Underwater Acoustics Acquisition System Architecture and Design

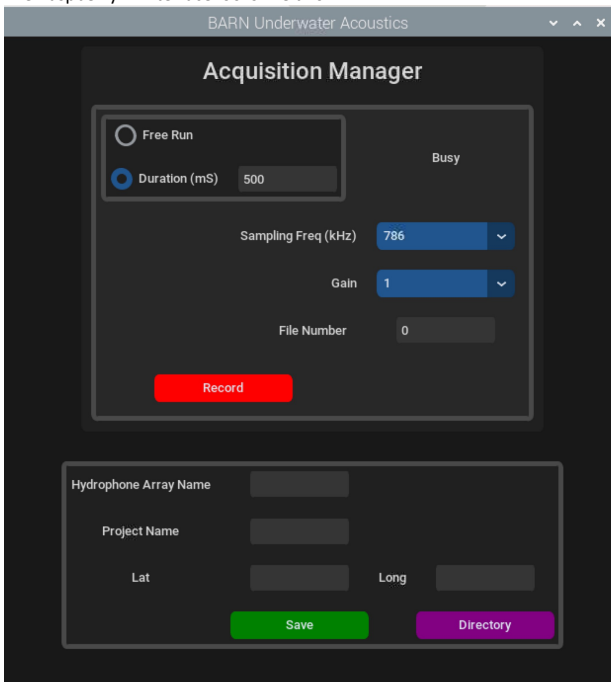
Sunday, October 1, 2023 1:38 PM

The Underwater Acoustics Acquisition System is based on an STM32H7x board that is responsible for the hydrophone analog signal acquisition and storing the recorded values on an SD card on the STM board.
The STM board is under the control of a Raspberry Pi that has a GUI which the operator uses to control the acquisition

The Raspberry Pi communicates commands to the STM32 using an SPI interface. The STM32 executes the requested command and, if required, returns data to the Raspberry Pi.

The STM32 board also has a serial port through which diagnostics messages are sent during normal operation

The Raspberry Pi interface looks like this



The user defines the parameters for the upcoming recording including duration, sampling frequency and gain.
At present, the Free Run option is not supported
At present, the only Sampling frequency supported is 786kHz
At present, the only gain supported is 1
The user can optionally set a File Number in the range 1 - 9999. If zero is selected, then the acquisition system assumes that the "next file number" should be used i.e. If the last file was numbered 3402, then the next file will be 3403.
When files are stored on the SD card, all files are stored in the root directory. File names are of the form FILEnnnn.DAT where nnnn is the File Number.
At present the only method to format the SD card and/or to restart the file numbering is to remove the SD card from the STM board and to makes changes when the SD card is mounted on a PC.

When initially started, the STM32 mounts the SD and scans the SD card to detect the highest file number being used.
If the SD card cannot be mounted, a error message is sent to the STM32 diagnostics port

When the **Record** button is pressed, all parameters for the recording are sent from the Raspberry Pi to the STM32.

The Raspberry Pi also sends the latest date/time to the STM32 which the STM32 uses to set it's internal real time clock
The STM sets up the ADC, DMA and initiates data collection.
ADC data is captured using the 16 bit ADC on the STM32.
ADC data is captured continually with the completion of each ADC conversion initiating the next ADC conversion
Captured data is stored in memory that is organized as 16k samples. The memory is further subdivided into two halves (an upper and lower buffer). Initially, data is saved to the lower buffer and when the buffer is full, further data is saved to the upper buffer. At this time, the STM32 initiates the use of DMA to transfer the contents of the lower buffer to the SD card. This flip flop process continues until the end of the Duration.
At the end of the duration the STM32 closes the file and increments the "highest file number" counter
The STM32 also sets the date/time on the file

When the **Save** button is pressed, the Raspberry Pi sends the File Number to the STM32. If the File Number is zero the STM uses the "last recorded file"
The STM32 returns the number of bytes in the file
The Raspberry Pi then requests each byte in turn from the STM32
The DAT file format on the STM32 SD card is essentially a standard WAV file format with the exception that the analog values are stored as 16 bit unsigned integers. When the data is sent from the STM32 to the Raspberry Pi, the STM32 subtracts 0x8000 from the analog values to provide a file that has data values stored as 16 bit signed integers.
The Raspberry Pi opens a file in the file system (currently a fixed file name but this need to be changed)
The Raspberry Pi is the Master for the SPI interface and to ensure it does not outpace the STM32, a BUSY handshake line is implemented such that the STM32 will tell the Raspberry Pi when it is busy and cannot accept a further SPI request. The Raspberry Pi only sends data requests when the STM32 is not busy. The BUSY line is implemented on PF3-D10 on the STM32. The BUSY line connects to GPIO 27 on the Raspberry Pi

Under normal operation the STM is waiting from commands from the Raspberry Pi. When the Raspberry Pi sends a command, it first takes the Chip Select low (SPI_NCS). The SPI_NCS line is implemented on the STM32 with PF13-A5. The NCS connects to SPI_0 _CE0_N / GPIO8 on the Raspberry Pi

Analog signals received by the STM32 ADC must be between 0.5v and 3.0v

The Directory button initiates a directory scan on the STM32. The directory is printed on the STM32 diagnostic port

Pinouts

STM32	**STM Design**	**Use**	**Raspberry Pi**	**USB Diagnostic Serial**	**STM Debugger**
SWCLK	J6 - Pin 2	SWCLK			SWCLK
SWDIO	J6 - Pin 3	SWDIO			SWDIO
GND	J6 - Pin 4	GND			GND
RST	J6 - Pin 5	RESET			RST
3.3v	J6 - Pin 6	3.3V			3.3v
GND	J6 - Pin 8	Digital Ground	GND - Pin 6		
PF11 - A1	J6 - Pin 9	Analog Input			
PF13 - A5	J6 - Pin 13	SPI6_NCS	SPI_0 _CE0_N / GPIO8 - Pin 24		
PG13 - SPI SCK	J6 - Pin 14	SPI SCK	SPI_0_SCLK / GPIO11 - Pin 23		
PB5 - SPI MOSI	J6 - Pin 15	SPI MOSI	SPI_0_MOSI / GPIO10 - Pin 19		
PB4 - SPI MISO	J6 - Pin 16	SPI MISO	SPI_0 MISO / GPIO09 - Pin 21		
PD9 - USART3_RX	J6 - Pin 17	RX - Diag Serial		TXD	
PD8 - USART3_TX	J6 - Pin 18	TX - Diag Serial		RXD	
GND	J6 - Pin 19	Gnd - Diag GND		GND	
Analog Input Gnd	J3 - Pin 2	Analog Ground			
PE10 - D11	J3 - Pin 6	Sync (used for ADC testing)			
PF3 - D10	J3 - Pin 7	Acq Busy	GPIO 27 - Pin 13		
PA0 - D9	J3 - Pin 8	SD Card Detect			
PC13		Red Onboard LED 0			

Command Structure between Raspberry Pi and STM32

Command	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Follow Up		Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Status	0	0	0	0	0	0	0	0										
Record	1	Sampling freq msb	Sampling freq lsb	Gain	Duration msb	Duration lsb	Filenumber msb	Filenumber lsb										
Stop	2	0	0	0	0	0	0	0										
DateTime	3	Hour	Minute	Second	0	Month	Day	Year - 2000										
Location	4	0	0	0	0	0	0	0	Lat and Lng									
Save	5	Filenumber msb	Filenumber lsb	0	0	0	0	0	File Data		File size msb bytes	File size lsb bytes						