SCA BASED IMPLEMENTATION OF STANAG 4285 IN A JOINT EFFORT UNDER THE NATO RTO/IST PANEL

Sarvpreet Singh, M. Adrat, S. Couturier, M. Antweiler
Research Establishment for Applied Science (FGAN)
Wachtberg, Germany
singh@fgan.de

Martin Phisel, Steve Bernier
Communication Research Centre (CRC)
Ottawa, Canada
martin.phisel@crc.ca

ABSTRACT

The NATO RTO/IST *Regular Task Group on Software Defined Radio* (RTG on SDR) is working on the portability and interoperability of waveforms in a Software Communications Architecture (SCA) based environment using the STANAG 4285 as a test waveform. This paper presents several SCA based realizations of this waveform at different granularity levels (number of divisions). The paper discusses the overheads incurred by dividing the SCA based transmitter resource into two and four resources. It shows that the overheads increase linearly with the increase in the number of resource divisions. An important result from the analysis is that, an SCA resource should perform considerable signal processing to overcome the overheads associated for its functioning. It is also seen that there are some fixed overheads for running an SCA resource along with some variable costs, depending on the amount of data processed by the waveform.

1. INTRODUCTION

It is anticipated that the future SDRs in the military domain will be based on the *Software Communications Architecture* (SCA) [1]. Since the SCA is a US development, only limited knowledge and experience is available to the other NATO nations till now. Thus, in mid 2007 a *Regular Task Group on Software Defined Radio* (RTG on SDR) has been established under the umbrella of the NATO RTO/IST panel (Research and Technology Organization / Information System Technology) [2]. The group consists of experts from industry, universities and governmental organizations of around 10 NATO nations. The focus on technical expertise and the possibility of sharing experiences on SCA and SDR in the international cooperation is unique and of value to its members. The key objectives of this group are:

- the SCA based implementation of waveforms,
- demonstrating their portability on various platforms,
- and finally, demonstrating the interoperability of the different realizations.

During this three step process, the group also aims to share their learning experiences among all group members. In this paper, we will present some of the results which have been achieved at FGAN so far, focusing on the SCA based implementations. In Section 2, we will briefly review the SCA development tool suite being used by us as well as the test waveform. The group agreed to work on a common waveform called the STANAG 4285 [3]. In Section 3, we will describe the different implementations at different granularity levels. In Section 4, we will present some profiling results of these granular implementations. In Section 5, we will compare the profiling results of implementations at different configurations and discuss their cost overheads.

This research project was performed under contract with the Technical Center for Information Technology and Electronics (WTD-81), Germany.

2. DEVELOPMENT TOOLS & WAVEFORM

2.1. SCA Development Tool Suite

The tool used for the development of SCA based waveform for this work was the SCARI Software Suite [4] developed by the Communication Research Centre (CRC), Canada. The complete tool suite consists of a full featured JTRS SCA Core Framework 2.2, a component development library (CDL), and the SCA Architect modeling tool. The SCA Architect is an IDE (Integrated Development Environment) provided as a plug-in for the Eclipse Framework. The tool allows us to create graphical models of various elements, to perform real time validations, to generate code responsible for implementing SCA specifications to the elements, and to assemble the various elements into applications and nodes.

Since code can be generated from models, the tool allows the waveform developers to focus more on implementing the specific functionality of a waveform and frees the developers from the constraints of the SCA.

2.2. STANAG 4285 Waveform

The waveform used for the work presented in this paper is the STANAG 4285 [3]. It is a NATO standard for HF communication. STANAG 4285 offers six modes between 75 bits/sec and 2.4 kbit/sec. The individual modes can be set by the three configuration parameters: Code Rate (for encoding), Interleaver Size (for interleaving) and PSK Scheme (for modulation).

The block diagram in Fig. 1 shows two parts: *Frame Collection* and *Frame Processing*. The *Frame Collection*

part is a kind of state machine, and the first state we see is the 'AGC' (Automatic Gain Control). It provides prerequisite information for the AGC at the receiver to maintain a fairly constant input signal level. The next state is the 'SOM' (Start Of Message) which gives information regarding where the message starts. The 'User Data' state comes next which defines the actual data. The 'EOM' (End Of Message) state gives information about where the message ends. And the 'Flush Bits' state is used to terminate all the processing functions to a defined state (Ex: to fill in the interleaver with zeros). Each of these states can be switched to perform the *Frame Processing*.

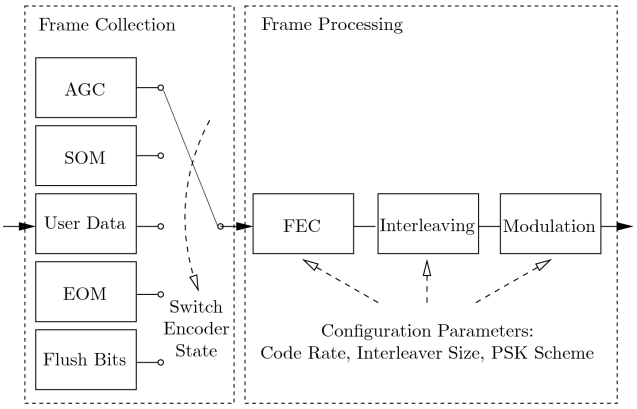

Fig 1: Block diagram of STANAG 4285 transmitter

We can switch between the states and pass the respective data to FEC, Interleaving, and Coding for *Frame Processing*. In the *Frame Processing* part, the 'FEC' component is used for the error correction with code rate 1/2 and memory 6 convolutional code with generator polynomial $G(133, 177)_8$. Higher code rates like 2/3 for the 2.4 kbit/sec mode are achieved by puncturing. Lower code rates like 1/16 for the 75 bit/sec mode are achieved by repetition coding. The 'Interleaving' component is responsible for data permutation to break up burst error events (both 'long' and 'short'). The 'Modulation' component includes scrambling and filtering in addition to modulation.

3. SCA BASED IMPLEMENTATIONS

3.1. Basic Implementation: TX as one SCA Resource

The functionality of the STANAG 4285 waveform is provided by one of the RTO/IST RTG on SDR members, Telefunken Racoms. The functionality is implemented in ANSI-C in two parts, namely the transmitter (TX) and the receiver (RX). To use this functionality in a C++ based SCA environment resource, separate libraries are compiled from the C-code for TX and RX. For the basic implementation, two SCA based resources are implemented using the CRC tools, one each for TX and RX. The respective libraries are called from the SCA based resources to perform the signal processing part of the waveform. To keep the working simple, we use a text of characters which is encoded by TX and in turn decoded by RX. The flow of data from TX resource to RX resource is in the form of IQ-values through SCA ports. The communication is handled by CORBA.

3.2. TX split into two SCA Resources

To increase the granularity of the application, a decision is made to separate the TX resource further into sub resources. Since the goal of the *RTO/IST RTG on SDR* is to achieve easy portability of the waveforms, we need several resources of the waveform which can then be shared with other participating nations of the group to test portability and interoperability. The functionality of TX code is separated into two parts: *Frame Collection* and *Frame Processing* (see Fig. 1). Separate libraries are made for both the separated functionalities, to be used with the SCA resources. Two new SCA resources are then implemented using the CRC tools and the library calls are made from the respective resources. The RX resource is reused from the previously made application.

3.3. TX split into four SCA Resources

To achieve higher granularity TX is separated to a higher level. Reusing the resource responsible for *Frame Collection*, the *Frame Processing* functionality is divided further into three parts handling different responsibilities. The first part is responsible for FEC encoding, the second part for interleaving and the third part for modulation. Once again, the CRC tools are used for making three new SCA based resources for each of the above mentioned functionalities. Like with previous resources, the code responsible for performing the functionalities is compiled as a library to be used with the SCA resources.

3.4. Summary of different Implementations

Table 1 gives the summary of different implementations with the number and name of resources as discussed earlier in this section. The table also provides us the notations for the resources used in Section 5.

Table 1: Resource Implementations with notations

Implementations	**Resources**			
TX as 1 Resource	Basic Implementation [1 TX]			
TX split in 2 Resources	Frame Collection [2 TX (1)]	Frame Processing [2 TX (2)]		
TX split in 4 Resources	Frame Collection [4 TX (1)]	FEC Encoding [4 TX (2)]	Interleaving [4 TX (3)]	Modulation [4 TX (4)]

4. APPLICATION PROFILING

4.1. Profiling Tools

The different implementations were profiled to see their individual overheads on the system. The profiling tool called Callgrind, available from the Valgrind [5] tool suite was used to profile the TX resources for the different test implementations.

To perform profiling, a text of characters was read from a text file and then used by TX for encoding. The profiling results presented in this section are for 5080 characters encoded by the 75 bits/sec mode with a 'long' Interleaver. The profiling results include the waveform specific overheads of STANAG 4285 for transmitting the EOM, SOM, flush bits associated to the 5080 characters. Thus, we first calculate the total number of effective bits processed. This includes 32 bits each for the SOM and EOM and 870 flush bits in addition to the 40640 bits of the actual message (5080 char = 40640 bits). Therefore, the total number of bits comes out to be 41574. Now, according to the 75 bits/sec mode of STANAG 4285 with a 'long' interleaver, 8 bits (1 Byte) are transmitted per frame. Therefore, the total number of frames are 5197 (approx).

Table 2: Determination of a Number of Frames

Bits/sec	SOM	User Data	EOM	Flush Bits	Total Bits	Bits/Frame	Number of Frames
75	32	1200	32	870	2134	8	267
75	32	40640	32	870	41574	8	5197
75	32	81280	32	870	82214	8	10277
75	32	162560	32	870	163494	8	20437

All the profiling results in Section 5 will be determined as a function of the number of processed frames. Table 2 shows the calculation for the 'Number of Frames' for different amount of processed data. Only the second row is used in this section. The values from other rows are used and discussed in Section 5.

4.2. First Profiling Results

From the results presented in [6], we know that the overhead cost of CORBA on SCA based components is not significant as compared to the cost of signal processing itself. In this paper however, we investigate the overhead incurred on splitting a single SCA based resource into multiple SCA based resources keeping in mind the long term goal of portability. The reason for this investigation is also based on the fact that in future we will be implementing different component resources on different processors depending upon the requirement imposed by the functionality of that particular resource. Thus, it is important to know the overheads incurred to achieve this. Although the STANAG 4285 waveform is a much simpler waveform (used for academic purposes) which can be implemented on a single GPP, we use it as a reference to implement more complex waveforms which require different parts to be implemented on different processors.

A point to note here is that only TX components were profiled to gain information on the overheads incurred by the splitting up of single resource to multiple resources. The overhead cost of the 'Assembly Controller' is not discussed. Another point worth mentioning here is that the protocol used for the CORBA communication between the resources in our simulations is TCP. Since TCP is not the fastest available transport protocol, smaller latency can be achieved by using better and faster transport protocols [7]. Real-time CORBA products offer several off-the-shelf pluggable transports and the possibility to implement new ones [8]. It is acknowledged that using a different, more optimized transport could provide better performances and thus affect positively the fixed cost associated with CORBA.

Some terms used with respect to the profiling tool are defined first:

- The 'Cost' of a function …
  is defined by the event counts of a particular function. This means the number of instructions/data accesses. It also tells us of the instructions that do/do not reference memory.
- The 'Self Cost' …
  is the cost of the function itself.
- The 'Inclusive Cost' …
  is the cost of all functions called by a particular function. This can also be called the cumulative cost.

In the following sections we focus on the 'Self Cost'. Whenever we use the term 'cost' in the following sections, we refer to the 'Self Cost'.

4.3. Basic Implementation: TX as one SCA Resource

After analyzing the profiling data for '1 TX', we find that the total instruction fetch cost of this resource is 486.82 thousand. The total number of functions involved in the complete processing of the resource is found to be 3988. Table 3 shows the functions executed in the resource with their respective self cost percentage. The table shows the signal processing function cost of the 'Executable', other function costs to run the 'Executable' and the overhead costs contributed by TAO, SCA, ACE, C/C++, ld.so on the implementation. We can see that the 'Modulation' component has the highest relative self cost of 65.51% in comparison to the total cost. This functionality is responsible for implementing the modulating, scrambling and filtering scheme for the transmitter. The 'Interleaving' and 'Encoding' functionality contributed to a self cost of 1.97% and 0.05% respectively. The 'Frame Processing'

function contributed 0.31% to the self cost. In addition to calling the 'Interleaving' and 'Encoding' functionality, the 'Frame Processing' itself performs the calculation of bit streams from the encoded 32 bit values calculated by the 'Encoder'. The 'Frame Collection' basically works as a state machine. And its functionality of including the AGC prefix, calculating the EOM and SOM and including the flush bits contributed to a low value of 0.01% in self cost. 6.3% is the self cost in order to run the signal processing functions in the 'Executable'.The rest of the self cost from the remaining 3983 functions are contributed by the TAO calls, CORBA function calls, SCA specifications and other standard C/C++ functions like malloc, strcpy, memcpy etc.

Table 3: Self Cost Table for TX as 1 resource

Functions	**Self Cost (%) 1 TX**
Modulation (in Exe)	65.51
Interleaving (in Exe)	1.97
FEC Encoding (in Exe)	0.05
Frame Processing (in Exe)	0.31
Frame Collection (in Exe)	0.01
Other Functions (in Exe)	6.39
Overhead (TAO, SCA, …..)	25.76

4.4. TX split into two SCA Resources

The profiling results for the two resource implementation are shown in Table 4. It can be seen that the first resource, '2 TX (1)' which is responsible for the 'Frame Collection' does not put a significant overhead on the functionality of the resource. The self cost is found to be 0.09% and an additional cost of 0.81% to execute the functionality. The total instruction fetch cost of the resource is 52.15 thousand. The major contributors to self cost of this resource are the TAO orb, CORBA function calls in addition to other standard C/C++ functions.

Table 4: Self Cost Table for TX as 2 resources

Functions \ **Self Cost (%)**	**2 TX(1)** (Frame Collection)	**2 TX(2)** (Frame Processing)
Modulation (in Exe)	-	64.69
Interleaving (in Exe)	-	1.94
FEC Encoding (in Exe)	-	0.04
Frame Processing (in Exe)	-	0.30
Frame Collection (in Exe)	0.09	-
Other Functions (in Exe)	0.81	6.32
Overhead (TAO, SCA, …..)	99.10	26.71

The reason for the high percentage (99.10%) of self cost for TAO orb, CORBA, C/C++ function calls is contributed to the fact that the signal processing functionality of this resource is not significant. The resource should perform significant signal processing to exceed the fixed costs. Thus, the relative values of the non signal processing functions are high for this resource. Moreover, out of the above mentioned major overhead contributing functions, the cost of some functionalities remains fixed for all resources. This is explained in more detail in Section 5.

For the second resource, '2 TX (2)' performing the 'Frame Processing' we get similar cost values for the functions as for the single resource. Table 4 also shows the self cost of the major cost contributor functions by both resources. The total instruction fetch cost of the resource is 493.31 thousand. We are aware of the fact that the '2 TX (2)' has a higher self cost than the '1 TX' (486.82 thousand). The reason for this also is given in the Section 5.

4.5. TX split into four SCA Resources

For the four resource implementation, the results for the first resource, '4 TX (1)' which is responsible for 'Frame Collection' is again similar to the one mentioned above because it is reused. The resource, '4 TX (2)' which is responsible for 'Encoding' also does not show any significant values for its data processing functions. The total self cost for this resource is found to be 56.54 thousand.

Table 5: Self Cost Table for TX as 4 resources

Functions \ **Self Cost (%)**	**4 TX (1)** (Frame Collection)	**4 TX (2)** (Encoding)	**4 TX (3)** (Interleaving)	**4 TX (4)** (Modulation)
Modulation (in Exe)	-	-	-	60.87
Interleaving (in Exe)	-	-	10.11	-
FEC Encoding (in Exe)		0.62	-	-
Frame Processing (in Exe)	-	-	-	-
Frame collection (in Exe)	0.09	-	-	-
Other Functions (in Exe)	0.81	3.25	12.13	8.3
Overhead (TAO, SCA, …)	99.10	96.13	77.76	30.83

In the resource, '4 TX (3)' performing the 'Interleaving', the percentage self cost is 10.11%. The total self cost for this resource is found to be 112.1 thousand. Moreover, we see that the resource performing the modulation is using the maximum consumption with the percentage self cost of 60.87%. The total self cost in this case is 524.3 thousand. Again, the reason for this high self cost is given in the next section. We can again see that the relative cost of 'Overhead' is high for the first three

resources. The reason again is the low cost incurred by the signal processing functionality of these resources.

5. PERFORMANCE COMPARISON

In this section we take up profiling information from various tests and compare them. We performed several profiling tests of the implementations with different amount of processing data to give us a clearer picture of the comparisons. After examining the data from the profiling results, we classify the total cost of running an implementation on the basis of ELF (Executable and Linking format) objects. The major contributions towards the total cost made by the ELFs were under the Executable, ACE, TAO, SCA, C/C++ and ld.so. 'ld.so' is the dynamic library loader to find and load the shared libraries for the binaries. ACE and TAO are responsible for the CORBA costs. One might argue that C/C++ and 'ld.so' are not SCA overheads but we will see from the discussion in this section that the qualitative conclusions remain the same but quantitative terms might slightly change due to them.

We performed several tests for processing 150, 5080, 10160, 20320 characters respectively with all the three implementations (1 TX, 2 TX, 4 TX). For a better understanding of the vast amount of collected profiled data, we made graphs for each implementation displaying the costs involved for them. Fig. 2 shows the cost of different contributors with different amount of processed data for different implementations. The profiled cost value can be seen as triangles, circles, squares etc. in these figures for different number of frames (different amount of processed data). The number of frames are calculated and used from Table 2. Thus, the x-axis in these figures shows the number of frames and the y-axis show the self cost. We will now discuss the results for each implementation.

5.1. One TX Resource Cost Comparison

In the upper left part of Fig. 2, we see different curves representing different cost contributors to the implementation. The dotted curves are for the overheads due to TAO, ACE, SCA, C/C++, ld.so. When we compare the solid cost curve of 'Executable' with the dashed curve of the sum of these overhead costs, we can observe that the cost overhead is less if we process more data. We can see that for processing 150 characters (267 Frames); the cost of the 'Executable' is slightly less than the sum of all overhead costs (cmp. Table 3). But when we process more data, we see that the overhead cost is exceeded. Thus, at 150 characters (267 Frames) we see about 50% 'Executable' cost and 50% overhead cost in contrast to the 75% 'Executable' cost and 25% overhead cost for higher number of frames. It is worth to be mentioned that the costs of all overheads slightly increases with the amount of processed data. The only exception is the costs of "ld.so" which is fairly constant for all number of frames. Another point to note here is that there is a significant overload by C/C++ and TAO. Moreover, the costs for the overload do not start from zero. There is a substantial overload as soon as few frames are processed.

5.2. Two TX Resources Cost Comparison

The upper right part of Fig 2 shows the cost graphs for the '2 TX' resource implementation. The left graph shows the costs for '2 TX (1)' and the right one shows for '2 TX (2)'. We can see that the total overhead cost shown by the dashed

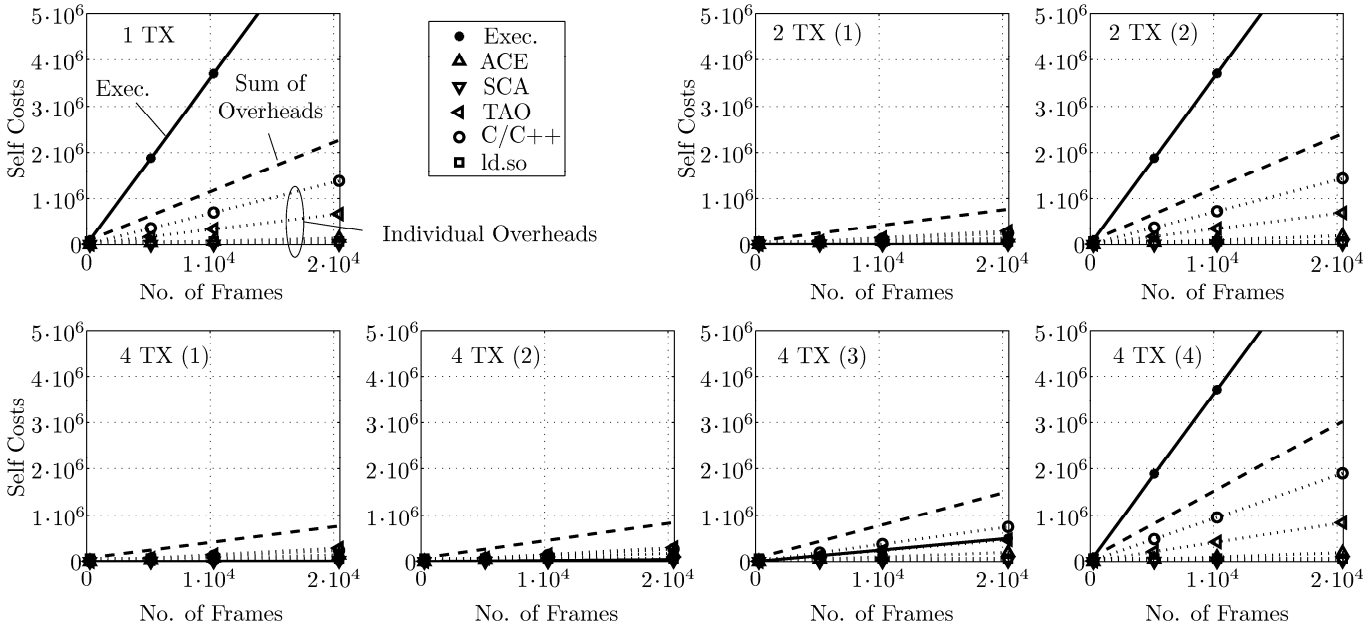

Fig. 2: Profiling results for the individual SCA resources of the different implementations 1 TX, 2 TX, 4 TX

curve for '2 TX (1)' is high and the cost of 'Executable' negligibly small. The overhead becomes even higher at higher amount of data being processed i.e. at higher number of frames. The reason is that the slope of total overhead costs is higher than the cost of 'Executable'. Thus, we arrive at a very important result i.e. making a resource like '2 TX (1)' is not worthy enough because the overhead costs of this resource always exceed the signal processing cost. The overhead increase even more as more frames are processed. The results for the '2 TX (2)' on the other hand are similar to the '1 TX' as this is the major signal processing cost contributor resource in the implementation.

5.3. Four TX Resources Cost Comparison

The lower part of Fig 2 shows the cost graphs for the '4 TX' resource implementation. Again, we see that the total overhead costs of '4 TX (1)', '4 TX (2)', '4 TX (3)' is higher than their signal processing cost i.e. cost of their 'Executable'. The '4 TX (4)' responsible for performing modulation has a higher cost of its 'Executable' the total overhead costs. This shows that the splitting the resource this way also is not efficient as the first three resources are contributing to higher cost overheads in the implementation.

5.4. Overall Cost Comparison

The left side of Fig. 3 shows us a summation of the results of the above sections in the form of six curves. The solid curves represent the total "Executable" cost and the dashed curves represent the total overhead cost for each of the three implementations. Thus, the lower solid curve shows the cost of "Executable" for 1 TX and for 2 TX ((1) + (2)). Both costs are nearly the same because the separation of 1 TX into *FrameCollection* (2 TX (1)) and *FrameProcessing* (2 TX (2)) required only minor changes in the original C-code. From the signal processing point of view the cost of "Executable" for splitting the '1 TX' to '2 TX (1)' and '2 TX (2)' is almost negligible. The upper solid curve shows the total cost of "Executable" for 4 TX (4 TX (1) + 4 TX (2) + 4 TX (3) + 4 TX (4)). Here the costs are higher because the separation of *FrameProcessing* into "FEC", "Interleaving" and "Modulation" made a reorganization of the C-code necessary which leads to higher signal processing costs. The dashed curves show the total overhead costs for the respective implementations. Here we can observe that the total overhead costs increases due to the splitting of resource. Thus, we observe after extrapolating this graph (with the same pattern) that for an implementation with more than 5 TX resources will result in higher overhead costs in comparison to the cost of the 'Executable'.

The right side of Fig. 3 shows the comparison of the relative overhead cost with the number of frames processed. An important result which comes forward from it is that the relative overhead cost is higher for processing less number of frames. But it is lower and almost constant for processing larger number of frames. We can also see that the relative cost for more number of resource implementation is higher.

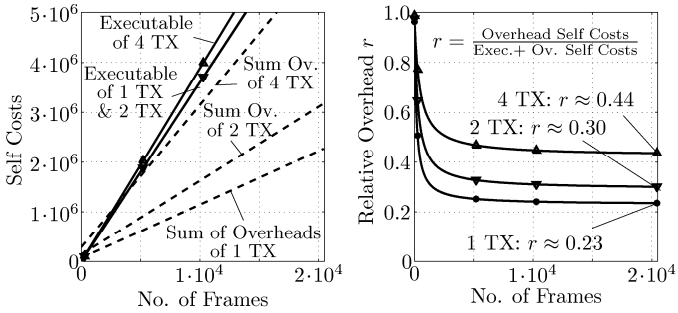

Fig. 3: Comparison of profiling results for the different implementations 1 TX, 2 TX, 4TX

Similar results came forward by processing the same data at other modes (e.g. 2.4 kbits/sec). The results showed higher cost of the 'Executable' because of more signal processing. There was higher overhead due to CORBA as larger packets were being transmitted between resources.

6. CONCLUSION

The paper presents the details of the profiling analysis of various SCA based implementations. The test waveform for all the implementations used was STANAG 4285. We see that the division of an SCA based resource creates overheads. All the overhead costs and the 'Executable' costs increase almost linearly. Most overheads have some fixed costs and the variable costs depend on the number of frames processed. We also see that the resources should not be made small and not in large numbers. In other words, the signal processing done by a resource should be significant to overcome the costs of non signal processing functionality.

7. REFERENCES

[1] "SCA" http://sca.jpeojtrs.mil/
[2] "RTO Group" http://www.rta.nato.int/
[3] NATO, Military Agency for Standardization (MAS), "STANAG 4285: Characteristics of 1200/2400/3600 Bits Per Second Single Tone Modulators/Demodulators for HF Radio Links", 1989.
[4] "SCARI Software Suite 2007" http://www.crc.ca/
[5] "Valgrind Tool Suite" http://valgrind.org/
[6] Philip J. Balister, Max Robert, Jeffery H.Reed, "Impact of the use of CORBA for Inter-Component Communication in SCA Based Radio", *SDR Forum,* 2006.
[7] Steve Bernier, "SCA Myths and Realities", *SMi Annual Software Radio Conference, London, UK,* 2006.
[8] Giovanni Middioni, "CORBA over VMEbus Transport for Software Defined Radios" http://motorola.com/