SRM9000 GPIF User Application

"Development Environment Documentation"

Version: 1.00

Date: 5 / March / 2003

Doc Ref: GPIF User Application Environment Iss100.doc

This is an Uncontrolled Copy unless otherwise stated.

Owner: J.Kuhrt

Authors: J.Wright, J.Kuhrt

Date	Name	Signature	Role
5/3/2003	Jens Kuhrt	Jens Kuhrt	Product Manager

Note: The owner should be informed of any errors or inadequacies found in this document. Company or product names mentioned in this document may be trademarks or registered trademarks of their respective companies.

1270 Ferntree Gully Rd,
Scoresby, 3179
Victoria, Australia

CHANGE HISTORY

Version	Date	Change	by
Draft	11/9/2001	Compilation in progress	JW, JK
Draft	30/12/2002	Added more global procs and vars	JW
1.00	5/Mar/2003	First Release to Website	JK

ISSUE HISTORY

Version	Date	What
Draft006	26/9/2001	UK for initial comment
Draft007	30/12/2002	Updated doc and added new examples
1.00	5/March/2003	Formalised Rev100 release

File Name: GPIF User Application Environment Iss100.doc

INCOMPLETENESS RECORD

The following items are not yet covered by this document and remain to be addressed in future versions:

Contents

1 INTRODUCTION

This document provides information to allow software applications to be written for the "User-Application" version of the SRM9000 General-Purpose-InterFace (GPIF) Board.
The MA-DMAP Option is a version of the GPIF Board as described below.

This board :

- fits into the Options-Slot in the SRM9000 Transceiver.
- connects to the Transceiver radio board via a 26-way miniature ribbon cable.
- has a DSP processor that can be programmed in 'C'
- has two async serial ports (RS232 levels on DB25 and 0/5V levels on RJ45 connectors) capable of speeds up to 38.4kbaud.
- has an 8-bit Input/Output port for parallel IO
- has extensive control over the Radio and radio functionality
- may have an optional GPS receiver fitted.
- has option for a dual-CODEC for audio IO and processing.

Refer to SRM9000 Application Notes A9k-505, A9k-811 and A9k-821 for details on general interface capabilities and stand applications.

Application Software can be downloaded into the GPIF Board FLASH via the standard SRM9000 Programmer. The compiled and linked Software (.BIN file) should be copied to the folder:

C:\ Program Files \ Simoco \ SRM9000 \ FPP \ Radio_SW \ GPIF_SW

The Programmer function "Options > Upgrade Software > Getfile > … > Download" can be used to select and download the file into the radio via the Front-RJ45 connector. The radio will automatically put the SW into the GPIF card (if fitted).
Note: This can only be done through the Transceiver Front-RJ Connector, not the RJ connectors on the GPIF board.

This document should be read in conjunction with the examples on the SRM9000 GPIF Developer Disk. These examples show:

- DEMO1 :
  Initialisation of variables, and assigns Ext Serial Port A to 4800B, 1start, 2stop bits.
  Monitors Serial Port A for ASCII Characters
  If 'I' then stops the LED (on SRM9030 Control Head connected to Port B) from being updated.
  If 'P' the makes radio PTT.
  If 'R' then releases radio PTT .
- DEMO2 :
  Sends RSSI and Channel Info to Serial Port A once a second.
- DEMO3 :
  Sends Volume-Request to Radio
  Receives Volume-Report from Radio
  Displays Volume settings on SRM9030 Control Head connected to radio Front RJ
  Restores normal Radio displays after 3 seconds of no-change in Volume.
- DEMO4 :
  Sends a MPT-Status message (Increments every time) to an address at 5 second intervals. If the Status send is successful the Status number is incremented (wrapping around from 30 to 1) and the next Status is sent following a delay of 5 seconds. If the send fails the same Status number is repeated at 15 second intervals until it suceeds.

- DEMO5 :
  Monitor External Parallel IO Inputs (1…4)
  Copy to Outputs (5…8)
  Monitor Serial Port B (4800,N,8,1) and echo any received characters to SRM9030 Control Head connected to Serial Port A.
  * Hi/Lo status of all Parallel IOs
  * Serial Activity on Serial Port
  (This is also useful for confirming that IO is operational)

  This demo shows how to intercept and utilise a control head connected to the GPIF Serial Port. The demo code allows the Radio to actually initialise the control head then takes control of it.
- DEMO6 :
  Monitor External Parallel Input 1 and debounces input when going low.
  Displays result (on SRM9030 Control Head connected to radio Front RJ connector)
  * Status of Input 1 (when low).
- DEMO7 :
  Read GPS data from GPS Receiver and display ‘as received’ data on SRM9030 Control Head connected to Serial Port A of GPIF.
- DEMO8 :

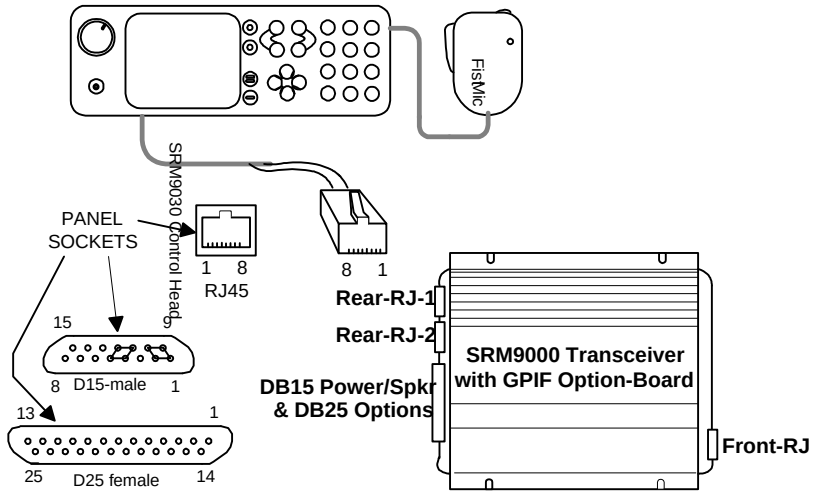
  Monitor GPIF GPS data flag and build into 3 display lines to send to 9030 control head on either serial or Radios's RJ45.

  Only display GPS data if AUX function key is ON.

  This requires that the Radio config has an 'Auxiliary' Function Button defined.

  This demo closely emulates the display of GPS data when activated by *56# in a direct GPS Radio software build.

1.1 Recommended Setup and Connections



Typically the Control Head is connected to the Front-RJ, but several applications are possible with it connected to RJ-1(A), or RJ-2 (B).

All programming (using FPP Lead) should be done via the Front-RJ.

Pin	DB25 Function	DB15 Function
1	GND (Supply)	GND (Supply)
2	RS232 TxD1 *1	GND (Supply)
3	RS232 RxD1 *2	Ignition Sense IP
4	RS232 RTS1	+VE Supply
5	RS232 CTS1	+VE Supply
6	RS232 DSR1 *5	Speaker –ve
7	GND (Supply)	Input#0
8	RS232 DCD1 *5	Output#0
9	Parallel IO #4	GND (Supply)
10	Parallel IO #5	GND (Supply)
11	Parallel IO #7	+VE Supply
12	Analog IN #3	+VE Supply
13	+VE (Switched)	Speaker +ve
14	5V TxD1 *1	Audio Input
15	5V RxD1 *2	Audio Output

Pin	DB25 Function
16	RS232 TxD2 *3
17	RS232 RxD2 *4
18	Parallel IO #1
19	Parallel IO #2
20	RS232 DTR1 *5
21	Parallel IO #3
22	RS232 RING1 *5
23	Parallel IO #6
24	Parallel IO #8
25	Analog OUT #3

Pin	Rear RJ-1 Ftn	Rear RJ-2 Ftn	Front RJ Ftn	Comment
1	Serial-1 TxData	Serial-2 TxData	TxData (Output)	0/5V levels
2	Serial-1 RxData	Serial-2 RxData	RxData (Input)	0/5V Levels
3	On/Off (Input)	On/Off (Input)	On/Off (Input)	Brief Low = toggle
4	Mic Ground	Mic Ground	Mic Ground	
5	Switched +13.8V	Switched +13.8V	Switched +13.8V	250mA max
6	Audio-1 Output	Audio-2 Output	Handset Audio	
7	GROUND	GROUND	GROUND	
8	Audio-1 Input	Audio-2 Input	Mic Audio (Input)	

Notes:
*1 = “5V TxD1”, “RS232 TxD1” and “Serial-1 TxData” lines are all driven together (with same signal).
*2 = “5V RxD1”, “RS232 RxD1” and “Serial-1 RxData” lines have wired-OR function.
*3 = “5V TxD2”, “RS232 TxD2” and “Serial-2 TxData” lines are all driven together (with same signal).
*4 = “5V RxD2”, “RS232 RxD2” and “Serial-2 RxData” lines have wired-OR function.
*5 = Not used or driven

SRM9030 Control Head is recommended, however 9020 and 9025 consoles could also be used (with reduced functionality). The SRM9010 Fist-Mic is NOT supported by the GPIF SW.

1.2 GPIF-Brd Circuit Functional Description

1.2.1 Flashing LED (D207)

The D207 LED is flashed by the SW at a slow rate (ie. 300ms on, 300ms off) while the SW is running normally.

During SW-Download, the LED flashes at higher rate (ie. 100ms on, 100ms off)

If this LED stops flashing, then the GPIF background supervisory loop has stopped and the Watchdog will reboot the GPIF after approx 1 second.

2 GPIF USER APPLICATION : DEVELOPMENT ENVIRONMENT

This document should be read in conjunction with the EXAMPLEs on the SRM9000 GPIF Developer Disk.

Each of the EXAMPLE folders are a complete development directory. Details below refer to these EXAMPLE folders.

The GPIF Processor is an Analog Devices 2185 DSP. All software references are to a 2181 which is functionally equivalent for GPIF SW purposes

2.1 User Files

2.1.1 'C' Software main() function

- The user application environment is presented as a C language function called **main()**.
- The GPIF central code calls main() at a 500uS rate, or twice per millisecond.
- The main() function must be written as a 'fall through' routine taking less than 2500 cycles or instructions.
- Do not execute long loops in 'C' as this will cause GPIF to Radio and serial communications to fail.
- Timers can be implemented by counting the number of times main() is executed.

2.1.2 Variable and Constant definitions

- User Application *variables* should be added to **cvars.dsp** file.
  File contains static variables used by main(). This file may be edited to add new variables. Note that compiled code will be faster and smaller even when non-static variables are allocated in cvars.dsp rather than allocating them in mainc.c.
  Approx 8000 (decimal) words of DM is available. If this is exceeded, then a "not enough dm rom" error is shown at the compile/link stage.
- User Application *constants* may be added near the beginning of mainc.c.

2.2 Other Files

All other files (ie. other than **main.c** and **cvars.dsp**) should not be modified since any GPIF SW updates supplied by Simoco may alter or overwrite them.

The following files should always be included in main().

- **defines.h** = General GPIF FLASH access routines, config access to Radio and Console message subtypes.
- **dtedll.h** = Serial Interface definitions
- **duart.h** = DUART register definitions
- **extio.h** = MASKS for determining External IO functions
- **key.h** = Console Key definitions (eg for SRM9030 Control Head)
- **map27mpt.h** = Defs for MAP27 Trunk Serial IO mesages
- **map27pmr.h** = Defs for PMR Serial IO messages
- **trnkdefs.h** = Defs for Trunk Air-Messages

2.3 Software Development Tools

Required Development tools are: Analog Devices ADSP21xx Tools Release 6.1.

Note that the GPIF SW was developed using the Release-6.1 tools and it currently does not support the requirements of Release-7 tools (which are now available from Analog Devices). Simoco may (or may not) upgrade the SW for compatibility with the Release-7 tools in the future.

These Tools include Assembler, Compiler and Linker.

The **c.bat** file compiles and links the User Application **main.c** and **cvars.dsp** files with the supplied GPIF object files.

2.3.1 Limitations of ADSP21xx Tools Release 6.1

There are a few notable idiosyncrasies of the Release 6.1 tools.

- The library routines are inaccessible as they are defined as /ram. If you wish to use routines that are normally held in the library, either write your own or find the appropriate routine and copy the source code to your project. GPIF modules may not be in /ram.
- Static variables and arrays can not be defined in 'C'. These variables must be defined in a companion .dsp file. The GPIF examples use **cvars.dsp** to hold static variables for the 'C' routines. The reason for this is similar to why the library routines can not be used in that the default location for variables is /ram.
- Do not place any variable name or 'void' in front of the name of a 'C' function. All functions return an integer.
- Treat all variables as integers. It is unknown if structures or unions work.
- There is no need to declare external functions before they are used. Attempting to declare them may produce unnecessary frustration.
- If you remove a variable from cvars.dsp and mainc.c and the linker persists in searching for it, try removing all mainc files except mainc.c then re-compiling. I.e. Remove files mainc.i, mainc.s, mainc.is, mainc.obj, mainc.cde and mainc.int.

2.4 Software Installation

The supplied information consists of:

ADSP21xx Tools	containing setup files for development environment.
\Installation Notes	containing a copy of this document.
\Examples\…	containing separate folders for each GPIF SW example.

To Install the environment perform the following steps:

Step 1. Run the "setup.exe" program to install the ADSP21xx Tools.

Follow the prompted installation steps.

This should install the tools into "C:\ Program Files \ Analog Devices \ VisualDSP \ …" folder.

Manuals are installed as follows:

…\Docs\21xx__at	contains Assembler and Linker doc
…\Docs\21xx__ct	contains 'C' Tools doc
…\Docs\21xx__um	contains User Manual for the 21xx series DSPs
…\Docs\21xx_rtl	contains 'C' Run Time Library Manual
…\Docs\V21k_dbr	contains Debugger doc

Executables are as follows:

…\bld21.exe	System Builder (refer Assembler/Linker doc)
…\asm21.exe	Assembler 'C' Preprocessor (")
…\asmpp.exe	Assembler Preprocessor (")

…\ld21.exe	Linker (")
…\lib21.exe	Librarian (refer Assembler/Linker doc and 'C'tools doc)
…\hspl21.exe	HIP Splitter (refer Assembler/Linker doc)
…\spl21.exe	Prom Splitter (")

Step 2. Copy the Example Folders on the CD to a suitable location on your C: drive (eg. C:\GPIF_Dev)

Step 3. Open a DOS window, go one of the Examples (eg. c:\GPIF_dev\Examples\demo1) and execute the c.bat file.

The software should compile without errors.

The file GPIF.BIN is generated (and copied to the "C:\ Program Files \ Simoco \ SRM9000 \ FPP \ Radio_SW \ GPIF_SW" folder.) This file can be downloaded to the GPIF Card via the SRM9000 Programmer.

Step 4. The Programmer function "Options > Upgrade Software > Getfile > … > Download" can be used to select and download the file into the radio via the Transceiver Front-RJ45 connector. The radio will automatically put the SW into the GPIF card (if fitted).
Note: This can only be done through the Transceiver Front-RJ Connector, not the GPIF RJ connectors.

3 GPIF GLOBAL PROCEDURES AND VARIABLES

These procedures allow the 'C' program to send and receive messages to the radio and/or the Serial / Parallel Ports. Refer to EXAMPLES for usage.

3.1 Communication with the Radio

There are two functions provided for Radio communication, the first is for general use and the second intended for Trunking applications. These functions, documented below, are used to send data messages to the Radio for control purposes.

Information flowing back to the GPIF from the Radio always passes through the Map27 Parser and is available to user applications by intercepting the Parser. See the global flags interceptParser and parserMessageAvailable. For Trunking applications an additional flag and data buffer are available for messages normally destined for a connected DTE – these are the global flag map27MptMessageAvailable and variable array map27OpMessage

3.1.1 int SendPacketToRadio(*packetPointer)

packet[0] = length of following data. Maximum value 240.

packet[1] = Type. Must be 0.

packet[2] = SubType. See SRM9000 Serial Documentation and following GPIF Packet Format documentation.

packet[3..] = depends on Subtype. All data must be bytes (only bits 0 to 7 used).

eg. Request the radio to report its current volume settings.
This is done by sending Command 186 (0xBA) Set Volume (Request Report) message, defined in "SRM9000 Serial Command Definitions" document

```
packet[0] = 3;              /* No of following bytes */
packet[1] = 0;
packet[2] = SET_VOLUME;         /*Set Volume Command from map27pmr.h */
packet[3] = VOLUME_REQUEST; /* Request Volume from map27pmr.h */
if (SendPacketToRadio(packet))
        radioPacketFailed = 1;   /* Not enough room in queue */
```

3.1.2 Map27DteMessageToRadio(*packetPointer)

Note: This function does not utilise a queue. Only one message may be sent per pass through the 'C' mainline.

packet[0] = length in bytes of following data. Maximum value 240.

packet[1] = Message Type. See "Map27 Mobile Access Protocol for MPT 1327 equipment".

packet[2…] = Message data – depends on Message Type.

eg. Request the radio to report its current Control channel and Sys code. This is done by sending Message Type 0xB0, Reason 4 defined in "Map27 Mobile Access Protocol for MPT 1327 equipment".

```
packet[0] = 2;              /* No of following bytes */
packet[1] = RADIO_INTERROGATION;                /* From Map27Mpt.H */
packet[2] = REQUEST_NETWORK_INFORMATION;    /* From Map27Mpt.H */
Map27DteMessageToRadio(packet);
```

3.2 Serial Functions

These procedures should only be used if the relevant serial port has its override flag set. See global flag SerialAConfigOverride and SerialBConfigOverride.

They allow communications with in/out the two Serial Ports.

3.2.1 DuartInitA(baudrate + modeFlags)

3.2.2 DuartInitB(baudrate + modeFlags)

Used to initialise the specified Duart and serial Input/output queues if the user application will be controlling the specified serial port. The relevant procedure MUST be executed once prior to using any of the serial queue functions.

baudrate (bits 0-2) 0 = 300
1 = 600
2 = 1200
3 = 2400
4 = 4800
5 = 9600
6 = 19200
7 = 38400

modeFlags bit 7 = Map27 mode
if clear, then MAP27 or SUP messaging is not required for the port. (ie. Like "serialXConfigOverride").
if set, then port is used for MAP27 or SUP messaging. Serial queue functions (ie. All those described below) may not be used.

bit 6 = clear for 1 stop bit, set for 2 stop bits.

bit 3 = clear for MAP27, set for Simple Unacknowledged Protocol (SUP)

eg: DuartInitA(3+64); /* init for 2400 baud, 2 stop bits */

See also duartAConfig and duartBConfig.

3.2.3 RoomInSerialTxQA()

3.2.4 RoomInSerialTxQB()

Returns the amount of free space in bytes in the specified serial TX queue. This function must be used if either PutCharInSerialTxQx or PlaceDataInSerialTxQx are to be used. The Put/Place functions may only be executed if there is sufficient room for the data to be added. Ie. 1 byte (for PutCharInSerialTxQx) or data length (for PlaceDataInSerialTxQx).

eg:

```
if (RoomInSerialTxQA())
        PutCharInSerialTxQA(key);
```

3.2.5 PutCharInSerialTxQA(character)

3.2.6 PutCharInSerialTxQB(character)

Places 1 byte of data in the specified serial TX queue. The function RoomInSerialTxQx must have been used to check available room.
Queue length is 512 bytes, when empty..

eg. For outputting 1 byte at a time to an async serial port.

3.2.7 PlaceDataInSerialTxQA(length, *dataPointer)

3.2.8 PlaceDataInSerialTxQB(length, *dataPointer)

Place the data pointed to by dataPointer in the TX queue for length bytes. The function RoomInSerialTxQx must have been used to check available room.

eg. For outputting a string to an async serial port

3.2.9 CharactersInSerialRxQA()

3.2.10 CharactersInSerialRxQB()

Returns the number of bytes available (ie. size of message(s)) in the specified RX queue. Used in conjunction with GetDataFromSerialRxQx.

Max queue length is 64 bytes.

3.2.11 GetCharFromSerialRxQA()

3.2.12 GetCharFromSerialRxQB()

Returns a negative value if there are no bytes in the RX queue or a positive value (the extracted byte) if there was data available. There is no need to call CharactersInSerialRxQx before using this function.

3.2.13 GetDataFromSerialRxQA(length, *dataPointer)

3.2.14 GetDataFromSerialRxQB(length, *dataPointer)

The function, CharactersInSerialRxQx must be used to determine the number of bytes to extract from the RX queue before using this function. Extract length bytes from the serial RX queue into the data buffer pointed to by dataPointer. If length exceeds the number of bytes available, the serial queue pointers will be corrupted, so it's best to ensure that the buffer being written to is at least 64 bytes long.

eg.

```
length = CharactersInSerialRxQA();
if (length >= 0)                    /* only if there is something in the que */
        GetDataFromSerialRxQA(length, &myBuffer);
```

3.2.15 RoomInMap27DataQueueA()

3.2.16 RoomInMap27DataQueueB()

Only for use when Map27 is active on the relevant serial port. Returns the amount of free space in bytes in the specified Map27 TX queue. This function must be used if SendMap27DataPacket is used and may only be executed if there is sufficient room for the data to be added.

eg:

```
if (RoomInMap27DataQueueA())
        SendMap27DataPacketA(myRadioPacket);
```

3.2.17 SendMap27DataPacketA(*packetPointer)

3.2.18 SendMap27DataPacketB(*packetPointer)

Only for use when Map27 is active on the relevant serial port. Places the data packet pointed to by pointer in the specified queue.

Packet has the same format as given for SendPacketToRadio.

3.3 Audio Path Control Functions

These functions allow control of the various audio paths in the GPIF.

Note that these functions are documented for the standard DMAP version of the GPIF – no CODEC fitted.

3.3.1 ClearTxAudioPathForLocal()

Selects U406 multiplexer 0.

Stop TX audio from Line A or Line B.

Set RX audio to both Lines A and Line B.

This is the default setting in receive mode.

3.3.2 SetTxAudioPathFromLineA()

Selects U406 multiplexer 1.

Local TX audio from Line A, local RX audio to Line A.

3.3.3 SetTxAudioPathFromLineB()

Selects U406 multiplexer 2.

Local TX audio from Line B, local RX audio to Line B.

3.3.4 UserSetRxAudioPath(pathSelect)

This function controls U406 but allows any multiplexer selection.

pathSelect = 0x0C	Same as SetTxAudioPathFromLineA()
pathSelect = 0x00	Same as SetTxAudioPathFromLineB()
pathSelect = 0x04	Selects multiplexer 3. TX/RX audio from/to DB25.
pathSelect = 0x08	Same as ClearTxAudioPathForLocal()

3.3.5 SetTxAudioPathLineAToLineB()

Closes U403 and opens U413.

Routes Line A RX audio to Line B TX audo.

3.3.6 SetTxAudioPathLineBToLineA()

Closes U413 and opens U403.

Routes Line B RX audio to Line A TX audio.

3.3.7 ClearLineToLineTxAfRoute()

Opens U403 and U413. Clears line to line TX audio path.

The above functions are typically used as follows:

To route audio from the serial port A RJ45 connector to the local Radio's TX audio

```
ClearLineToLineTxAfRoute();
SetTxAudioPathFromLineB();
```

To route audio from the serial port B RJ45 connector to the local Radio's TX audio

```
ClearLineToLineTxAfRoute();
SetTxAudioPathFromLineB();
```

To route audio from a control head or TX audio source on the serial port A RJ45 connector to a transmitting Radio on the Serial port B RJ45 connector.

```
SetTxAudioPathLineAToLineB();
UserSetRxAudioPath(4);          /* prevent mix, set local RX to line O/P */
```

To route audio from a control head or TX audio source on the serial port B RJ45 connector to a transmitting Radio on the Serial port A RJ45 connector.

```
SetTxAudioPathLineBToLineA();
UserSetRxAudioPath(4);          /* prevent mix, set local RX to line O/P */
```

Note that a control head (or external) PTT may disrupt these settings which will require the audio settings to be reinstated when the disruptive PTT is released.

3.4 Flash Access Functions

3.4.1 PrimeContextSave

This function causes the 32 word array contextVars[32] to be saved in Flash memory 2 seconds after the function is called. Because there is a delay, the function may be called many times within a short interval and the Flash save will occur when finally 2 seconds has elapsed. (The array contextVars is restored from Flash at power up.)

See contextVars.

Note that Flash save is prohibited in the first 10 seconds following power up. Calls to PrimeContextSave in this time will result in a Flash save after the 10 second interval has elapsed.

3.5 GPIF Global Variables / Flags

These variables / flags can be read by the 'C' program to monitor radio/GPIF state, and (in many cases) written to alter the radio/GPIF functionality.

Refer to EXAMPLES for usage.

3.5.1 startupComplete

Set when the GPIF has successfully contacted the Radio and has a copy of the Serial and External IO configuration. The user application should not attempt any Radio related operations until this flag is set.

3.5.2 serialAConfigOverride

Set to 0 upon GPIF initialisation.

The user application should set this flag to 1 the first time main() is called if control of Serial Port A initialisation is required. If set to 1, the GPIF will NOT initialise the Duart or serial queues. If user applications set this flag then they MUST initialise the Duart themselves (call to DuartInitA(baudRate+serialMode) before using Serial IO.

Note that the FPP configuration data rate and mode are still available in the flags duartAConfig and duartBConfig.

If this flag is not set by the user application, then the GPIF will initialise the port for Map27 operation at the data rate defined in the Radio's configuration.

3.5.3 serialBConfigOverride

As for SerialAConfigOverride but applies to Serial Port B.

3.5.4 parallelInputOverride

Set to 0 upon GPIF initialisation.

If the user's application is to be in charge of any of the 8 bit parallel inputs, the user application should set this flag to 1 the first time the function main() is called. If set to 1 then the GPIF will not apply the FPP configuration rules and responses to the parallel input lines. The 8 bits of parallel input are available in the global variable currentExtInputs. Parallel input can only be read if one of the following conditions apply:

- the flag parallelOutputOverride is set and the corresponding bit in userParallelData is clear.
- the relevant bit is set to an Input in the Radio Input/Output configuration. (Input 1 is bit 0.)

If this flag is not set by the user application, then the GPIF will regularly interrogate the parallel port and action the inputs as they are defined in the Radio's configuration.

If the user application wishes to handle some inputs and leave others to the GPIF then the override flag may be left clear and the inputs to be handled by the user application may be set to 'Undefined' in the FPP. The relevant bits in currentExtInputs can then be polled by the user application.

3.5.5 currentExtInputs

See parallelInputOverride above.

3.5.6 parallelOutputOverride

Set to 0 upon GPIF initialisation.

If control of the 8 parallel outputs is required from the user's application, the user should set this flag to 1 at the beginning of the first call to the main() C function. When set to 1 then the GPIF will not apply the Radio configuration rules to the parallel output lines. Parallel output may be controlled by setting the global variable userParallelData and is updated within 125uS from being changed.

3.5.7 userParallelData

See parallelOutputOverride above.

This variable may also be read to determine the state of various bits of the parallel output. If the user application is not in control of the parallel output then this global may be used to monitor outputs that have been set up in the Radios Input/Output configuration. See also externalOutputFlags.

3.5.8 externalOutputFlags

This variable is used to monitor some functions of the Radio and is directly updated by the Radio each time any bit changes. See extio.h for bit definitions (Trunk and PMR have different definitions).

3.5.9 interceptParser

(See description of Parser function below)
Set to 0 upon GPIF initialisation.

The user application should set this flag to 1 as soon as it begins execution if it wishes to be informed of any packet, either from the Radio or Serial port, that will be routed through the Map27 Parser. Intercepting the Parser is the only method of obtaining long data information (as against the externalOutputFlags) from the Radio. See also "PMR DTE packets".

3.5.10 parseMessageAvailable

(See description of Parser function below)
All Map27 messages from the Serial Ports (and messages from the Radio) are routed through a common point in the GPIF called the Map27 Parser. This flag is set when a message (in rxDtePacket) has been routed through the Map27 Parser. The user application has one opportunity to examine, change or suppress packets. If the user application does not do anything then the packet will be parsed resulting in an action in the GPIF or the packet being forwarded to a serial port or the Radio.

The user application can clear the flag parseMessageAvailable to prevent the packet from being parsed. The parseMessageAvailable flag is set when a packet is parsed if the flag interceptParser (see previous) is set by the user application.

3.5.11 duartAConfig

A copy of the FPP configuration information for serial Port A. This variable is provided so the serial port can be overridden by 'C' while still having access to the intended data rate for the port.

Note that only the bottom three bits should be used for data rate.

Bit 3 indicates that the Serial Unacknowledged Protocol (SUP) was selected in the configuration. If bit 3 is clear then MAP27 is selected in the configuration.

Bit 7 will be set.

3.5.12 duartBConfig

As for duartAConfig but applies to port B.

3.5.13 PortALoggedDevice

This variable indicates the type of device that most-recently logged into serial port A.

Values can be found in defines.h and are:

DTE = 0, Control Head = 1, PMR Radio = 2, Trunk Radio = 3.

3.5.14 PortBLoggedDevice

As for PortALoggedDevice but applies to port B.

3.5.15 oneHeadTwoRadiosMode

Set to 1 when the GPIF has detected a Radio plugged into either serial port A or B.

3.5.16 MostRecentConsoleType

Reflects the most-recent control head to log into either serial port on the GPIF.

Values are the same as the Console type in the serial command Set Console.

3.5.17 headToLocalRadio

Set to 1 when the Control Head is switched to the local Radio. The value is 0 when the control head is switched to a remote Radio (plugged into a serial port on the GPIF).

3.5.18 repeaterModeActive

Set to 1 if the internal repeater function of the GPIF have been activated.

3.5.19 lastKnownVolume

Reflects the most recent 9030 control head volume of a control head plugged into a serial port on the GPIF.

3.5.20 rxAudioPower

An indicator of the level of received audio of the local Radio. This is a positive integer that may be used for detecting audio activity on received signals. This system was originally added to trigger transmit on a PMR Radio in a Trunk to PMR repeater application.

3.5.21 RandomData

This variable may be used as a seed for a Pseudo random number generator or as a limited source of random data. The value is usually in the range –100 to +100 but may be any signed integer value.

3.5.22 GpsSerialDataOverride

Set to 0 upon GPIF initialisation.

This flag should be set to 1 if the user application wishes to stop the core GPIF routines from interpreting serial data from a GPS receiver installed on the GPIF board and updating the Radio. Regardless of the state of this flag the user application can interrogate the variable GpsSerialCharReceived for each serial character. See following.

3.5.23 GpsSerialCharReceived

Bits 8 to 15 of this variable are set each time the core GPIF routines have an 8 bit character ready for interpretation. The bottom 8 bits indicate the byte received from the installed GPS receiver. Bit 15 of this variable is used by the GPIF core routines. Bit 14 may be used by User 'C' applications and should be cleared when the variable is read (do not clear the entire byte, just bit 14). The GPS data in the low 8 bits is typically an NMEA string of ASCII characters in GGA format. Since the GPS data is presented at 4800 baud, the 'C' user application has ample time to read each byte made available. The flag GpsSerialDataOverride does not have to be set to read serial GPS data.

3.5.24 GpsReportReceived

This flag is set to 1 each time a complete GPS report is interpreted and packed into on-air format by the core GPIF routines. The flag indicates only that GPS data has been received, not whether the GPS data is valid. See GpsHeldTime following. This flag will not be set if the flag GpsSerialDataOverride is set.

3.5.25 GPS Held Data*

Note: The above is NOT a global variable name – see following.

12 Bytes of GPS data that has been packed into on-air format by the core GPIF routines.

This data has the same format as described in the document "9000_Ser_Cmds_Defs_V116.doc", the message "Data Message Received" subheading "GPS Data Format".

Only the bottom 8 bits of each word is used.

The GPS data is accessed using the following variable names:

```
GpsHeldTime[3]          /* 6 BCD digits - hours, minutes, seconds UTC */
GpsHeldLatitude[4]      /* 8 BCD digits – first digit always '0' */
GpsHeldLongitude[4]     /* 8 BCD digits */
GpsHeldFlags            /* see 9000_Ser_Cmds_Defs_V116.doc */
```

In addition to the GPS data, there is a single 16 bit word that indicates the number of seconds that have elapsed (maximum 32767) since the last valid GPS coordinates were received.

```
GpsPositionTimer        /* this one is 16 bits and is not accessible 'on-air' */
GpsNumberOfSatellites /* 2 BCD digits  - also not used 'on-air' */
```

3.5.26 contextVars{32}

This array consists of 32 words reserved within the GPIFs Flash context record for user 'C' applications.

Once a value is placed in any of the 32 words you should call the function PrimeContextSave().

This data will be restored at power up, prior to the first execution of the 'C' application.

The initialisation value of all the elements of contextVars is 0.

It is prudent to mark your data in some way that identifies your application so that false startup information is avoided.

Eg.
```
if (contextVars[1] == 0x4654)     /* My application ID  = 'FT' */
{
        feedingTime = contextvars[0];
}
else
{
        contextVars[1] = 0x4654;
        feedingTime = DEFUALT_FEEDING_TIME;
        contextVars[0] = feedingTime;
        primeContextSave();     /* don't really need this here */
}
```

3.5.27 contextFlags

Currently holds 2 bits flags for the core GPIF routines.

Bit 0 Set if the control head was switched to the local Radio when the Radio was turned off.

Bit 1 Set if the standard GPIF repeater mode was active when the Radio was turned off.

Note: In applications where you do not want the control head to be switched or the internal repeater to become active you should override this variable with the value 0x0001 at the beginning of your application.

3.5.28 contextLocalVolume

The last known value sent by a 9030 control head to the local Radio. This variable may be used to restore the volume on the local Radio following setting the volume for repeater applications.

3.5.29 contextRemoteVolume

The last known value sent by a 9030 control head to a remote Radio (connected to a serial port of the GPIF). This variable may be used to restore the volume on the remote Radio following setting the volume for repeater applications.

3.5.30 rxDtePacket[256]

Array of bytes. This buffer contains the packet that has been presented to the Map27 parser. The first 5 elements of this buffer are not documented in the serial specification for the SRM9000 and are specific to the GPIF. Messages in this buffer consist of bytes. i.e. Each (16-bit) word uses only bits 0 to 7.

If data is placed in this buffer then the top 8 bits of each word in the buffer MUST be clear.

rxDtePacket[0] = length of packet. Data length beginning at the TYPE field is calculated as rxDtePacket[0]-4.

rxDtePacket[1] = source of packet. 0 = Radio, 1 = Serial port A, 2 = Serial port B

rxDtePacket[4] = C parsed flag. Set to 1 means that the packet was intercepted by C and will be parsed when C exits provided that parseMessageAvailable is set to 1.

rxDtePacket[5] = Type field as per “SRM9000 Serial Command Definitions” document.

rxDtePacket[6..] = SubType and remainder of packet as per SRM9000 serial documents and/or GPIF Packet Format section in this document.

eg. In response to a Volume-Request Command (186 = 0xBA) the radio will report its current volume settings by sending a Volume-Report message (156 = 0x9C) as defined in the document “SRM9000 Serial Command Definitions”.

```
if ((ParseMessageAvailable)
  &&(rxDtePacket[SUBTYPE] == VOLUME_REPORT))
{
        currentVolume = rxDtePacket[SUBTYPE+1];
        currentAlertOffset = rxDtePacket[SUBTYPE+2];
        currentDataVolume = rxDtePacket[SUBTYPE+3];
}

where   SUBTYPE = 6;                    /*from Dtedll.H */
        VOLUME_REPORT = 156;            /* from Map27Pmr.H */
```

3.5.31 trunkFlag

Set (once startupComplete is set) if the GPIF is installed in a Trunk radio or is a Dual mode radio in Trunking mode.

3.5.32 map27MptMessageAvailable

Trunking only. Set if the GPIF core routines have decoded a Radio message into a message for interpretation as a Map27 DTE message. See following buffer map27OpMessage. There is no need to clear the flag map27MptMessageAvailable – it is cleared following every call to the ‘C’ mainline.

3.5.33 map27OpMessage

Trunking only. The data in this buffer is valid if the flag map27MptMessageAvailable is set. The buffer contains the same data that would be sent to a Trunking Map27 DTE with the addition of a preceding length byte. i.e. The Message Type field is in the second byte of the buffer. For message details see “Map27 Mobile Access Protocol for MPT 1327 equipment”.

eg. In response to a RADIO INTERROGATION, Reason 4 (Network Information), the GPIF core software will set the flag map27MptMessageAvailable and fill out map27OpMessage with the appropriate data.

```
if ((map27MptMessageAvailable)
  && (map27OpMessage[1] == NETWORK_INFORMATION))
{
    currentIContrChan = (map27OpMessage[2]  << 8) | map27OpMessage[3];
    currentISysCode = (map27OpMessage[4] << 8) | map27OpMessage[5];
}

where   NETWORK_INFORMATION = 0xB5;     /* from Map27Mpt.H */
```

3.5.34 ChannelType

Trunking only. Set to 1 when the Radio is on a Traffic Channel.

3.5.35 IndicationStatus

Trunking only. See trnkdefs.h for values. The main use of this variable for ‘C’ applications is bit 15 which indicates when the Radio is In Service.

3.5.36 BusyWithCall

Trunking only. See trnkdefs.h for values. Set to non-zero value when the Radio is performing call setup or busy with on-air functions. Note that BusyWithCall is normally zero while the Radio is in a call (on Traffic Channel).

3.6 The MAP27 Parser Function

The MAP27 Parser is responsible for acting upon and/or redirecting all messages that pass through the GPIF board regardless of their source, serial port or Radio.

To allow user applications in the GPIF to examine and selectively discard messages, the Parser does not immediately action a message if the flag interceptParser is set. Instead, it sets the flag parseMessageAvailable and performs one execution of the ‘C’ mainline. If parseMessageAvailable is still set on return from the call to ‘C’, the message is parsed.

All messages that are sent through the Parser are stored in a buffer called rxDtePacket.

A Serial Port that has been “overridden” does not go through the Parser.

3.7 Trunk Radio Configuration Parameters

These parameters are read from the Transceiver at GPIF power-up, and as required during normal operation. They provide the User Application with information about how the Radio has been programmed.

MPT1343 to MPT1327 numbering conversions.

2-digit Fleet:
eg. Individual Address : MPT1343: 201 / 5743 / 23 = MPT1327: PFIX=1 , IDENT=7489
ie. = (201-200) / (5743-2000)*2+(23-20)
Individual-Base-Ident = (5743-2000)*2 = 7486

eg. Group Address : MPT1343: 201 / 5759 / 91 = MPT1327: PFIX=1 , IDENT=7519
ie. = (201-200) / (5759-2000)*2+(91-90)
Group-Base-Ident = (5759-2000)*2 = 7518

3-digit Fleet:
eg. Individual Address : MPT1343: 227 / 2001 / 223 = MPT1327: PFIX=27 , IDENT=25
ie. = (227-200) / (2001-2000)*2+(223-200)
Individual-Base-Ident = (2001-2000)*2 = 2

eg. Group Address : MPT1343: 227 / 2451 / 955 = MPT1327: PFIX=27 / IDENT=957
ie. = (227-200) / (2451-2000)*2+(955-900)
Group-Base-Ident = (2451-2000)*2 = 902

3.7.1 OwnPfix_

MPT1327 PFIX for current selected Trunk-Network. If Bit-7 is set the PFIX (in bits 0..6) is valid.
eg. MPT1343: 201 / 5743 / 23 = MPT1327: 1 / 7489

3.7.2 OwnIdent_

MPT1327 IDENT for current selected Trunk-Network..

3.7.3 OwnFlags_

Bit 14 of Flags set indicates 3-digit fleet.
Bit 13 of Flags set indicates Group (not used here but relevant for following variables).

3.7.4 IndBaseIdent_

MPT1327 Base Ident for the FleetIDENT for current selected Trunk-Network.

3.7.5 GroupAddresses_[32]

Contains 8 Group addresses (for receiving Group-Calls) of 4 words each (format same as OwnPfix_, OwnIdent_, OwnFlags_, IndBaseIdent_)

3.7.6 DispatcherAddress_[4]

Contains Dispatcher Identity address (format same as OwnPfix_, OwnIdent_, OwnFlags_, IndBaseIdent_)

3.7.7 DataAddress_[4]

Contains Data Identity address (format same as OwnPfix_, OwnIdent_, OwnFlags_, IndBaseIdent_)

3.7.8 HighestOwnFleetIndAddr_

Contains the Highest Unit Number allowed to call in own fleet.

3.7.9 HighestOwnFleetGroupAddr_

Contains the Highest Group Number allowed to call in own fleet.

3.7.10 GroupBaseIdent_

MPT1327 Base Ident for the Group (for making a Group-Call).

3.7.11 PrimeEmergencyNumber_[8]

BCD packed version of Emergency Identity (with ‘B’ for ‘*’, and ‘F’s for unused)
eg *9*2012001223 becomes B9B2, 0120, 0122, 3FFF

3.7.12 CallTypeFlags

Various bits in CallTypeFlags are used by the Radio to check the validity of making some type of calls. The relevant bits are set using the FPP.

See trnkdefs.h for CallTypeFlags bit definitions.

3.7.13 ConfigurationData

Various bits in ConfigurationData are used by the Radio to check the validity of accepting or making some type of calls. The relevant bits are set using the FPP

See trnkdefs.h for ConfigurationData bit definitions.

3.7.14 DiversionFlags

Various bits in DiversionFlags are used by the Radio to check the validity of making a diversion call or following diversion information. The first 2 masks defines whether the Radio is able to perform diversions. The last 4 masks dictate if and how the Radio will follow diversions.

See trnkdefs.h for DiversionFlags bit definitions.

3.8 GPIF Packet Format

This section contains GPIF-Card specific packets that are not described in the “SRM9000 Serial Command Definitions” document.

All packets that may be sent from a DTE or Control Head may be initiated by the GPIF. All packets in the following descriptions have byte 0 = 0 and byte 1 = Subtype. Note that GPIF specific packets and DTE PMR packets do not contain the Block and Count fields present in console (and some other) packets.

Global variables described in packets sent by the Radio only become valid AFTER the packet is allowed to be parsed. It is recommended that the user application set a flag indicating to itself that the relevant packet has arrived, exit, then examine the variables described for the packet upon re-entry (and reset the relevant flag). Alternatively the user application may make a copy of the packet data for its own use then either allow the packet or suppress it. Unless suppressing the packet is required it is generally recommended that the packet is left to be parsed.

All global variables are constructed using the first byte as the high 8 bits. Generally global variables contain 16 valid bits of data. Modem data packets are an exception.

3.8.1 PMR DTE packets

To enable PMR only packets to be sent by the Radio back to the GPIF, the DTE login packet must be sent to the Radio in the early stages of GPIF operation. The packet should be sent only once when the GPIF has contacted the Radio (startupComplete = 1). This packet is the same as used to login as a DTE to the PMR Radio – subtype 181, Set Console Type with console type set to 6. See “APPENDIX C. Usage Examples for Conventional-PMR Applications” in the document “SRM9000 Serial Command Definitions”.

Once the Radio has been informed that a DTE type device is present in the GPIF, packets destined for DTE devices (as described in the document “Serial Control Interface for Trunking MAP27 Applications and Conventional-PMR Applications”) will be sent to the GPIF.
User applications may then intercept these packets to obtain detailed information on Radio operation.

[End of Document]