Memory Restructuring in WSIP

Document Number	DIO – WSIP MEM DESG - 100
Date of Release	6.4.2000
Document File	Memdesg-100.doc
Author(s)	K.Narayanan
Revisions	1.0.1

Contents

Chapter 1

1.1 Purpose

This document discusses the memory organisation of the WSIP application code and its re-organisation to accommodate the growth of the application as new features are added.

1.2 Scope

The scope of this document is to understand the memory organisation of the WSIP application code to be followed. Organisation in terms of Internal PM, Overlay PM(of 2187), External flash memory (*Paging*), Internal DM and Overlay DM are discussed here.

1.3 Terminology

PMOVLAY	Overlay registers used in 2187 processors
DMOVLAY	
WS	Wallset
WS-IP	Wallset with Internet Port
CNTR	counter register used for implementing loops in ADSP processor
PC	Program Counter that stores the next address to be used by the ADSP processor
IMASK	Register used by ADSP processor to handle interrupt nesting.
ISR	Interrupt Service Routine
PM	Program Memory
DM	Data Memory
Flash	Programmable external memory for the processor.
FEC	Forward error correction.

1.4 Definitions

Paging

The concept of loading parts of application code (**Page)**, from the external Flash programmable memory, during runtime into a common allocated area in the processors memory (**Paging Area).**

Chapter 2 Requirements

The WS provides only Voice call support,the WS application uses the 16 K of PM & DM of the 2183 processor implementing paging with a paging area of 2K.

The WSIP is used for both voice and internet connectivity, the WSIP application provides simultaneous voice and internet access using multiple connections. The WS application code is expanded to provide these additional services. The 2187 ADSP processor was thus chosen for WSIP for higher MIPS and larger memory access.

The WSIP 2187 processor has 32 K of PM & DM each of which 8 K of PM & DM each is internal and the rest are overlays.

In addition to above mentioned feature the WSIP supports RS232 based flash programming , master - slave state machine. This state machine is used by the master (2187 processor that is responsible for implementing the DECT stack) to boot the slave processor (2183 processor that implements ARQ and the RS232 driver. In addition future enhancements in the application would also require additional PM and DM.

The PM requirement of the WSIP was identified as:

1. To Increase the Paging Area from 2K to 5K for the less time critical foreground activities.
2. To use PM overlays available with the 2187 processor for time critical activities like the Interrupt Service Routines.

As a part of providing reliable Data service using the DECT system it was proposed to implement FEC using Reed-Soloman coding. Coding and Decoding required lookup tables to be implemented as initialised DM, initialized DM to the tune of ~ 4k was required.

The DM requirement of the WSIP was identified as:

1. Isolate the DM access of the *SwIntRcv* and *SwIntTx* modules that implement the FEC Coding and Decoding to either the internal DM or the external Overlay DM – 4. The overlay DM – 4 would hold the initialized DM required for FEC.

Chapter 3 Introduction

3.1 PM Scenario in WSIP:

The Wallset application code can be viewed in terms of processing done in the foreground and processing done in the ISR's. WS application has four interrupt handlers namely *WakeUp*, *Sp1RcvIsr*, *SwIntRcv,* & *TimerIsr* . Nesting of interrupts is allowed in the application. The handlers have been listed in the order of decreasing priority.

The *run_hdr* is the entry point for the application. On power-on foreground activities commence by a jump to main. The foreground activities may be interrupted by any of the above-mentioned ISR's, which are handled as per their priority by the handlers.

In WS of the16K of memory, 2K was allocated for the ***paging area.*** Modules like *Eep*, *Dlc*, *Nwk*, *Nwkmm*, *Iwu* may be swapped in from the external flash memory on runtime or on demand. These modules are not time critical and need to be used only if there is any specific event to be processed. The ISR handlers and other common routines use up the rest of the 14K of PM.

In WSIP the application has grown to exceed the16K limit in both the foreground and in the interrupt handlers. And hence the upgrade to 2187 processor. This processor is faster and has access to more memory. While the 2181 processor offers only 16K of PM and DM, the upgraded to 2187 offers 32K of PM & DM (8k Internal + Overlay 0, 4 & 5 of 8k each). To use the overlays a clear understanding of the code flow, the entry and exit for the various modules needs to be clearly identified making a *calltree* helps, Refer **-Annex1**

Since Paging has already been implemented in the WS code we propose to use them, in addition to the available overlay's of ADSP 2187 processor.

ISR's have a definite entry and exit points the *run_hdr*, hence moving the ISR's to the overlays as self contained modules is the first step in space creation in the internal PM to facilitate the increase in the paging area. Time critical tasks like the four interrupt handlers would now use the PM overlay 4 and 5. PM overlay 0 would still be a part of the Internal 8k vital for the foreground activities.

The current paging scenario in the WSIP application code:

Pages	Alloc	Used
Eepflpgm	3k	2696
Dlc	4k	2128
Iwu	4k	4066
Nwk	4k	2386
Nwkmm	4k	2725

Since *Iwu* page has already reached its capacity, for future expansion it is proposed that we increase the paging area to 5k. The paging area would be in the internal memory and not in PM overlay 0 to avoid the care to be taken in removing *CNTR's / Loop's* in swappable overlay Refer: **Annex 2**.

3.2 DM scenario in WSIP :

The WSIP application code uses 8k of internal DM and 8k of DM overlay 0**.** RS Encoding and Decoding requires approximately 4K DM variables as **initialized DM**. Since this a large chunk of DM, DMOVLAY 4 is proposed to be used.

RS Encoding and Decoding is done in *SwIntRcv* and *SwIntTx*, Therefore it is proposed that both these modules use variables either in DMOVLAY 4 (0x0 to 0x1fff) or the Internal memory (0x1fff to 0x3fdf).

The foreground processes use Internal PM and PM overlay 0. The ISR's would use respective PM overlay's, save & restore *PMOVLAY* to take care of ISR nesting. Ultimately PM overlay 0 is restored for the foreground processes to continue.

4.1 Program Memory organisation

INTERNAL PM	Refer **-Block Diagram**
Run_hdr	The interrupt vector table Takes care of loading and restoring PMOVLAY's for the various interrupt handlers.
Led Module	For Checking ISR timings.
TestRegs	Modules called in baseloop.
BatSchedular	UpdtBatVal called in Wakeisr is defined here.
InterOvlStubs	Rxswint in PM-Overlay 5 now extensively calls routines in PM-Overlay 4
Pt_main	InitialiseRf in PM-Overlay 4 uses stub to load overlay before initializing.
Wake/Mac_func	Has modules used by both ISR and foreground therefore must be in the internal PM.
Wsppl	TimerIsr and Serv10ms uses some common routines defined here.
Paging Area 5K	Dlc,Nwk,Nwkmm,Iwu,Eepflpgm pages
Library Routines	~350 PM words.A new file lib.dsp created that has all the lib routines to be restricted to the internal segment.
	Will have ~900K PM words free for future use.

PMOVLAY 0	**Swappable overlay No *CNTR***'S
Sysinit	SetCEnvironement -> pt_main, BbiDelay-> wakefunc Ensure no use of cntrs in this overlay either by using –mno-doloops flag (C file) or explicitly removing them from the DSP files.
+ routines that are	Used in the foreground
	Will have ~1500K PM words free for future use.

PMOVLAY 4	
WakeIsr,Sp1Rcv	Interrrupt handlers and their associated routines are in this overlay.
Enc/Decoding	For FEC will also fall here.
	Will have ~1000K PM words free for future use.

PMOVLAY 5	**Swappable overlay No *CNTR***'S
TimerIsr	Lowest priority & *CNTR* less
SwIntRcv	Some *CNTR*'s must be removed. SwIntRcv has calls to functions in overlay4 which have *CNTR*'s e.g Modem, Mac_isr, Encoding/decoding, routines in phlfunc module, RdRssiVal. A new file InerOvlStubs created for these interoverlay calls in the internal PM (call depth in this is increased by one – take care).

The above tables show us the usage of the various PM overlays and the internal PM.

4.1.1 Internal PM

The internal PM should contain only essential routines like the baseloop, routines like library routines that can be used by any overlay module and provision for inter overlay calls.

The library routines can no longer be linked during runtime using the group file, as it must be ensured that it resides in the internal PM to facilitate unversal usage across overlays.

Since modules in one overlay should be able to call modules in another overlay, provision to swap the overlay by setting appropriate value in the PMOVLAY register and then calling the modules should be done in the internal PM.

The *paging area* is allocated in the internal PM to ensure that the pages are loaded in a non-swappable area (with regards to overlay swapping). If the paging area was to fall in the swappable area all the pages used have to be ensured *CNTR* free which is not possible.

4.1.2 PM overlay 0

PM overlay 0 should have routines used in the foreground. The modules would represent the DECT layers of *Dlc,Nwk,Iwu,Nwkmm,Mac & Eep* involved in the call processing scenario. This overlay should contain the entry points to the above pages which would be loaded if needed and modules that are used across pages.

Common modules like buffer, timer, memfunc used extensively by the foreground activities must reside here. The pagemanager that is reponsible for loading the various pages must also reside here.

All the above mentioned modules have to be ensured *CNTR* free as PM overlay 0 is a swappable overlay.

4.1.3 PM overlay 4

This overlay is the non-swappable overlay as it houses the highest priority interrupt handler the *WakeUp*. Moreover it also handles the *Sp1RcvIsr* that occurs every 125 u sec which comes next to *WakeIsr* in priority. All modules required for the above two handlers are also included in this overlay.

Since this overlay is non-swappable by virtue of the priority of the interrupts it handles this overlay may contain modules with *CNTR'*S. Hence it also contains modules used by *SwIntRcv* handler (which resides in PM overlay 5), viz. *Acquire, Modem, Mac_RcvIsr,Encoder and Decoder* all of which have *CNTR'*S.

4.1.4 PM overlay 5

This overlay contains the SwIntRcv interrupt handler and the TimerIsr interrupt handler. Since both these interrupts can be overridden by the higher priority interrupts this overlay is also swappable and is hence is to be ensured *CNTR* free.

The modules that these handlers use if do posses *CNTR's* must be included in the PM overlay 4 which is the only non-swappable overlay.

The inter-overlay function calls provision provided in the internal PM is to be used for swapping and then calling the functions.

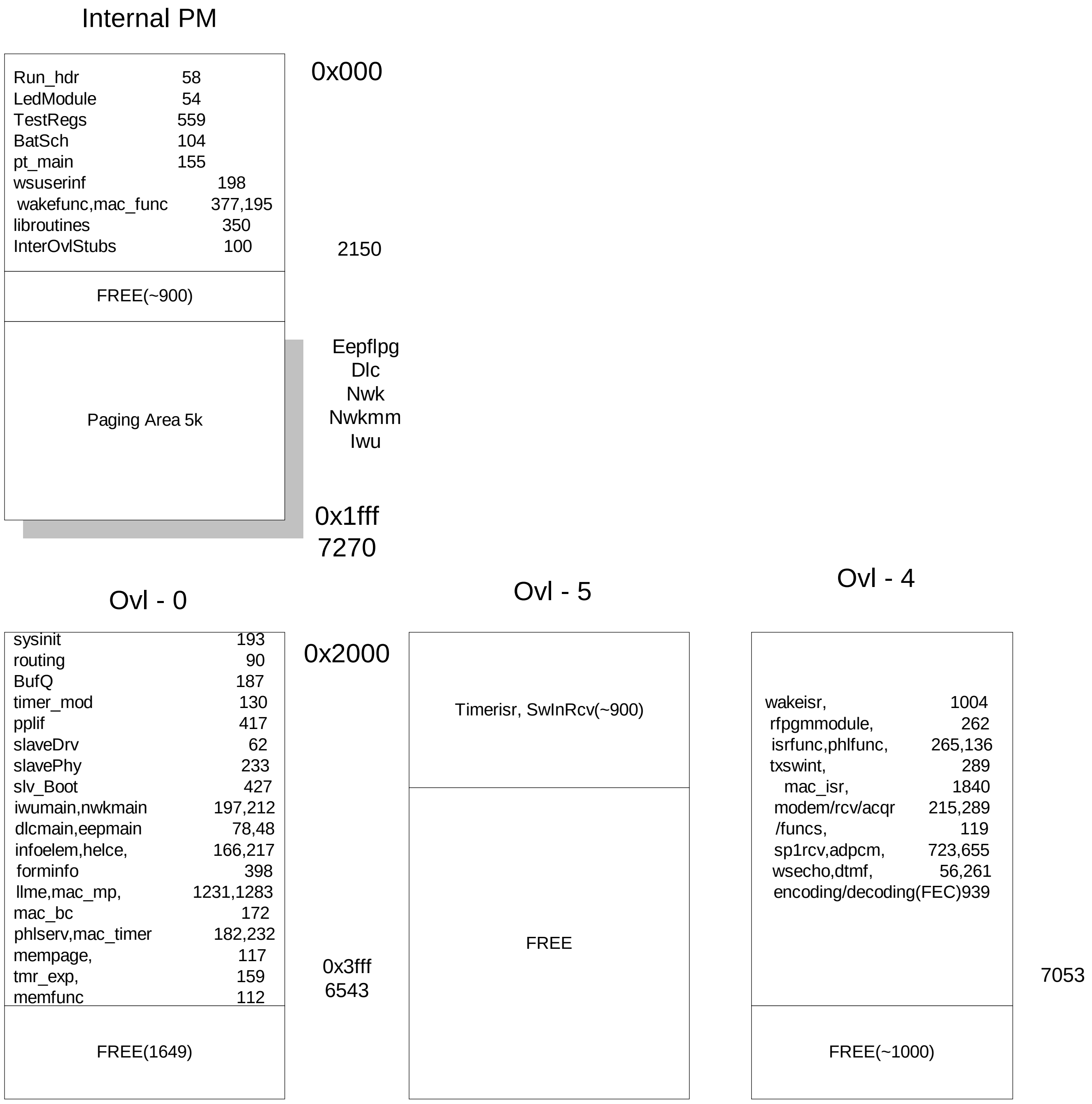
4.2 Data Memory organisation

The DM Block diagram – Sec4.4 indicates the usage of Data Memory by the various modules. The internal DM and DM overlay 0 are used by all the modules with the exception to *SwIntRcv* and *SwIntTx* modules which should use variables in the internal DM or DM overlay 4.

DM overlay 4 essentially contains the initialised DM required for Encoding/Decoding modules for FEC. This initialisation is done on bootup by loading the DM overlay from the external flash. A separate page that contains only this DM initialisation has to be created for this purpose.

The splitter has been modified to create a rxtxdat.exe exclusively for the DM initialization. The **–D0** flag in *spl.dat* file identifies this paged exe whose initialised DM would not be appended to the main dmloader but be retained in the page. This flag is used to create a paged exe to handle only the initialized DM required for RS Encoding/Decoding which is approximately 4k. The paged exe thus created must have only DM declaration and initialization and no additional code as the splitter will not handle them when it encounters the –DO flag. The page thus created is loaded into DM Overlay 4 during bootup in **___***libsetupeverything*.

4.3 PM Memory Block Diagram



Internal PM

0x000

Run_hdr	58
LedModule	54
TestRegs	559
BatSch	104
pt_main	155
wsuserinf	198
wakefunc,mac_func	377,195
libroutines	350
InterOvlStubs	100

2150

FREE(~900)

Paging Area 5k

Eepflpg
Dlc
Nwk
Nwkmm
Iwu

0x1fff
7270

Ovl - 0

0x2000

sysinit	193
routing	90
BufQ	187
timer_mod	130
pplif	417
slaveDrv	62
slavePhy	233
slv_Boot	427
iwumain,nwkmain	197,212
dlcmain,eepmain	78,48
infoelem,helce,	166,217
forminfo	398
llme,mac_mp,	1231,1283
mac_bc	172
phlserv,mac_timer	182,232
mempage,	117
tmr_exp,	159
memfunc	112

FREE(1649)

0x3fff
6543

Ovl - 5

Timerisr, SwlnRcv(~900)

FREE

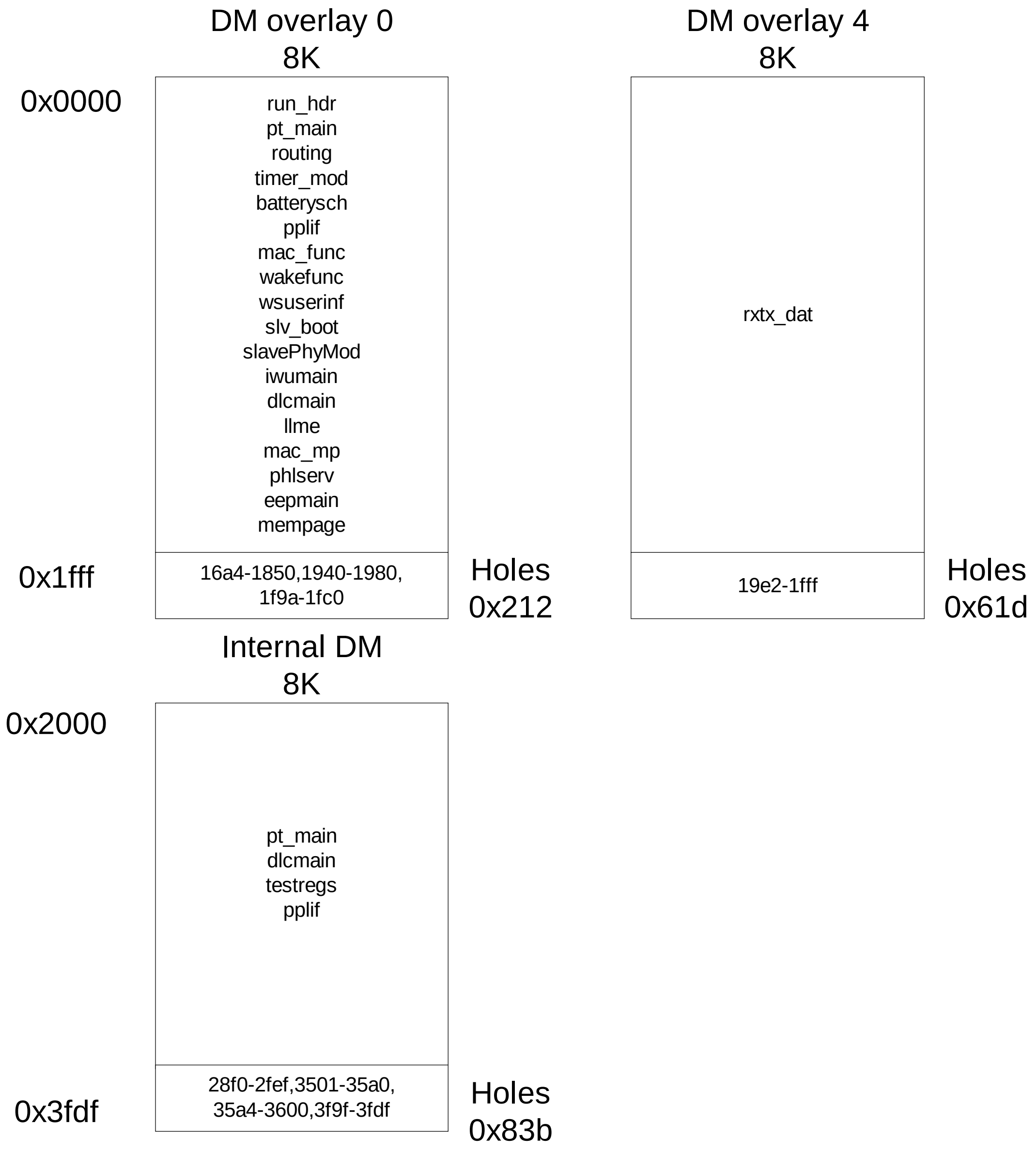
Ovl - 4

wakeisr,	1004
rfpgmmodule,	262
isrfunc,phlfunc,	265,136
txswint,	289
mac_isr,	1840
modem/rcv/acqr	215,289
/funcs,	119
sp1rcv,adpcm,	723,655
wsecho,dtmf,	56,261
encoding/decoding(FEC)939	

FREE(~1000)

7053

4.4 DM Memory Block Diagram



Chapter 5 Implementation Details

5.1 Implementation details for PM Overlay

+ The -lib option in WSIP.grp removed for wsmain.exe. A new file *lib.dsp* created for linking library routines to the internal PM. Note if a new library routine is to be used its appropriate header file must be included in *stdinc/libhdr* dir and the source code of that routine is to be included in the *lib.dsp*.

+ *Flashpgm* moved from overlay 5 to *Eep* page as it is not time critical and is required only when the application code is upgraded.

+ Only essential routines like *runhdr,mac/wakefunc,testregs,pt_main,libroutines,introvlcall,led, batsh,* retained in internal PM using seg = intlow_pm. Other routines are moved to PM overlay 0 using seg = inthigh_pm(dsp files) or –mpm-code = inthigh_pm(c files). Pt.ach defines intlow_pm and inthigh_pm.

+ Modules in PM overlay 0 and PM overlay 5 ensured *CNTR* free (as they are swappable) either by using -mno-doloops or by explicit removal.

+ *Rxswint*, *Timerisr* grouped in PM overlay 5. *ovlstubs.dsp* created for interoverlay calls by *rxswint.dsp*

+ *Wakeisr*, *Sp1RcvIsr, Adpcm, Echo, Dtmf, Modem*-related routines, *mac_isr* grouped in PM overlay 4 . Additionally *mac_isr*, *rfsynth*, *Acquire* and *Modem* also included in this overlay.

Note:

+ All PM circs or PM variables restricted to internalPM.
+ *IMASK* set in run_hdr itself and not in the handlers.

5.2 Implementation Details for DM overlay:

- A new file *rxtxdat.dsp* created to for the initialised DM used by RS Encoding/Decoding. This file is used to create a separate page that is used to initialise the DM overlay 4 on power on.
- Before *SwintTx* is called the DM Overlay is set to 4 and restored after the call. DM Overlay is set to 4 in *run_hdr* before jumping to the *SwintRcv* interrupt handler thus ensuring only internal DM and DM Overlay 4 are available for the above modules which are involved in RS Encoding/Decoding.

<u>Restructuring of the variables used in wsip:</u>

1. *macglob.h* has variables common to SwIntRcv*/SwIntTx isr's* , *Wakeisr* and foreground. Variables declared here are restricted to the internat dm using –mdmdata=segment flag during compiling.
2. A new header file *Phymac.h* has been created to move the variables used by SwIntRcv*/SwIntTx isr's, Wakeisr* and foreground to the internal dm using the above mentioned flag.
3. Variables in other header files like *wakeup.h, system.h, phlsp1.h* used by SwIntRcv*/SwIntTx isr's* given absolute locations in internal DM with comment to retain modularity.
4. Some variables used only by SwIntRcv*/SwIntTx isr's* are now shifted to *rxtxsw.h* which is now declared in *rxtxdat.exe*
5. Files included in *rxtxdat.dsp:*

   Modem.h

   Rxtxsw.h

   Encdec.h

Annex 1 CallTree of ISR's

								CALLTREE OF ISR'S				ANNEX 1
wakeup		sp1rcv	rxswint		timerisr							
								run_hdr				
x		x	x			x		LedModule				
								TestRegs				
									CheckRegsInt_			
									CheckRegsPol_			
x		x	x						Freeze_			
									InitSpy_			
									InitDebugData_			
									CSpy_			
x			x						SetCEnviron			
x			x						BbiDelay			
								BatSch				
									InitBatteryVars_			
x									UpdtBatVal_			
									BatterySchedular_			
									CheckPowerSupply_			
								pplif				
									InitDTMFDecode,AdpcmVarInit,Sp1VarInit_,WsCancellorInit			
									StopDataPort_,StopVoicePort_			
									ConnectVoice_,StartVoicePort_,StartDataPort_			
									StartFeedRingTone_,StopFeedRingTone_			
								pt_main				
									main_			
									ModuleUnderTest_			
								wakefunc				
									InitWakeupVars_			
									ReInitWakeupVars_			
x									InitWakeupLinkTbl			
				x					FrameAdjustTo8_			
									ActivateBearer_			
				x					SetIdleRcvFrame_			
				x					CheckFrameNo			
									CheckSlotStatus_			
	x		x						ChgKeyGenBitInWkFlg			
									ChangeFreq_			
								mac_func				
				x					WrLclMsg_			
									RdLclMsg_			
									PowerOnMacInit_			
									ReInitMac_			
									InitRssi_			
									SortChannels_			
								wsuserinf				
									InitDSPTimer_			
									ServWs10msA_			
									GetKeyFrmSubDialBuf_			
									AlertPtUser_			
									InitVoiceUsrIface_,InitDataUsrIface_			
									IndicateOffHook_,IndicateOnHook_			
						x			ReadLineSts			
						x			RingFeedOn,RingFeedOff,ToggleRingFeed,TurnMetPulseOn/Off			
								sysinit				
									___lib_setup_everything			
								routing				
									InitLMR_			
									ReInitLMR_			
									LMRSchedular_			
									MsgRout_			
								BufQ				
									InitMsgBuf_			
									FreeMsgBuf_			
									SetBufAccess_			
									IsQEmpty_			
									InitQ_			
									RdHeadFromQ_			
									RdFromQ_			
									WrInQ_			
									FreeQ_			

Key: 'x' indicates that the ISR calls a routine in the specified module

wakeup		sp1rcv	rxswint		timerisr			timer_mod				ANNEX 1
									InitTimer_			
									StartTimer_			
									StartShortTimer_			
									StopTimer_			
									TimerSchedular_			
									RdFromTmrQ_			
								slaveDrv				
									SlaveDrvSchedular_;			
									InitSlaveDrv_;			
								slavePhy				
									InitSlave_			
									WrInSlaveQ_			
									SlaveSchedular_			
									SlaveRcvBytePhy			
									SlaveTxBytePhy			
								slv_Boot				
									BootSlaveDsp_			
									ReadSlvThroughPut_			
									SndMsgToSlave_			
									MasCallProcServer			
									MasSlvOnHookHandler_			
								iwumain				
									InitIwuVars			
									IwuSchedular			
									ActOnTrafTimeOut			
									IwuTmrExpHdlr			
								nwkmain				
									InitNwk_		NwkLceTmrExpHdlr_	
									NwkMMSchedular_		SetTpui_	
									NwkSchedular_		GetSubTblAddr_	
									NwkCcTmrExpHdlr_		CopyIpuiOtherTypeToTpui	
									NwkMmTmrExpHdlr_			
								dlcmain				
									PowerOnDlcInit_			
									DlcSchedular_			
									DlcReTxTmrExpHdlr_			
									DlcChovTmrExpHdlr_			
									DlcChovPendTmrExpHdlr_			
									GetFreeLapc_			
								eepmain				
									InitEeprom_			
									EepromSchedular_			
									UpdateFrzData			
									EepPasswdTmrExpHdlr_			
									Routines common bet nwk,nwkmm,iwu pages			
								infoelem	FormMsgUnit, CopyInfoToMsg,ExtractInfoEle,InitInfoEle			
								helce	SendMsgToLce,ReleaseDlcLnk,CompareBits,			
								forminfo				
									FormBasicServ		FormPortableId	
									FormRelReason		FormFixedId	
									FormSingleKeyPad		FormRejectReason	
									FormMultiKeyPad		FormIwuToIwuRAP	
									FormCdPartyNum		FormIwuToIwuUserHdr	
								llme				
									LlmeSchedular		DummyInfoConfirmed	
									SyncProcOver		UpdtMacDummyInfo	
									SetSysLockedState		DummyInfoNOK	
									InitPeriodicRssi		BearerReleased	
									GetFreeTbcChnl		ConnectionReleased	
									SelPhyCh		CheckRelock	
									AnotherBaseFound		TrafficBrEstb	
									AnotherBaseSrchFailed		CheckOtherBase	
									RelockBaseFound		FindStrongChannels	
									RelockBrFailed		RelCurDummy	

Key: ‘x’ indicates that the ISR calls a routine in the specified module

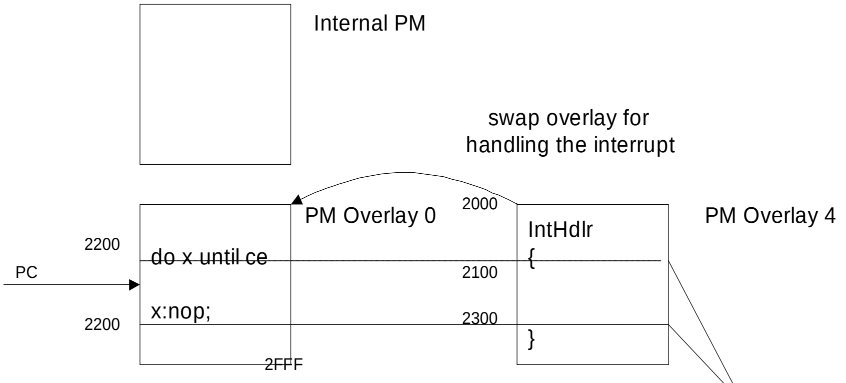
wakeup		sp1rcv	rxswint		timerisr			mac_mp				ANNEX 1
									MacSchedular			
									DlcMacMsgProc			
									LocalMsgProc			
									SendMacPrim			
									FreeTBCState			
								mac_bc				
									ReleaseTBC_			
									ReleaseMBC_			
									InstallDBC_			
									StoreFmidPmid_			
								phlserv				
									SyncProc_			
									BeginSyncProc_			
									InitPlmeSyncState_			
								mac_timer				
									MacConDeferTmrExpHdlr_			
									MacConEstbTmrExpHdlr_			
									MacNoBrHoTmrExpHdlr_			
									MacBrHoDfrTmrExpHdlr_			
									MacMeSyncTmrExpHdlr_			
									MacMeRssiTmrExpHdlr_			
								mempage				
									LoadMemPage_			
									LoadWakeOverlay_			
									LoadLibroutines_			
									LoadDM_			
									LoadTimerIsr_			
								tmr_exp	TmrExpHdlr_			
								memfunc	MemCmp,MemSet,MemMove,MemCpy			
								Timerisr	TimerIsr			
								wakeisr	WakeUp			
								rfpgmmodule				
									InitialiseRF_			
x									PgmRfRcvSynth_; { @ -400 ar = freq 0..9 }			
x									PgmRfRcvBbi_; { @ -200 ar = RcvTask }			
x									PgmRfRcvSlotSigs_; { @ 0 ar = rccr }			
x									PgmRfTxSynth_; { @ -400 ar = freq 0..9 }			
x									PgmRfTxBbi_; { @ -200 }			
x									PgmRfTxSlotSigs_; { @ 0 }			
x									PgmRfEnd_; { @ +400 }			
x			x						RdRssiVal_; { ret: ar = normalised rssi bin value }			
x									RdBatteryVal_;			
								isrfunc				
x									LinkEntries			
x									DelBearer			
x									SlotCorrect			
x									LinkWkLnkEntry			
x									DeLinkEntry			
x									ActivateNextNEntries			
x									DeActivateCurEntry			
x									EnableSport0Rcv			
x									EnableSport0Tx			
								phlfunc				
	x		x						ScrambleData			
	x		x						ComputeACRC_			
	x		x						ComputeBCRC_			
	x		x						CryptData			
x								txswint	SwIntTx			
								rxswint	SwIntRcv			
x			x					mac_isr	MacTxIsr_,MacRcvIsr_			
			x					modemrcv	Modem			
			x					modemacqr	Acquire			
			x					modemfuncs	inter1,inter2,corr,peak_detect,count_1s			
								sp1rcv	sp1rcv			
		x						adpcm	AdpcmEncode,AdpcmDecode			
		x						wsecho	WsCancellorNear,			
		x						dtmf	DetectDTMFCodes			
								enc/decoding				

Key: ‘x’ indicates that the ISR calls a routine in the specified module

Annex 2 Caution in using PM Overlays

Care is to be taken in swapping *PMOVLAYS* during **loop execution**.
The PC (Program Count) generation is by default a linear increment of the address of the currently executing instruction. But during a loop execution a loop compararator determines the last instruction in the loop and checks the termination condition if false the PC is loaded with the first instruction in the loop poped from the PC Stack.

Since the overlays use the common address space, if during a loop execution an overlay is swapped and in the swapped overlay if the PC reaches the loop boundary the loop compararator which has no notion of overlays may still pop the address of the first instruction in the loop if the terminating condition is false. This affects the normal PC increment which ensures linear code execution.



If during the execution of this loop an interrupt occurs and the current overlay is swapped to overlay 4 and the IntHdlr uses the same address space as the loop boundaries , the loop compararator may pop the first instruction in the loop which is not the logical flow for the IntHdlr

The interrupt handler execution and the loop execution address space if not mutually exclusive will create abnormal code execution

Above anomaly can be averted using:

1. Mutually exclusive address space between loop and swapped overlay execution.

or

2. The overlay to be swapped has no loops at all.

Hence the swappable overlay must never contain loops or it must be ensured that the swapped overlay does not use the loop boundary in any case during its execution.