	Same with EN100
EES	FPGA Diagnosis & Control	Adjusting based on EN100

Table 3-1 Module list of System-Base

3.1.3 Application Component

Application component includes a variety of protocol, such as SNMP, RSTP, IEC-61850 and so on.

Component Name	Description	Compare with EN100 and Smart-Device
SNTP	Simple Network Time Protocol	Adjusting based on Smart-Device
RSTP	Rapid Spanning Tree Protocol	Adjusting based on EN100
SNMP	Simple Network Management Protocol	Adjusting based on Smart-Device
IEC61850	IEC61850 protocol, including MMS and GOOSE	Adjusting based on Smart-Device

Table 3-2 Module list of Application

3.1.4 Boot-Rom Initial

The device will start up after while power on. Basic workflow of Boot-Rom defines as below.

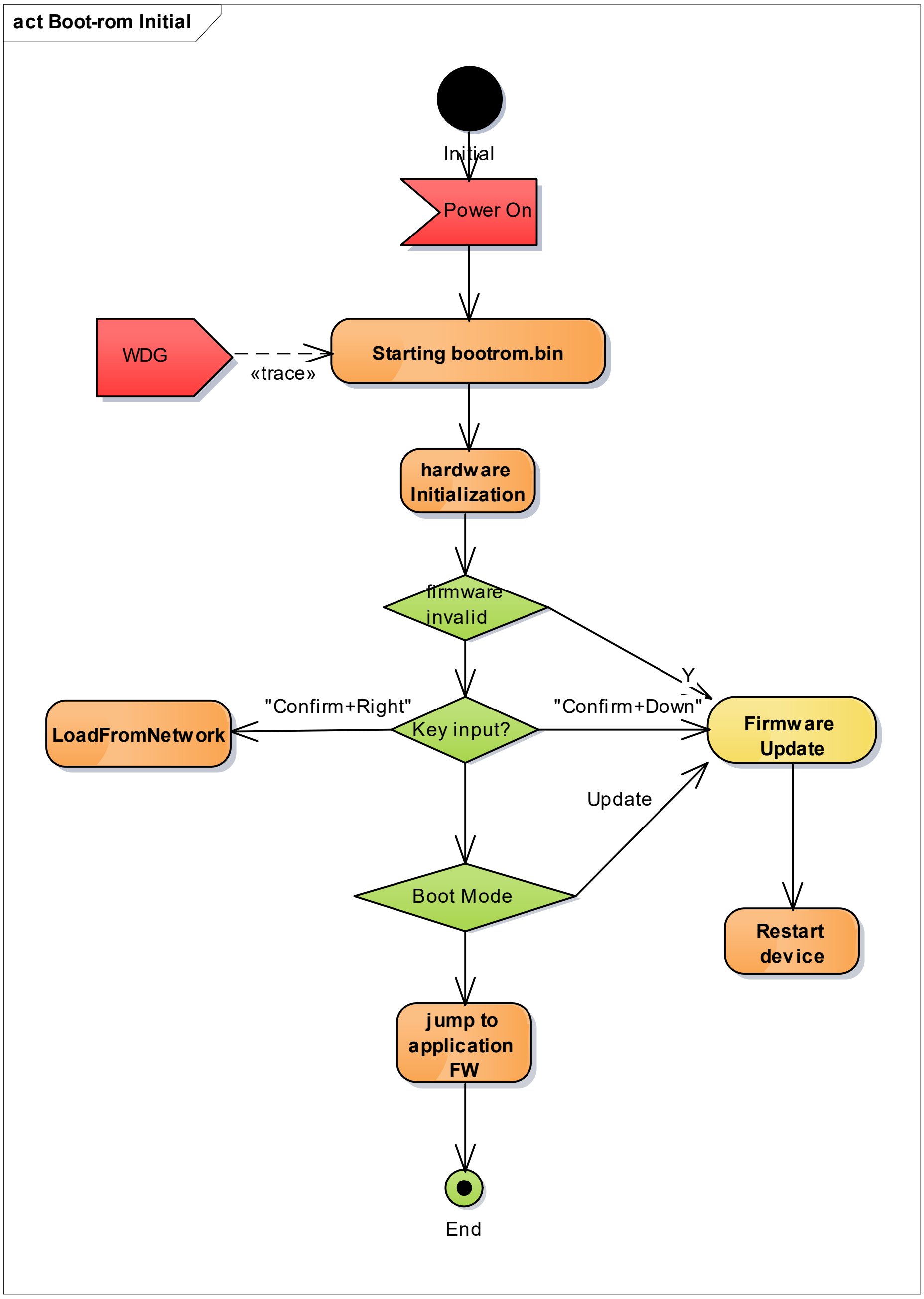


Figure 3 Boot-Rom initial.

3.1.5 Application Initial

The basic workflow of application defines as below.

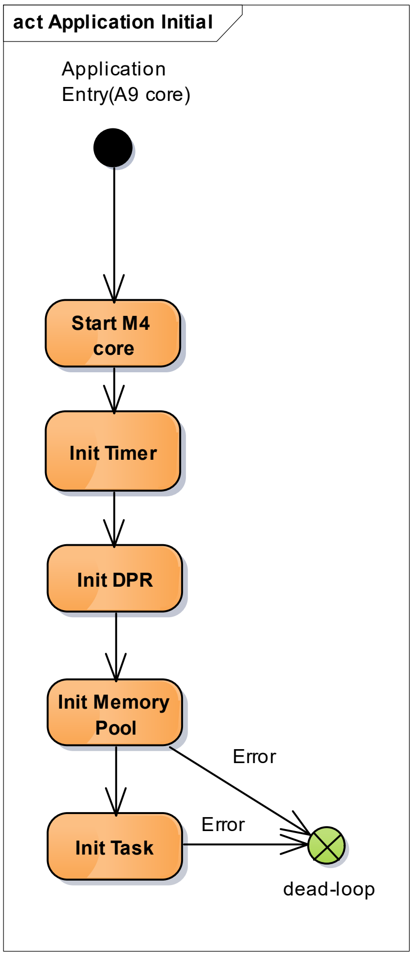


Figure 4 Application Initialize

3.1.6 A9 - Firmware Update process

The A9 firmware update process is described in Section 5.2, page 33.

3.2 M4 Firmware Architecture

The same design, construction, and mode of operation currently used in the RC products will be adhered to. This is shown in Figure 5. Both the firmware and the HWT are loaded into the product during the production phase and execution of either can be selected. It must be noted however, that only one can run at a time; either application firmware which is the default mode of operation, or HWT which is the test mode of operation. If no application firmware is present, only HWT, then HWT will always be started. Neither the firmware nor HWT depends on the other existing. To jump from one to the other a restart of the device is required.

To start the M4 core the A9 core must first boot up and decide which program to run – HWT or the application firmware.

It will be the responsibility of the M4 core to initialize all the peripherals it requires – i.e. GPIO ports, SPI, UARTS etc. Also, after starting it must gain control of the USB from the A9 core so that the front port can be used during normal operation. Once started the M4 core will run independently from the A9 core.

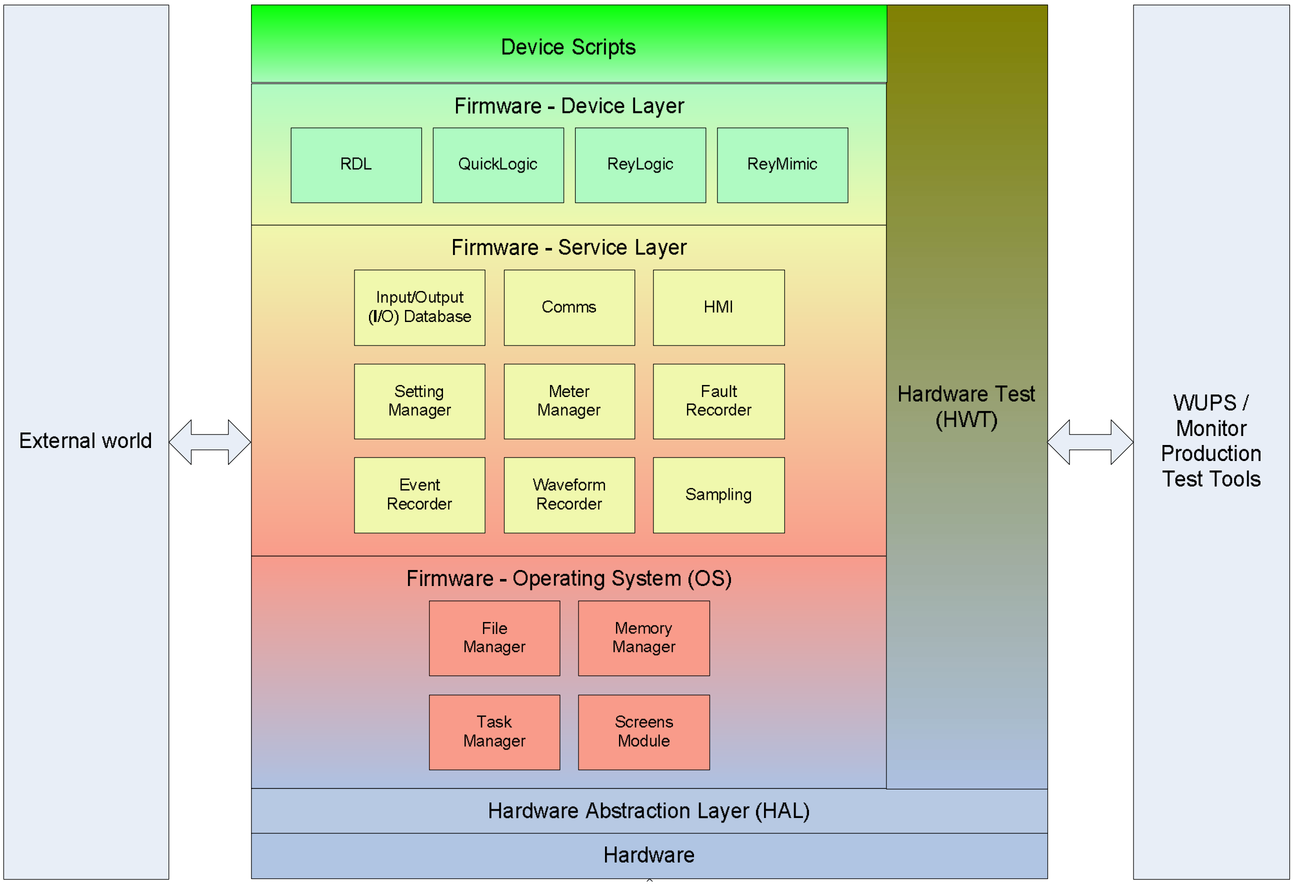


Figure 5 – M4 Architecture

The RC reusable architecture has two main components.

- Binary application
- RDL script files

Following subsections describe these components.

3.2.1 Binary Application

The binary is the application or executable code that provides all the generic features for the device; to turn the device into a product such as the 7SR5x. A set of device configuration files need to be installed into the device.

The binary has been divided into several layers, each containing logical blocks that make it easier to understand and to simplify integration and testing.

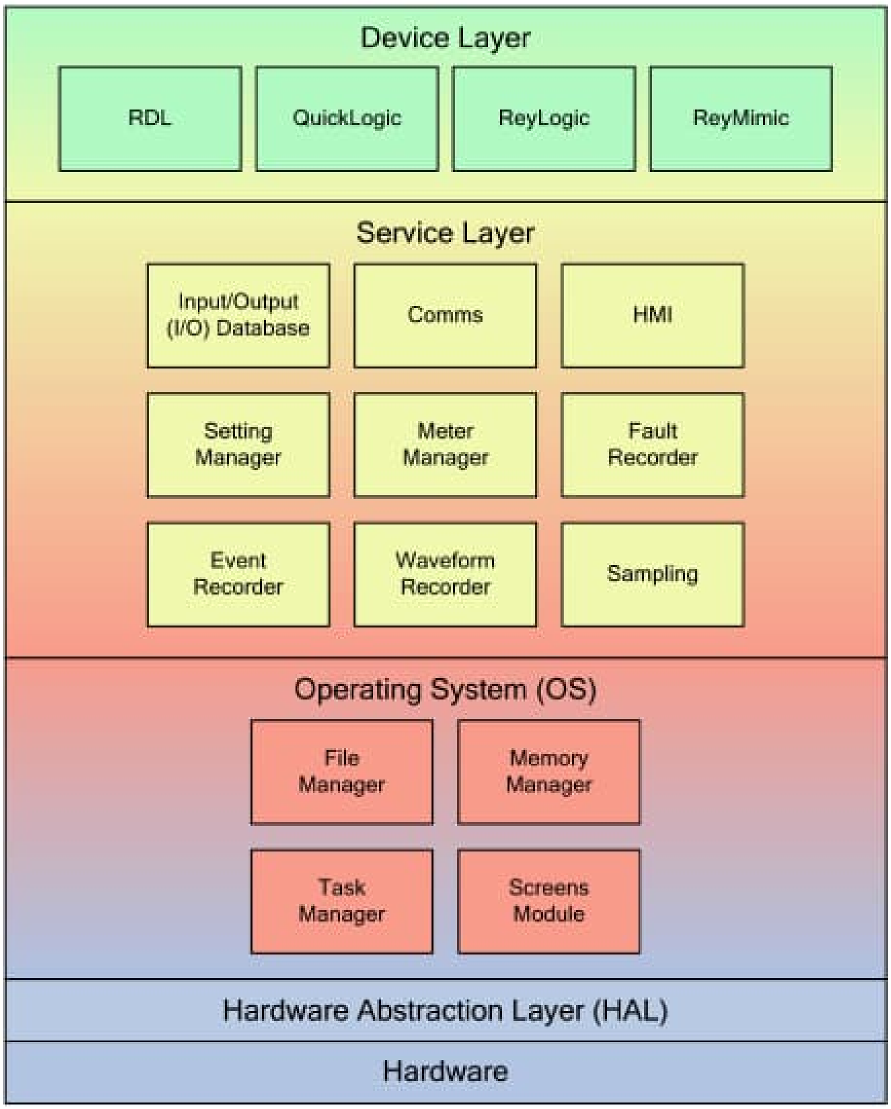


Figure 6 – Logical structure of M4 binary.

- The HAL provides a basic interface onto the actual hardware.
- The Operating System (OS) is responsible for the management and coordination of activities and the sharing of the resources within the device. This includes file manager, memory manager, task manager and screens module.
- The Service Layer provides the generic frameworks that are common across all products on the RC platform. This includes sampling, HMI, communications, settings, instruments, faults etc.
- The Device Layer provides the frameworks that allow customization for the final products. These include the RDL engine, QuickLogic, ReyLogic and ReyMimic.

3.2.2 RDL script files

RDL scripts are part of device layer. Most of the protection and control functions required in relays are created as block elements. These elements are written in C code which is platform/controller independent. Then these elements are provided as graphical blocks to a tool named as "Reylogic toolbox". Through this software logical block diagrams are created with possible parameterization that is set through device settings. These scripts are added on top of the basic Binary application to turn device into a product such as a 7SR5x.

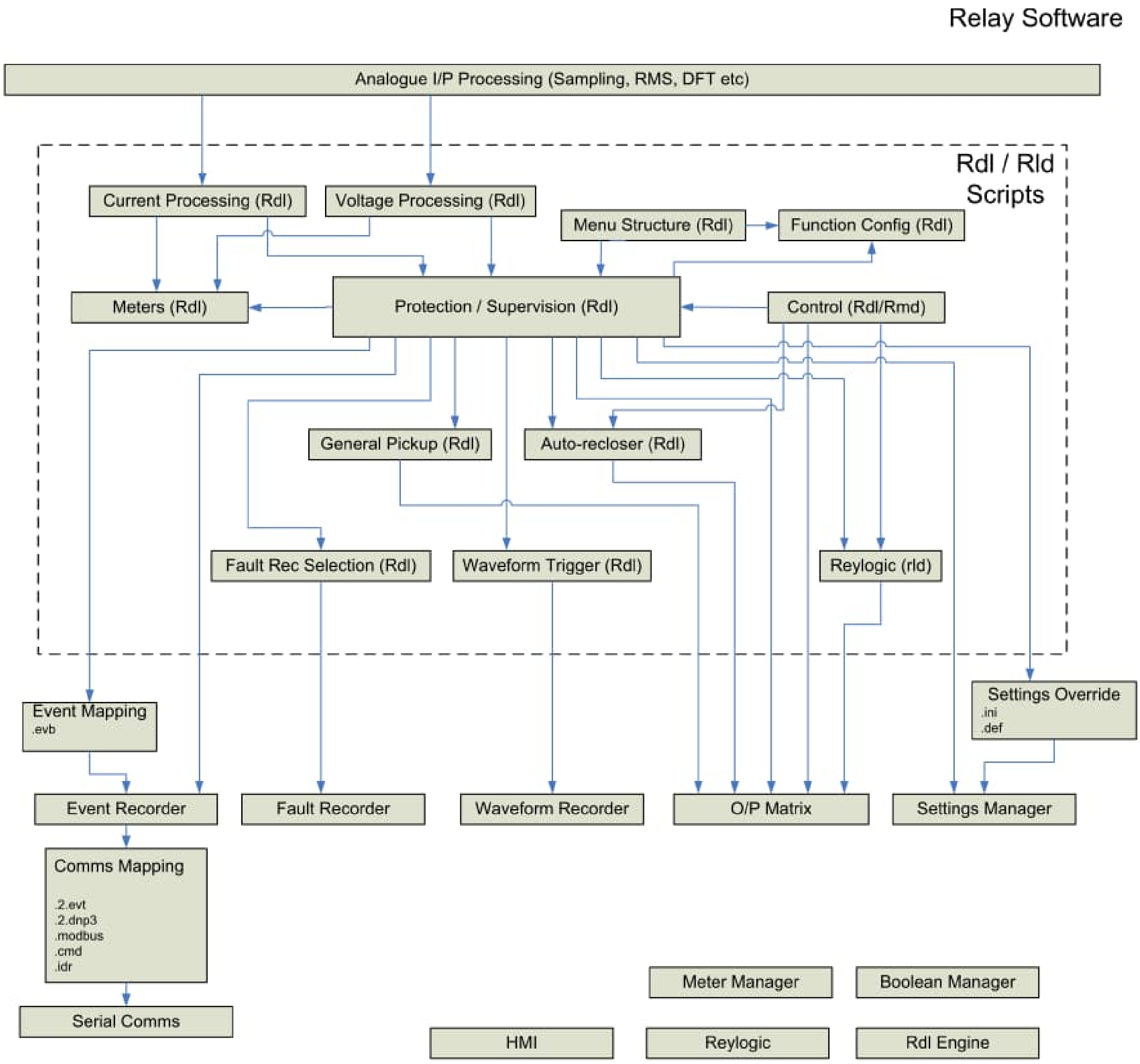


Figure 7 – Logical structure and interconnections of RDL scripts.

3.2.3 SPI Interface

Interaction between the iMX6 SX and the hardware is done via an SPI interface using a proprietary protocol of which a detailed explanation can be found in the Architecture document for the hardware. The protocol uses a register-based access method to enable control of the attached cards. Each card must publish its own register set. Figure 8 shows an example definition of a register memory layout.

Flash programming of the cards can be done in-situ and will be controlled by the M4 core.

FASCIA REGISTER MEMORY

Control Mode Access Addresses

H = 0	H = 1		Size	Access	Description	Default Value	
		Header Info					
-	0	SpiInfo	4	RO	Maximum baud rate supported	12500000	
-	4		4	RO	tD_ns - Command to data delay	1100	
-	8		4	RO	tDC_ns - Min data to checksum delay	1500	
-	12		4	RO	tC_ns -Char delay before checksum	100	
-	16		4	RO	tF_ns - Time between frames	2000	
-	20	BoardInfo	2	RO	Board ID	**See Note 1**	Depends on board detected
-	22		32	RO	Board Name	**See Note 1**	Depends on board detected
-	54		2	RO	Number of data registers	162	
-	56		2	RO	Number Of Operate Registers	7	
	58		1	RO	Register Memory Version	0	Depends on the board implementation
-	59		1	RO	Operate mode Master To Slave Enabled	TRUE	
-	60		1	RO	Operate Mode Slave To Master Enabled	TRUE	
-	61		20	RO	Board Article Number	**See Note 1**	Programmed by MF
-	81		20	RO	MF Number	-	Programmed by MF
-	101		20	RO	Software Article Number	-	Programmed by Software Build
-		Operating Registers					
-							
		Operating Mode - MISO data	4	RO	KeyPad Mask	-	Slave to Master Data
			3	RO	Not Used	-	
		Operating mode - MOSI data	1	WO	Control Flags - Self resettable by slave	-	**See Note 2**
			2	WO	Video Memory Address / LED Bank	-	Master To Slave Data
			4	WO	Video Memory Data / LED Data	-	
		Data register	-				
0			2	WO	Video Memory Address / LED Bank	-	
2			1	WO	Bytes To Write	-	
3			1	WO	Control Flags - Self resettable by slave	-	**See Note 2**
4			64	WO	Video Memory Data / LED Data	-	
68			1	R/W	LCD Contrast - Percentage	50	Sets the LCD contrast in %age
69			4	R/W	LCD Timeout ms	3600000	Sets the timeout of the LCD
73			4	R/W	Backlight Timeout ms	300000	Sets the timeout of the backlight
77			1	R/W	LCD Refresh Period ms	20	Sets the update rate of the LCD
78			1	RO	Number Of Keys	**See Note 1**	Defines the number of keys present
79			1	RO	Number Of LEDs	**See Note 1**	Defines the number of LEDs present
80			2	RO	LCD X size	**See Note 1**	Defines the size of the LCD (x-axis)
82			2	RO	LCD Y Size	**See Note 1**	Defines the size of the LCD (y-axis)
84			1	RO	LCD Colour Depth	**See Note 1**	Defines the number of colours - **See Note 3**
85			1	RO	LCD Is Graphical	**See Note 1**	TRUE if graphical LCD, FALSE for characters
86			4	RO	Keypad mask	-	Mask of the current keys pressed
90			4	RO	5V Supply	-	Returns the 5V monitored supply rail
94			4	RO	3V3 Supply	-	Returns the 3V3 monitored supply rail
98			64	RO	Error Buffer	-	Error message buffer
162							

Notes:

1

Board ID	Board Name	Board Article Number	Number Of Keys	Number Of LEDs	LCD X size	LCD Y Size	LCD Colour Depth	LCD Is Graphical
0	E6,128x128LCD,7K,28L	C53207-A431-B21-x	28	7	128	128	4	TRUE
1	E12,128x128LCD,7K,28L	C53207-A431-B11-x	28	7	128	128	4	TRUE

2 **Control Flag: Self resettable by the slave device**

Bit 0	Update Video Memory - When set the data transmitted is for the video memory.
Bit 1	Update Green Leds - When set the data transmitted is for the green Leds.
Bit 2	Update Red Leds - When set the data transmitted is for the red Leds.
Bit 3	Reinitialise LCD
Bit 4	LED Test
Bit 5-7	Not Used

3 **Defines the number of bits used for each pixel.**

1	B&W
2	4 Colour greyscale
4	16 Colour greyscale
8	256 Colour
16	65536 Colour
24	True Colour

Figure 8 – Fascia Register Memory

3.2.4 A9 core Interface

As mentioned previously the A9 core will be used to implement an equivalent of the EN100(+) module so that native IEC61850 can be provided. To minimize changes within the M4 port the interface to the A9 core will be through an equivalent version of the EN100(+) dual-port RAM implementation. This dual-port RAM interface will be in an internal area of RAM of the iMX6 SX device. It's format and implementation will be the same as the EN100(+) interface except for how the inter-processor interrupts are generated and cleared. Instead of read/writing to the interrupt cells of the dual-port RAM (last 2 locations of memory) the cores will communicate via their internal inter-processor communication system, i.e. interrupts.

3.2.5 M4 Flash - Memory Map

Error! Not a valid link.

Figure 9 – M4 Flash - Memory Map

3.2.6 Third party components

Following table provides information on third party components used on the M4 core.

Table 3-3: Third party components

Component	Description
File manager	Segger System emFile Flash Memory filing system is incorporated
Communication Manager	Triangle Microworks DNP 3.0 Protocol Stack is incorporated
Operating System	FreeRTOS
Graphics Driver	Segger System emWin graphics library. This will be the free version provided by NXP/Segger.
BZIP	Open-source zip program.

3.3 7SR5 Device Architecture

3.4 Reydisp Manager Template Generation

7SR1 and 7SR2 Reydisp Manager Templates are generated using a combination of user definitions and device interrogation. Device interrogation involves locating suitable hardware, loading the target product installer, downloading the runtime configuration (settings, setting values, Booleans, etc.). Due to the number of hardware variants for a product, this process is time consuming.

7SR5 Reydisp Manager Template generation will remove this time-consuming process by making the device state deterministic by adding additional definitions in the Reylogic Toolbox project for a product.

3.4.1 Template Input Format

3.4.2 Template Output Format

3.4.3 RDS/RDO Processing

7SR1 and 7SR2 template generation relies upon 61850 sections being defined in RDO. This section contains an XML-like representation of the IEC 61850 SCL tree for the RDO. The SCL trees for each RDO defined in the product RDS are collected and used as input into the ICD generation process.

This process will be modified to serve the purposes of 7SR5 template generation with these overall goals:

- Use standard and established XML formats and technologies.
- To increase the expressive power of the RDO data.
- To allow the RDO to define not only SCL for the template process, but also relay settings, user logic, events, etc.
- To allow ad-hoc and dynamic platform descriptions of SCL, relay settings, user logic, events, etc to be generated by Reydisp Manager 2.
- To include validation mechanisms that the RDO data is defined correctly before use.

As a result, the *61850* section of an RDO will contain more than just 61850 definitions. This section should be renamed to correctly reflect the intended contents. However, keeping the *61850* section name requires less modification of existing libraries.

3.4.3.1 RDO XSL

The *61850* section of RDO files will be modified to contain only valid XML. A subset of XSL 1.0 can be used within the XML. The XML tree will be extracted, automatically converted to an XSL stylesheet, and evaluated as an XSL transformation, the output of which will be validated against an XML Schema (XSD).

```
RDL OBJECT DEFINITION

  OBJECT NAME FOO
  OBJECT DESCRIPTION Just an example RDO.

  OVERVIEW
    A demo RDO.

  61850
  <RDO xmlns="http://tempuri.org/RdsDataSchema.xsd"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <Settings>
      <Menu
        Id="{$Name}"
       Menu="FOO PROTECTION"
        Description="{$Name}" />

      <SimpleSetting
          Id="FOO.{$OBJNAME}.Setting"
          Description="{$Name} Setting"
          Menu="{$Name}"
          Default="{$DefaultLevel}"
          Units="{$Units}"
          Common="false">

        <DisplayRule>
          <MinStepNext><xsl:value-of select="$SettingRange" /></MinStepNext>
       </DisplayRule>
      </SimpleSetting>

    </Settings>
```

```
  </RDO>

  PROPERTIES
    STRING "Name"          DEFAULT "Foo Name" HELP "The name."
    STRING "Units"         DEFAULT "s"        HELP "Units for foo setting."
    STRING "Setting Range" DEFAULT "0,1,20"   HELP "Range for foo setting."
    FLOAT  "Default Level" DEFAULT "2"        HELP "Default value for foo setting."

  INPUTS

  OUTPUTS

  RULES

END.
```

Example RDO including XML *61850* section. The example shows menu and setting definitions, the use of RDO property variables using XSL *attribute value templates*, the use of XSL variables, and the use of *xsl:value-of.*

```
FOO MY_FOO, Name = “My Foo”, Units = “ms”, Setting Range = “5,5,1000”, Default Level = 500
```

Example definition of above RDO in an RDS file. Each property has been changed from its default value.

For each RDO declared in an RDS, the following process happens:

1. Locate the correct RDO class for the declaration.
2. Instantiate the RDO class.
   a. Create a representation of an RDO object based on the class.
   b. Set all RDS properties for the RDO object.
   c. Apply all RDO rules from the *RULES* section of the RDO. This may change the value of some properties.
3. Extract RDO XML
   a. Fetch the XML from the RDO.
   b. Convert the XML into an XSL stylesheet (see example below).
   c. Execute the XSL transformation.
4. Validate the XML output with an XML schema.

```
<stylesheet version="1.0" xmlns:RDO="Reyrolle:RDO" xmlns="http://www.w3.org/1999/XSL/Transform">
  <variable name="OBJNAME" select="RDO:Name()" />
  <variable name="OBJTYPE" select="RDO:Type() " />
  <variable name="Name" select="RDO:Property('Name')" />
  <variable name="Units" select="RDO:Property('Units')" />
  <variable name="SettingRange" select="RDO:Property('Setting Range')" />
  <variable name="DefaultLevel" select="RDO:Property('Default Level')" />

  <template match="/">
    <RDO xmlns="http://tempuri.org/RdsDataSchema.xsd"
         xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
      <Settings>
        <Menu
          Id="{$Name}"
        Menu="FOO PROTECTION"
          Description="{$Name}" />

        <SimpleSetting
            Id="FOO.{$OBJNAME}.Setting"
            Description="{$Name} Setting"
            Menu="{$Name}"
            Default="{$DefaultLevel}"
            Units="{$Units}"
            Common="false">

          <DisplayRule>
         <MinStepNext><xsl:value-of select="$SettingRange" /></MinStepNext>
        </DisplayRule>
        </SimpleSetting>

      </Settings>
    </RDO>
  </template>
</stylesheet>
```

Example intermediate output when converting RDS XML to an XSL stylesheet. Things to note:

- `xmlns:RDO="Reyrolle:RDO"` An extension object is made available via the namespace prefix *RDO*. This gives access to the RDO object being evaluated, such as:
    - `RDO:Property('Property Name')` evaluates to the set value of the property called '*Property Name*'.
    - `RDO:Input('Input Name')` evaluates to the name of the RDO object connected on the input called '*Input Name*'. A special input name '*$*' can be used to refer to the main input which shares the name of the RDO object.
- An XSL variable will be generated for each RDO property to allow property value lookups, e.g. `RDO:Property('Default Level')`, to be shortened to, e.g. `$DefaultLevel`. A conversion process to turn RDO property names into valid XSL variable names:
    - Any spaces are removed.
    - Replace any character that is not alphanumeric, hyphen (-), period (.), or underscore (_), with an underscore (_).
    - If the name doesn't start with a letter of an underscore (_), prefix the name with an underscore (_).
- Two special variables are created:
    - `$OBJNAME` is the name of the object in the RDS.
    - `$OBJTYPE` is the class name of the object.
- The *RDO* element of the RDO XML is inserted verbatim into an xsl:template. Thus, any XSL operation that is valid inside al xsl:template, is also valid within the RDO xml, such as:
    - xsl:if, xsl:choose, xsl:when, xsl:value-of, xsl:variable, etc

This stylesheet can then be evaluated to produce the desired output tree for the RDO object being evaluated.

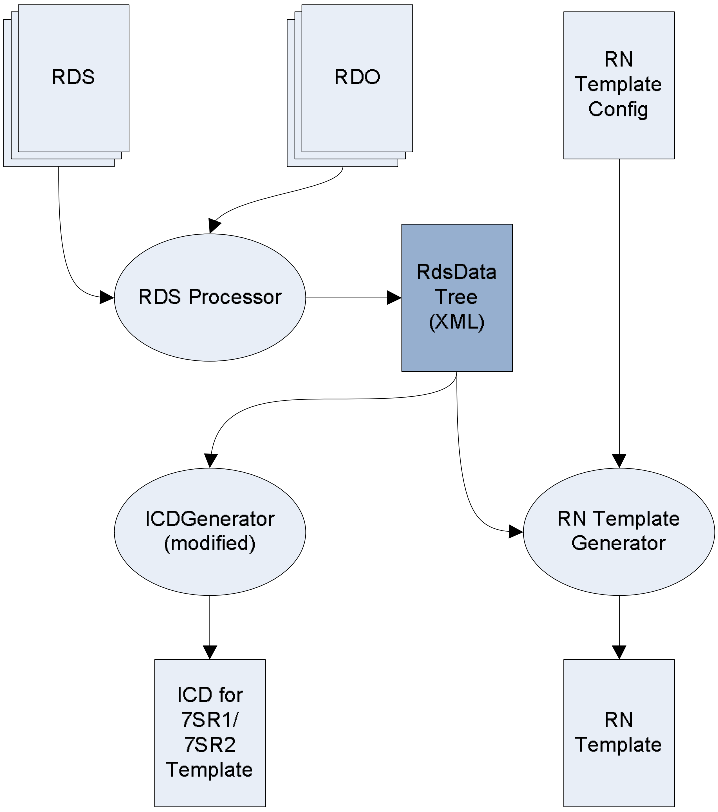
When the above RDO and RDS declaration are evaluated, the following output is expected:

```
<RDO xmlns="http://tempuri.org/RdsDataSchema.xsd"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <Settings>
    <Menu
      Id="My Foo"
      Menu="FOO PROTECTION"
      Description="My Foo" />

    <SimpleSetting
        Id="FOO.MY_FOO.Setting"
        Description="My Foo Setting"
        Menu="My Foo"
        Default="500"
        Units="ms"
        Common="false">

      <DisplayRule>
        <MinStepNext>5,5,1000</MinStepNext>
      </DisplayRule>
    </SimpleSetting>

  </Settings>
</RDO>
```

Example of RDO evaluation when applied to the RDS declaration:

FOO MY_FOO, **Name** = *"My Foo"*, **Units** = *"ms"*, **Setting Range** = *"5,5,1000"*, **Default Level** = *500*

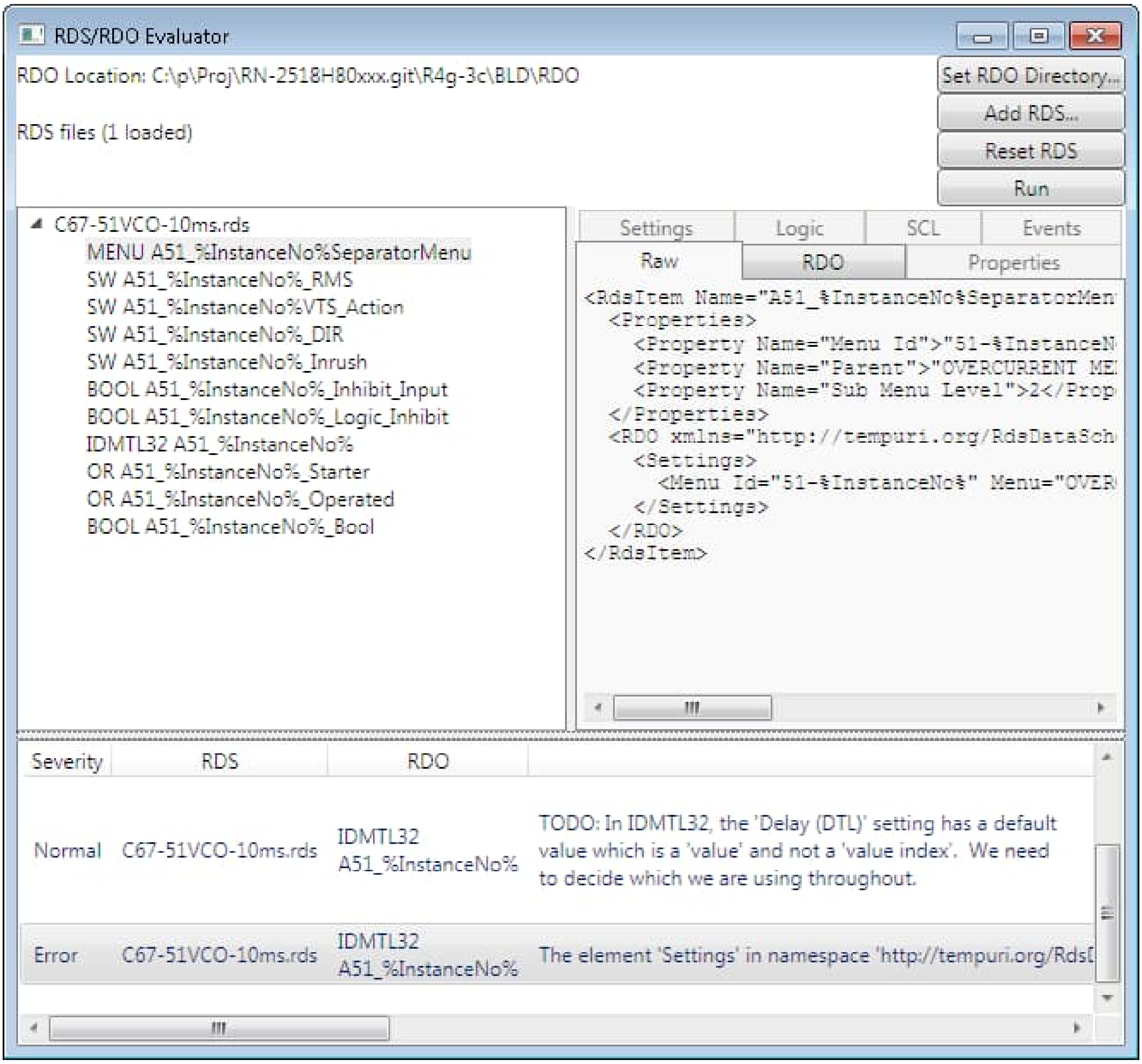
3.4.3.2 RDS Processor Output



A new RDS Processor will be created to extract RDS/RDO data (RdsData). The existing ICDGenerator will be modified to support this data as input to the IXD/ICD generation process. The RdsData will also be used by a new 7SR5 Template Generator tool to generate 7SR5 Templates.

XML Schema for RdsData (TODO: Insert link here).

3.4.3.2.1 Development tool RDS/RDO Validation



3.4.4 Dynamic SCL

4 General Architectural Topics

4.1 Release Classification & Version Planning

The release classification and numbering will adhere to the guidelines laid out in the Siemens official Release Classification document here:
I:\General-Data-D\Bln\Q_Docu\PLM_Lean\Process\04_Guide-CM_Release_Classification\PLM_TU_ReleaseClassification_EN.pdf

The FW and Devices will have the same version number to simplify things for developers and customers.

- Prototypes up until type test/product release will be V1.xx.xx
- First type test candidate will be V2.00.00
- First released product will be V2.00.03 after defect corrections.

In this way it is not seen as a first version product which it should not be as it will have been tested many-many times.

The RDM version number will remain separate at V2.xx.xx upon final release.

With the FW/Devices/RDM all at V2.00 it would give us the potential to keep everything in step – V2.x RDM and V2.x 7SR5 are fully compatible. And increment them where required.

4.2 File Formats – CMS/VOL/KON/PCK

This section gives a brief description of each file format and what they are used for.

4.2.1 File Formats – CMS

- The CMS is the output of the digital signing process.
- This will be the result of signing a firmware or a device configuration.
- This is an intermediate file and will not be uploaded to the device.

4.2.2 File Formats – VOL

- User configurations will use a VOL output file format.
- VOL files are not digitally signed, as this would require the customer to have access to digitally sign software and configuration packages.
- VOL files can only be uploaded to the device via the HTTPS loading mechanism.

4.2.3 File Formats – KON

- The KON file will be the output file format for firmware and device configurations.
- Once the file has been digitally signed as a CMS, then it will be converted to a KON which can be uploaded to the device.
- KON files can be loaded via the DTLS loader or the HTTPS loading mechanism.

4.2.4 File Formats – PCK

- The PCK is used to package multiple KON files into a single deliverable file.

- PCK files can be uploaded via the DTLS loader or the HTTPS loading mechanism.

4.3 Deliverable Packages

This section describes the various packages that will be produced during the 7SR5 platform lifecycle.

4.3.1 Deliverable Packages – Protection Firmware

- The protection firmware refers to the M4 firmware and the configuration scripts.
- Generally, it will only include the M4 firmware and configuration scripts, but where logical it can include other files such as HWT and the BootRom.
- Configuration scripts are system configurations that are uploaded into the SYSTEM area of the file system on the device.
- These are RDL scripts and configuration files that define protection elements and functionality within the device.
- The protection firmware package contains digitally signed files and can be loaded via DTLS or HTTPS.

$/PRODUCTS/518_RN/7SR5_Platform_2518H80022/

4.3.2 Deliverable Packages – Communications Firmware

- The communications firmware refers to the A9 firmware.
- This firmware is currently produced by Nanjing.
- It provides a Virtual EN100 interface for the device, and the software is based on the EN100 code.
- Generally, the Communications package will include the A9 firmware, but it can also include the BootRom, ComWebUI, and FPGA images, should they need to be upgraded.
- In the future, it is planned that this will include regular security updates.
- The communications firmware package contains digitally signed files and can be loaded via DTLS or HTTPS.

$/PRODUCTS/518_RN/7SR5_CommsFirmware_2518H80024/

4.3.3 Deliverable Packages – User Configurations

- User configurations are produced using Reydisp Manager.
- They allow the user to commission a device and setup the protection elements and functionality in a device.
- A user configuration package is not digitally signed, so it can only be uploaded using HTTPS (as it provides a secure communications protocol).

4.3.4 Deliverable Packages – Production

- There are three HEX files that are required for production.
- The hex images are flash images for the two external flash parts on the main CPU board.

Name	Description
7SR5_M4HwtPcba_FlashImgM4.hex	*M4 flash part image*
7SR5_M4HwtPcba_FlashImgA9_Electrical.hex	*A9 Electrical flash part image*
7SR5_M4HwtPcba_FlashImgA9_Optical.hex	*A9 Optical flash part image*

A9FW	0_Sign_A9FW.bat
M4FW	0_Sign_M4FW.bat
M4HWT	0_Sign_M4HWT.bat
Device Scripts	0_Sign_M4SYS.bat

Table 4-1 Manufacturing hex file deliverables.

4.3.4.1 Building intermediate PCK images

- The hardware test PCK files are used to create the hex images deliverables.
- Simply run “1_Build_7SR5_M4HwtPcba.bat” to create the “Electrical.pck” and “Optical.pck” files.
- All the files required can be found in Vault:

$/PRODUCTS/518_RN/7SR5_M4HwtPcba_2518H80023/1_build_pck/

4.3.4.2 Building production deliverable HEX images

- The Prerequisite hardware to build the HEX images is as follows: - processor cards (electrical and optical), PSU and fascia.
- Using additional cards will create additional files to be stored in the HEX images, such as calibration files from slave cards.
- The files required to build the hex image can all be found in Vault:

$/PRODUCTS/518_RN/7SR5_M4HwtPcba_2518H80023/2_build_hex/

1. **Prerequisite:** Update CMD files to ensure the correct Segger file path is used, based on the version of software installed on the local PC – for example: JLink_V614h
2. Attach debug connector and the front USB cable.
3. Follow the instructions found in the file 0_ReadMeInstructions.txt in the Vault directory above to create the A9 image.
4. **Repeat 3 the process for both optical and electrical images.**
5. The image for the M4 is generated automatically from the DevOps build process.

4.4 Device Upload – Firmware and configurations

4.4.1 DTLS loader – Recovery Mechanism (FMU)

- This method uses the BootRom to load packages into the device, using the SIPROTEC Firmware Upload program (FMU).
- The DTLS loader/FMU mechanism will only be used for recovery purposes, when the relay is returned to an authorized SIEMENS repair center – this may be the original manufacturing factory or a regional office – FMU with DTLS support is an internal tool and we do not wish to distribute it to customers.

4.4.1.1 Installing packages with FMU

- The device must be in boot mode to upload packages via the FMU.
- To enter boot mode: press and hold the DOWN, 1 and 0 keys simultaneously and power the device on.
- The device will now be held in boot mode and will not attempt to startup.
- Only digitally signed firmware and configurations scripts can be loaded using FMU.
- PCK containing signed KON files and individual signed KON files can be uploaded with FMU.

- VOL files and unsigned KON files **cannot** be uploaded using FMU.

4.4.2 HTTPS loader – Device Healthy upload mechanism

- While the device is running the HTTPS loader is active and packages can be uploaded to the device.
- The HTTPS loader will upload digitally signed firmware, configurations scripts and user configurations.
- PCK packages containing signed KON files can be uploaded via HTTPS.
- User configurations (user.vol) can be uploaded via HTTPS.
- Only PCK and user VOL files are recognized by the HTTPS loader – KON files are **not** accepted.
- It is possible for the HTTPS to accept unsigned packages (PCK packages that contain unsigned KON files). However, this will only be used in development and will be disabled on all customer devices.

4.5 Firmware Signing & Verification

- The firmware signing is a security feature so that only signed firmware or configuration scripts will be accepted by the device.
- Firmware signing is required for all binary firmware packages: -
  BootRom,
  FPGA,
  A9FW,
  M4FW and
  M4HWT.
- In addition, system configuration scripts must also be signed.
- User configurations are not signed.

4.5.1 Firmware Signing – Overview

- A PC with a special setup is used to perform the signing.
- The current PC is named MD3HHVVC, and all the tools are set up.
- The signing of the firmware is performed through the DevOps Release pipelines **7SR5 Build RDM2 Template & System**, **7SR5 Build M4FW,** and **7SR5-Comms-FW**.

4.5.2 Firmware Signing – File Formats

- The signing process uses a binary file as input and produces a signed binary file. The signed binary can then be used as normal.
- Firmware will be in the BIN format and output a CMS signed binary format file. This file is then converted to a KON file, which can then be uploaded to the device.
- Device configuration scripts use a raw uncompressed VOL format and output a CMS signed volume format file. This file is then converted to a KON file, which can then be sent to the device.

4.5.3 Firmware Signing – Signing Process

1. For a release the signing is performed through the DecOps release pipelines **7SR5 Build RDM2 Template & System**, **7SR5 Build M4FW,** and **7SR5-Comms-FW**.

Name	Vault Location	File to run
BootRom	$/PLATFORMS/518/FIRMWARE/BootRom_2518S80611/SIGN/	*Not signed in Hebburn*
FPGA-E	$/PLATFORMS/518/FIRMWARE/FPGA_Electrical_2518S80612/SIGN/	*Not signed in Hebburn*
FPGA-O	$/PLATFORMS/518/FIRMWARE/FPGA_Optical_2518S80613/SIGN/	*Not signed in Hebburn*
A9FW	$/PLATFORMS/518/FIRMWARE/A9FW_2518S80610/SIGN/	0_Sign_A9FW.bat

M4FW	$/PLATFORMS/518/FIRMWARE/Mod518_V1_2518S80031/SIGN/	0_Sign_M4FW.bat
M4HWT	$/PLATFORMS/518/FIRMWARE/HWT_2518S80471/SIGN/	0_Sign_M4HWT.bat
Device Scripts	$/PRODUCTS/518_RN/RN 2518H80xxx/SIGN/	0_Sign_M4SYS.bat

Table 4-2 Firmware Signing locations.

4.5.4 Firmware Signing – Disable Firmware Signing (For Development Only)

- The HTTPS firmware signing checks can be disabled on startup, and shall only be used on development devices within R&D.
- The DTLS (recovery mode) firmware signing check **cannot** be disabled.

4.5.4.1 Firmware Signing – Enable unsigned mode.

- To enable the unsigned firmware mode, the following procedure is required:

1. The device must be started in hard debug mode. (Hard debug mode uses the file stored in secure, while the soft debug mode is access via the boot menu)
2. Enter the password command over the ASCII comms interface “PW”.
3. Enter the debug command “SIGNSIGNSIGN”. This will return “Done” if successful.
4. An INF entry will be added to the FACTORY and FACTORY_BACKUP areas on the file system.
5. Restart the device.
6. Unsigned firmware will not be accepted – the device must remain in debug mode (hard or soft) for the to be disabled.

4.5.4.2 Firmware Signing – Disable unsigned mode.

- To remove the unsigned mode and enable the firmware signing checks, follow the same procedure and call the “UNSIGN” debug command, instead of the “SIGNSIGNSIGN” command.

4.6 MAC Addresses

- The MAC address is usually programmed into the main processor card during the manufacturing process.

4.6.1 MAC Addresses – Creating a MAC in development.

- A MAC image can also be created and programmed in development.
- The files to create a MAC address can be found in Vault:

$/PLATFORMS/518/CONFIG/MacCreate_2975H80049/Release/

- The MAC address image will need to be digitally signed, so the signing server needs to be used.
- The MAC address image can be loaded by HTTPS or the DTLS loaders.
- The following process can be used to program a MAC address during development:

1. Log on to the signing PC - MD1RG6MC (you will need to have access rights to the signing server).
2. Get latest files from Vault folder.
3. Edit 0_CreateAndSignMacBinary.bat to include the appropriate MAC, serial number, and article number.
4. Run 0_CreateAndSignMacBinary.bat.
5. Store the output PCK – by default this is mac_cms.pck.

4.6.2 MAC Addresses – HBB R&D Assigning a MAC address.

- HBB R&D have been allocated one hundred MAC addresses from GOA MF database.
- These are predefined MAC addresses and the package to install each one has already been created and digitally signed.
- They are stored in Vault.

$/PLATFORMS/518/CONFIG/MacCreate_2975H80049/Stored/OfficialMacs/

4.6.2.1 Allocating an unused MAC address

$/PLATFORMS/518/CONFIG/MacCreate_2975H80049/Stored/OfficialMacs/AllocateMacAddress.xls

- Update the spreadsheet to allocate the next available MAC address to the device.

MAC	Allocated Date	CPU Serial Number	Device ID	Device MLFB	Owner
00-E0-A8-FC-EC-A8	27/06/2019	none allocated	GF1906501273	7SR5111-1AA21-0AA0/BB	Ham Lab
00-E0-A8-FC-EC-A9	27/06/2019	none allocated	GF1906501261	7SR5110-1AA21-0AA0/BB	Ham Lab
00-E0-A8-FC-EC-AA	27/06/2019	none allocated	GF1906501275	7SR5111-1AA21-0AA0/BB	Ham Lab
00-E0-A8-FC-EC-AB	27/06/2019	none allocated	GF1906501269	7SR5111-1AA11-0AA0/BB	Ham Lab
00-E0-A8-FC-EC-AC	27/06/2019	none allocated	GF1906501271	7SR5111-1AA11-0AA0/BB	Ham Lab

- Then rename the allocated MAC in Vault to include the serial number of the device.

Missing	MAC_00E0A8FCECB6_cms_GF1906501266.pck
Missing	MAC_00E0A8FCECB7_cms_GF1906501267.pck
Missing	MAC_00E0A8FCECB8_cms_GF1906501276.pck
Missing	MAC_00E0A8FCECB9_cms_GF1906501263.pck
Missing	MAC_00E0A8FCECBA_cms_GF1906501262.pck
Missing	MAC_00E0A8FCECBB_cms.pck
Missing	MAC_00E0A8FCECBC_cms.pck
Missing	MAC_00E0A8FCECBD_cms.pck
Missing	MAC_00E0A8FCECBE_cms.pck

- Load the MAC using either the HTTPS loader or DTLS loader.
- When the device restarts is should display “New MAC detected” on the first startup screen.

4.7 Watchdog Mechanism

- Both A9 and M4 provide their own watchdogs which must be serviced by the relative core. If either watchdog is not serviced correctly then the whole device will reset, and a watchdog event will be registered in the restart log.

4.8 Exception handing and error processing

- The M4 will process its exceptions independently from the A9. The M4 errors will be recorded by the SadRelay or UserError function and recorded in the restart log.
- The A9 will record its exceptions which can be accessed by the homepage functionality just like the EN100.

4.9 Endian Format

- EN100's Endian format is big-endian, but the IMX6SX only support little-endian.

5 A9 Firmware Architectural Topics

5.1 A9FW – CMP

To follow…

5.2 A9FW – Board Programming

The instructions below describe the process for programming the A9 Core code on the main CPU board on the 7SR5 platform.

5.2.1 Prerequisite Tools

5.2.1.1 USB Driver (DIGSI USB Driver)

The DIGSI USB drivers are used to program the device in boot mode via the USB port. If you do not have the DIGSI USB drivers already installed, then they can be found here (Siemens DIGSI V4.88 Drivers):

I:\Proj-Data-D\Hbb\351482_Reyrolle_7SRn_Native_61850\P.M100-P.M380_GeneralSubjects\Planning\NKG\Swap\20170825_Loaders\UsbDrivers\Setup.msi

5.2.2 KON file loader

The KON file can contain a whole flash image or an image for a single flash partition.

With respect to the firmware this is an executable binary that has been converted into a flash image.

The firmware upload program can be copied onto your local driver and run from there.

You can also associate it with KON files so that double clicking on a KON file will automatically open the firmware update tool, which can be found here:

I:\Proj-Data-D\Hbb\351482_Reyrolle_7SRn_Native_61850\P.M100-P.M380_GeneralSubjects\Planning\NKG\Swap\20170825_Loaders\FIRMWAREUPDATE.EXE

5.2.2.1 Uploading a KON file

- Power off the relay.
- Connect a USB cable.
- Start the relay up with the DOWN key pressed – and hold this for 10 seconds to put the relay into boot mode.
- Load the desired KON file into the firmware update tool.

5.2.2.2 Configuring your COM port

Note from JS:
My Siprotec driver installed as COM25. The only way I could set the upload tool to use this by default was to modify the FirmwareUpdate.ini file (mine was in C:\Users\john.sprague\AppData\Roaming\SIEMENS\Siprotec\Firmware) setting ComPort=25. Then with the device in upload mode I could double click the KON file and load it.

5.2.3 Boot Fuses

A brand-new board will not have the boot fuses programmed (burned in), so the processor card will not be able to boot from the flash.

Presently the way to burn the fuses is by programming the A9 core with the Hebburn A9ToStartM4 code, using the IAR debugger and J-Link. This IAR project can be found in Vault:

$/5_DEVELOPMENT/PLATFORMS/518/FIRMWARE/A9ToStartM4/A9ToStartM4.eww

5.2.4 Boot ROM

5.2.4.1 First Time Programming

On a blank board the A9 Boot ROM will need to be programmed for the first time using the IAR Debugger and J-Link tools. The initial Boot ROM can be found here:

I:\Proj-Data-D\Hbb\351482_Reyrolle_7SRn_Native_61850\P.M100-P.M380_GeneralSubjects\Planning\NKG\Swap\NKG Binary\NKG_Binary.eww

After the Boot ROM has been initially loaded then it is possible to program it using the normal KON file method described above in Section 5.2.2.

5.2.4.2 Normal Programming

After the Boot ROM has been initially loaded then it is possible to program it using the normal KON file method described above in Section 5.2.2.

5.2.5 FPGA

The FPGA is programmed using a KON file, which is described above in Section 5.2.2.

5.2.6 Network Settings

Presently we can only connect directly from the PC to the Ethernet port.

- To do this you will need to configure the TCP/IP v4 settings for this NIC (ip address 192.168.0.11; netmask 255.255.255.0, no gateway).
- Connect the USB adapter directly to the top RJ45 socket on the relay.
- You should see the green light on the CPU board come on.
- The default IP address for the relay is 192.168.0.4

5.2.7 VxWorks

The VxWorks image is programmed using a KON file, which is described above in Section 5.2.2.

5.2.8 M4 Binary

The M4 firmware will also produce a KON file that can be downloaded.

6 M4 Firmware Architectural Topics

6.1 M4FW – CMP

The full CMP document for the 7SR5 platform can be found here:

7SR5 Configuration Management Plan - CMP_7SR5_Platform_SW.doc
$/518_7SRn_Native_61850/Software/7SR5 Platform/CMP_7SR5_Platform_SW.doc
V:\6\518_7SRn_Native_61850\Software\7SR5 Platform\CMP_7SR5_Platform_SW.doc

6.1.1 Software Project Shell (SPS)

Software Project Shell V3.41.00.10 has been released to support the new 7SR5 platform tools. There are three new tool files relevant to the 7SR5 platform:

- iMX6SX_M4_Application.tl - Main processor application
- iMX6SX_M4_Library.tl - Main processor library
- K22F_SlaveCard_Application.tl - Slave card application

SPS Version Number:
2974H80001 R017 - Software Project Shell 3.41.00.10

Vault Location:
$/974_SoftwareProjectShell/2974H80001 R017 - Software Project Shell 3.41.00.10/

6.1.2 IAR Compiler

All firmware on the 7SR5 platform is built using the IAR compiler V7.70.1.

Once the compiler is installed the network license needs to be manually configured using the information below.

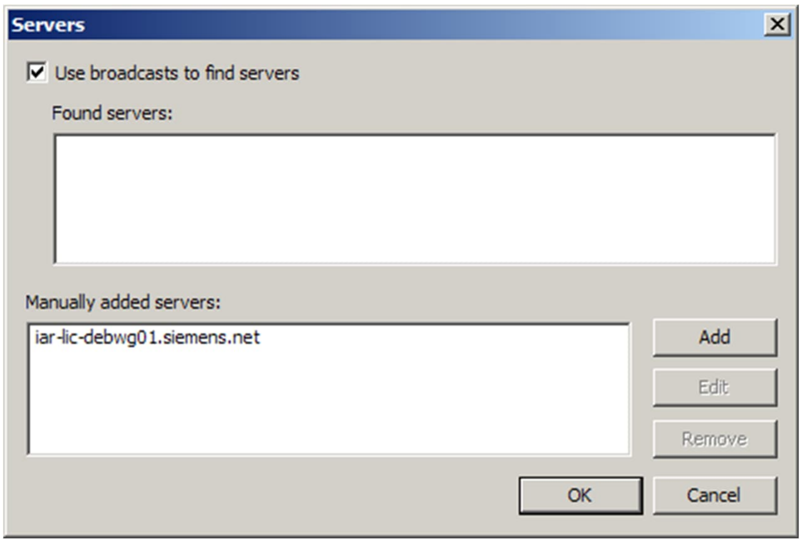


Figure 10 – Manual configuration of the IAR network license.

Compiler Version:
IAR ANSI C/C++ Compiler V7.70.1/W32 for ARM

Install Location:
Y:\437_Tools\v7.70.1\EWARM-CD-7701-11486\ew\setup.exe

Network license:
iar-lic-debwg01.siemens.net

6.1.3 J-Link Debugging Tools

The SEGGER J-Link debug tool is used to debug on the 7SR5 platform and will require V6.14e of the J-Link software, which can be found here:

Y:\437_Tools\J-Link\JLink_Windows_V634c.exe

6.2 M4FW – Slave Card Programming

This section describes the process to program the slave cards personality.

6.2.1 Basic Card Programming

- The personality of a slave card is programmed using an ASCII terminal program.
- The default COM port options are: 57600, No Parity and 8 Stop bits.
- The programming sequence and commands are listed below:

Login in:	CW 866315	
Enter MF mode:	start_mf_prog	*// Forces a restart*
Login in:	CW 866315	
Set Personality:	CPUID <Article Number> <Unique Serial Number>	
Exit MF mode:	quit_mf_prog	

6.2.2 Analogue Card Programming

In addition to the basic personality programming (as described above) the analogue cards need to have their calibration setup. The basic process to zero the calibration should be performed on all new cards, if they are not going to be calibrated immediately. The process is listed below:

Login in:	CW 866315
Reset calibration:	ZEROCAL
Store calibration:	STORECAL

6.2.3 Slave Card Article Numbers

Name	Description	Article Number
CT Card	4 CT + 3BI	C53040-A70-C77-x
VT Card	4 VT + 1BI + 2BO	C53040-A70-C78-x
IO Card	5BI + 2BO 5BI + 2HDBO	C53040-A70-C79 x C53207-A431-B81 x C53207-A431-B82 x C53207-A432-B171 x C53207-A432-B172 x C53207-A432-B181 x C53207-A432-B182 x

PSU Card	PSU + 4BO	C53040-A70-C75-x C53207-A431-B41 x C53207-A431-B221 x
S6 Fascia Card	S6,128x128LCD,7K,28L	C53207-A431-B21 x C53207-A431-B211 x
S12 Fascia Card	S12,128x128LCD,7K,28L	C53207-A431-B11 x C53207-A431-B191 x
S8 Fascia Card	S8,128x128LCD,7K,28L	C53207-A431-B161 x C53207-A431-B201 x
ARC Card	ARC PROTECTION WITH HS BO	C53207-A431-B131 x
TSI Card	8xTSI	C53207-A431-B91 x

Table 6-1 Slave Card Article Numbers

6.3 M4FW – Sampling

6.3.1 Analogue Input RDL naming convention.

The raw inputs for the CT and VT cards will be named as follows:

Name	Description
AN_I1 … AN_In	Phase Current Input
AN_S1 ... AN_Sn	Earth / SEF Input
AN_V1 … AN_Vn	Voltage Input

Table 6-2 Analogue Input RDL naming convention

This will be compatible with the existing RM and RC platforms.

6.4 M4FW – Timers

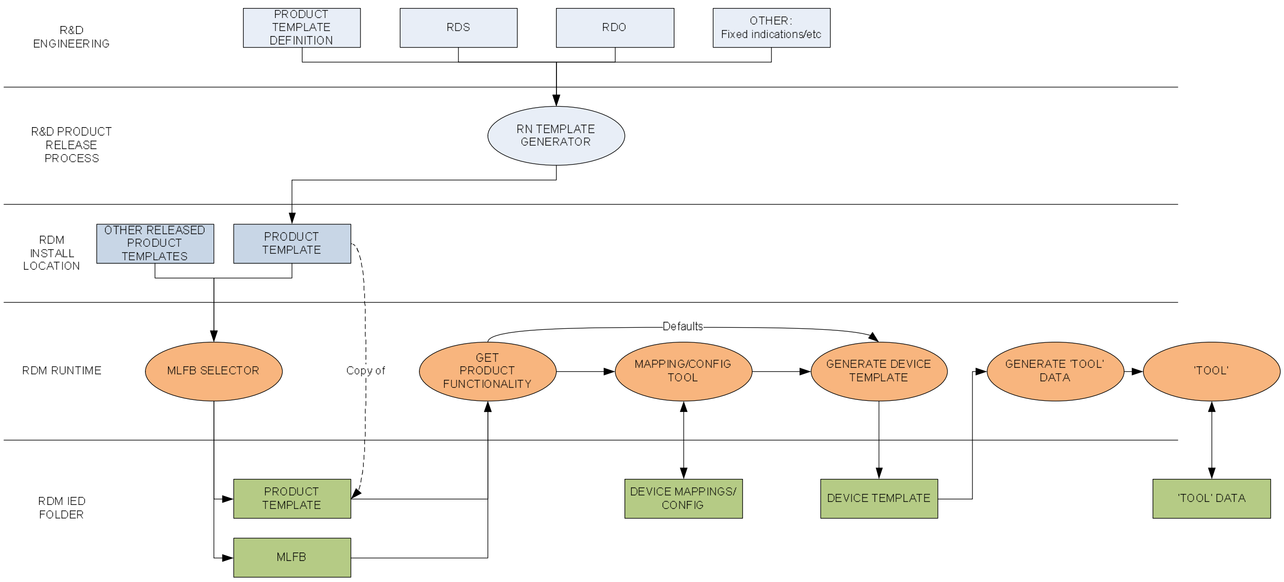
6.5 M4FW – Hardware timers

6.6 M4FW – Real Time Clock

7 7SR5 Devices Architectural Topics

8 Reydisp Manager Architectural Topics

8.1 Reydisp Manager Template Dataflow



8.2 Dynamic Setting View/Edit

Reydisp Manager 1 uses a *settings file* in RSF format to store settings. The source of the RSF file is a running device that is configured with the target MLFB. To modify a relay setting value, the target setting is found on the device by the index which specifies its location in the settings file; this means that the settings file must always be kept in step with the relay. A validity check of the settings file and relay setting database is performed by a CRC which checks that the structure matches. If the CRC validation fails, the entire setting database must be retrieved again from the relay. Any pending user modifications must then be re-applied to the settings and then the process re-attempted.

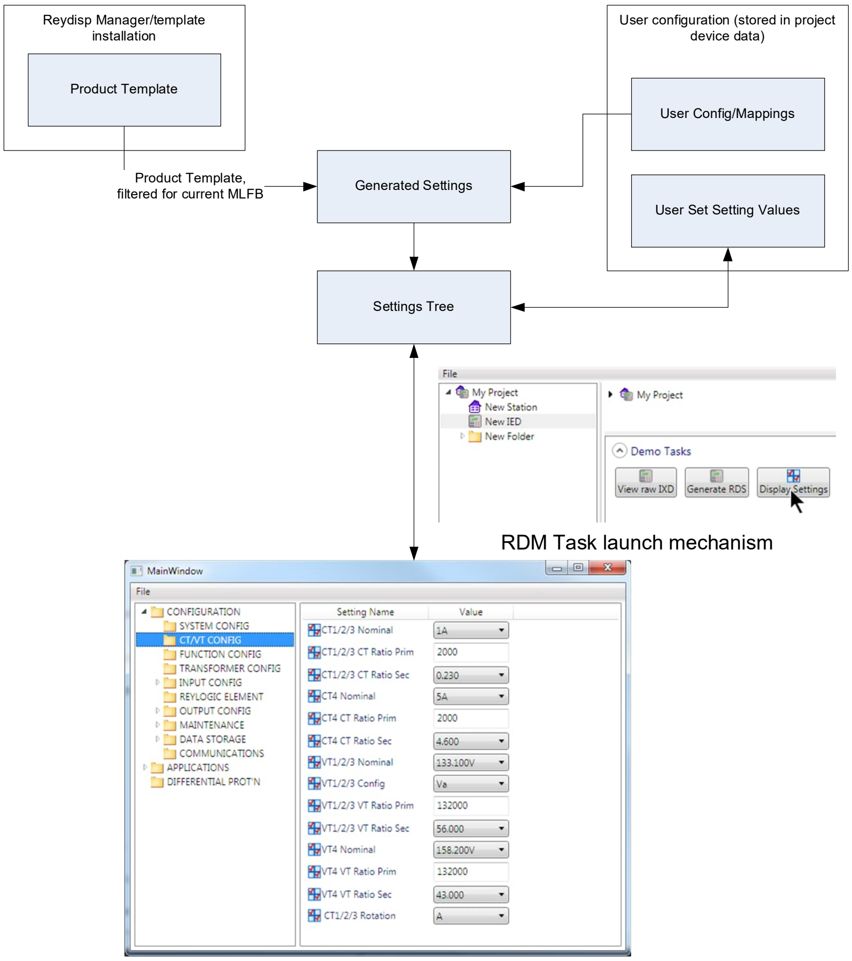
Producing device templates for Reydisp Manager 1 requires that settings files must be retrieved from running devices for a product release. This enables the user to configure the device offline without having been connected to it.

Handling settings in RDM2 for 7SR5 devices shall be modified with these goals:

- A setting tree should be deterministic from template data without the need to connect to a running device.
- Setting values should be communicated with a unique identifier, rather than by setting index (the position of the setting in the setting database).
- Setting values should be stored with only minimal data that can be applied to multiple IED configurations.
    - Setting ID
    - Setting Value (raw; value index, for example)

- Setting Value (display; setting value displayed to user)
- Setting Group (the group the value is applied to)

- The setting tree should dynamically represent the current IED configuration. Considerations to be made are:
  - The IO for the IED will be determined by the MLFB. Any settings which directly reference IO (status inputs, binary outputs, LEDs, function keys, etc) will be generated to reflect the current IO configuration:
    - Output/Input matrix bitfield settings should have the correct size and bit names.
    - Any IO configuration settings (output delays, etc) should be generated multiple times; once for each IO instance (BO1, BO2, BO3, etc).
  - Any setting values which rely upon device configuration state will be generated to reflect that state.
    - User curves which are settable as a characteristic setting (for an IDMTL for example) should be selectable values.
    - Language – TODO: Don't know how language selection is happening.
  - Any setting that relies upon an element list (trip storage, for example) shall be generated based on which functionality is currently selected by user configuration.
    - TODO: How to know which elements go into which trip storage setting, for example?



Settings viewer/editor

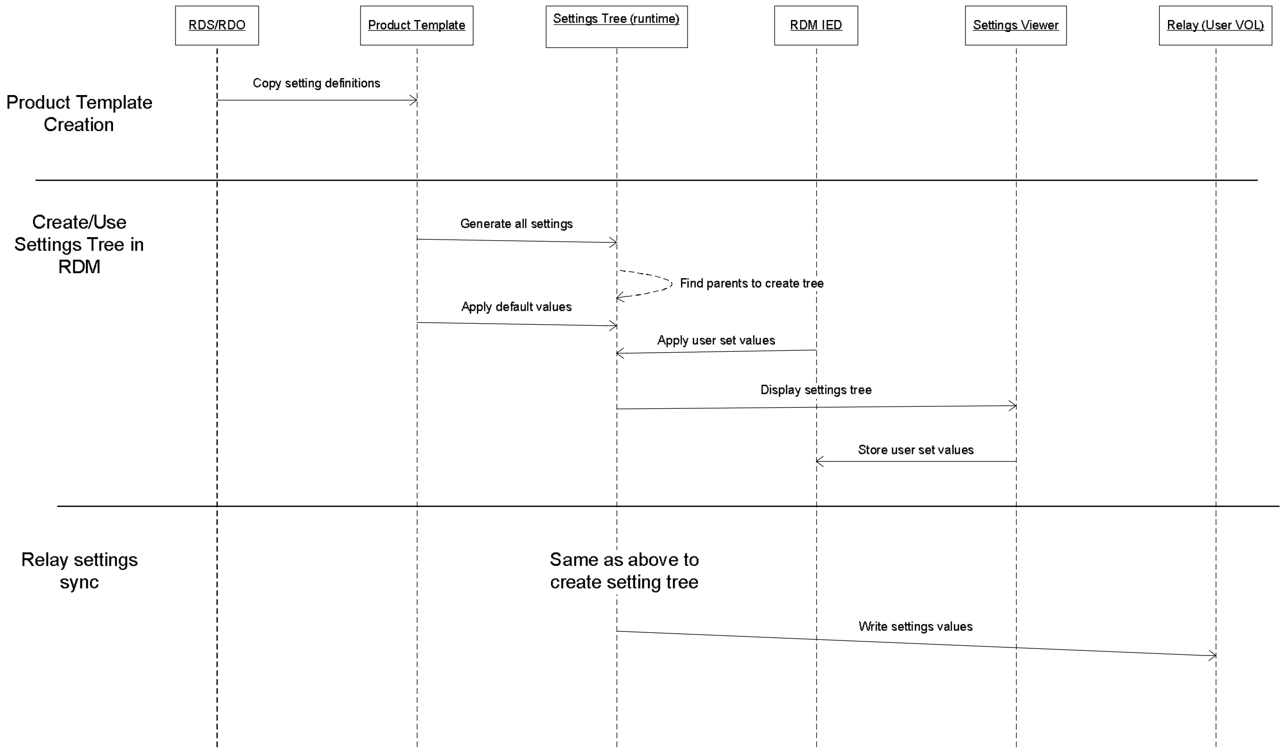


Figure 11 - 7SR5/RDM Settings Sequence

8.3 Complex Settings

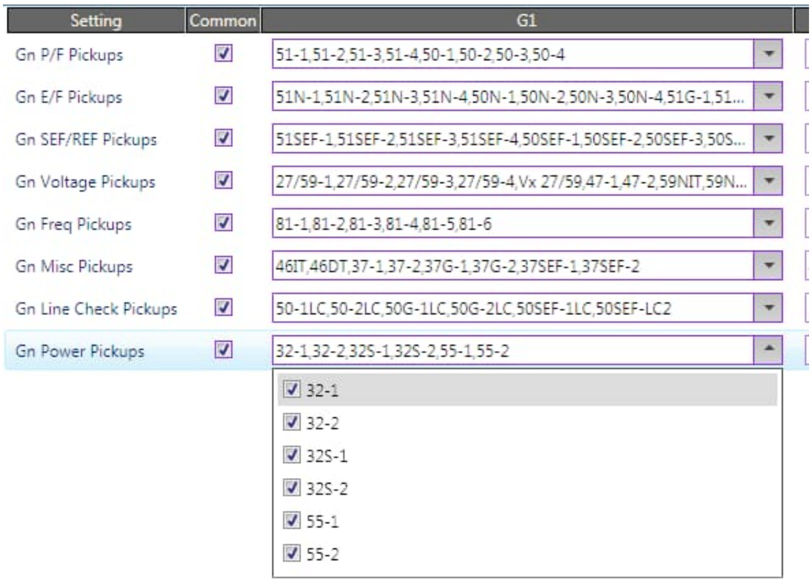
Some settings value in definition depending on the MLFB (IO, etc) or configurations state. Additional considerations need to be made when creating a setting of these types.

8.3.1 I/O Matrix Bitfield settings

Settings that define the IO (binary inputs, binary outputs, LEDs, virtuals, Fn Keys, etc) will change in definition depending on the MLFB that is used to initialize them, each bit in the bitfield representing IO that is present. The IO configuration for the selected MLFB will be given to the settings tree generator so that it may correctly initialize I/O matrix bitfield settings.

8.3.2 Or Masks (Pickup Config) Settings

Pickup config settings contain a list of bits for each element of a given function group. The relay RDL builds these settings by RDL BOOL registering a bit with an RDL ORMask object (by name). This process also needs to be performed in the Reydisp Manager template libraries for generating settings trees.



Implementation notes:

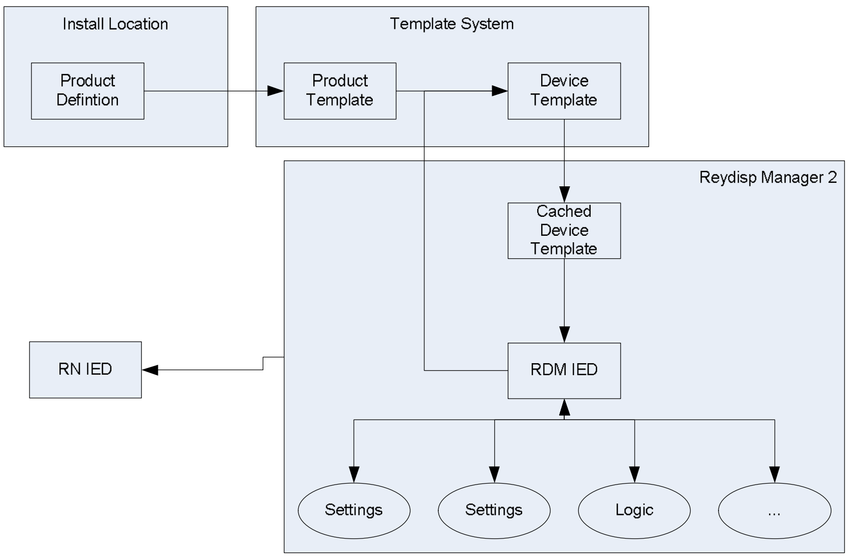
In RDL BOOL: <RegisterOrMask OrMaskName="{$Str OrMsk Name}" BitName="{$TagName}" />

Im RDL ORMSK:

<SimpleSetting …>
<DisplayRule>
<CollectOrMask OrMaskName="{$OBJNAME}" />
</DisplayRule>
</SimpleSetting>

8.4 Reydisp Manager 7SR5 Template Workflow

Product Definition	→	Product Template	→	Device Template
Contains description of product, including: Protection functionality, IO description, MLFB structure, Rules for element CT/VT mapping restrictions.	A publish process, 7SR5-Template generator, processes and validates the input data, collects RDS, collects RDO information, collects SCL fixed indications	Contains all template information required to configure an IED for every MLFB in the product.	Filters the product template down to what is required for a specific target MLFB (defined by Reydisp Manager). Filters the protection list down to what is specified by the user in Reydisp Manager. Applies element CT/VT mappings specified by the user in Reydisp Manager. Generates settings tree, user logic signals, events, SCL (etc) that are specific to the target MLFB and user configuration.	Contains:- Settings tree (with default values), User logic signals, SCL (IXD with default IED name and configuration), Events, RDS/RDO (internal representation to be converted to RDS).



9 References

1. 7SR5 Configuration Management Plan - CMP_7SR5_Platform_SW.doc
   $/518_7SRn_Native_61850/Software/7SR5 Platform/CMP_7SR5_Platform_SW.doc
   V:\6\518_7SRn_Native_61850\Software\7SR5 Platform\CMP_7SR5_Platform_SW.doc

2.

10 History

Author

Name	Department	Version	Date
Joerg Peters	EM EA PRO D 1	1.00	2013-01-31
Dr. Harald Kapp	EM EA PRO D 5 1	1.01	2013-01-31

Updates

Chapters / Pages changed	Version	Object and reason of change / Reference to change requirements
general	V1.00	first release after review
6 / 9	V1.01	Added history at the end of the document.
general	V1.03	Applicability changed to EM DG PRO
general	V1.04	Applicability changed to SI DG SA&P
General	V1.05	Applicability changed to SI DG EA-P&R
General	V1.06	Applicability changed to SI EA (Protection & Automation)