D-STAR INTERFACE

Introduction:

This project consist in an interface whose, with an analog FM radio allows send and receive digital audio and data in a format called D-STAR for the amateur radio community.
This protocol was developed by the JARL (Japan amateur radio association) and is detailed in http://www.jarl.com/d-star/shogen.pdf
This project will encode and decode only digital data because the audio part includes a proprietary codec.
This project communicates to the APRS world too, it will receive the signal data portion and show them in a terminal in a formatted output, plus it can send data in a periodic way as a “beacon”
By means of a command set, you can tailor the interface to your needs.

Description:

The software is based in a **SafeRTOS** environment, with 4 tasks, “Led”, “RX”, “TX”, “UART0” and a Scheduler.

Led Task:

It’s used mainly as an indication of the scheduler’s state that tells if it’s running OK.
It employs the evaluation kit led on Port D bit 0, staying on for 100mS and off for 1400mS.
It delays the tasks each pass trough it accord the state of the PD0 Led.

UART0 Task:

It gets and sends text to the operator’s terminal. It allows modify global variable values by means of user commands
Basically, it waits for characters from the UART RX input (from a terminal) and collect them in a buffer until a carriage return arrives.
Then it compares with a command table in a character by a character basis, if there is a match, the command is found, if not, compares again with the next command until the table’s end.

TX Task:

This task translated user digital data in an analog signal to modulate a FM narrow band transmitter.
The analog part is a GMSK modulator-demodulator (CMX589A) that was designed for radio data rate of 4800 baud, this modem send TX clock signal to the interface and accept TX data during clock transitions, to make a Gaussian filtered transmission analog signal.

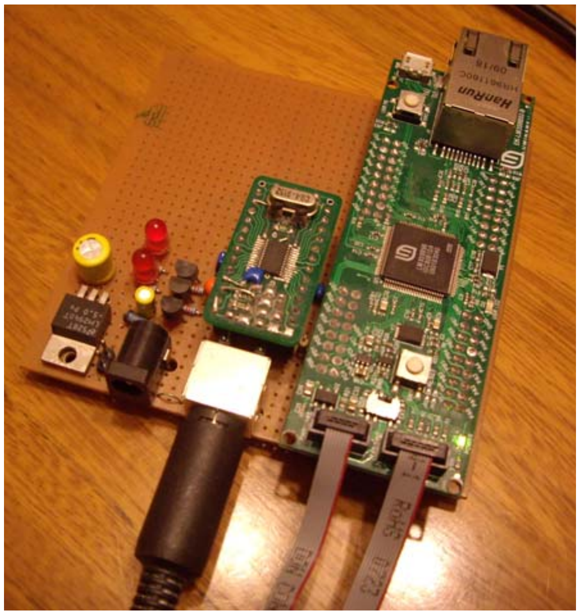
RX Task:

This task do the reverse process of TX task, interacts with interrupt handler by use of SafeRTOS queue, to monitor what’s happening in the RX stream.
After getting the RX frame sync, it waits to have 660 bits to decode the RF Header.
After that, gets the voice and non-voice frames to recover the payload data.
At last, it formats the data and send them to the serial port.

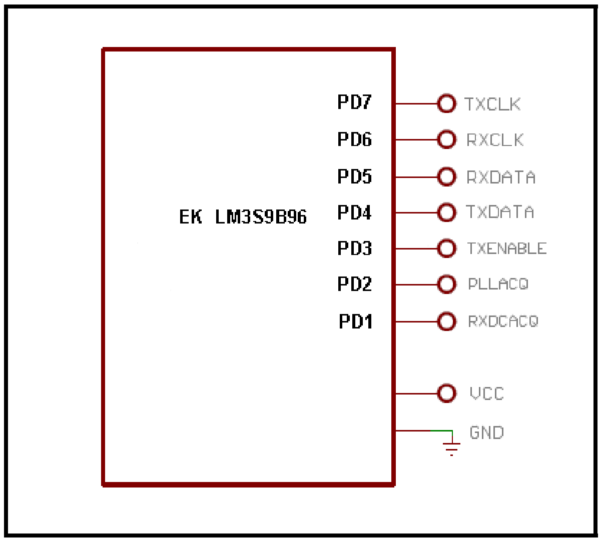
Abstract

Prototype Pictures and Schematic Diagrams:

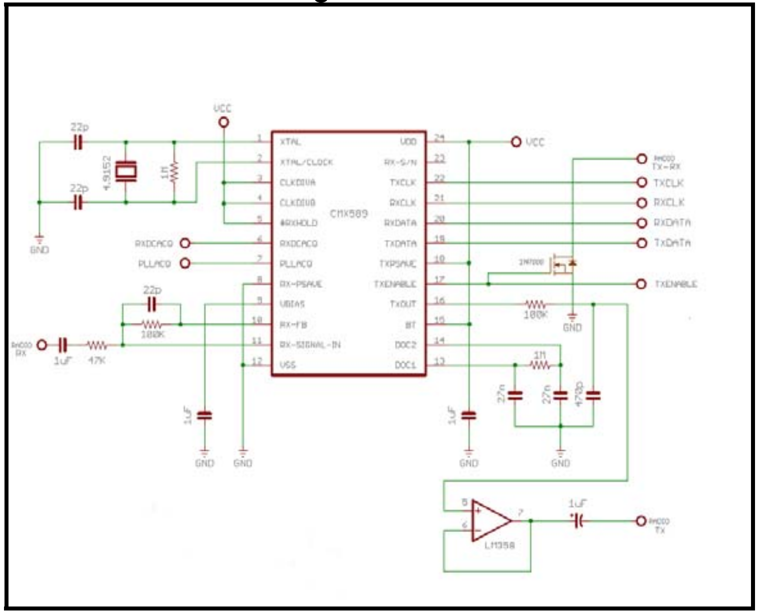
Evaluation Kit mounted with Modem board



Evaluation Kit connections

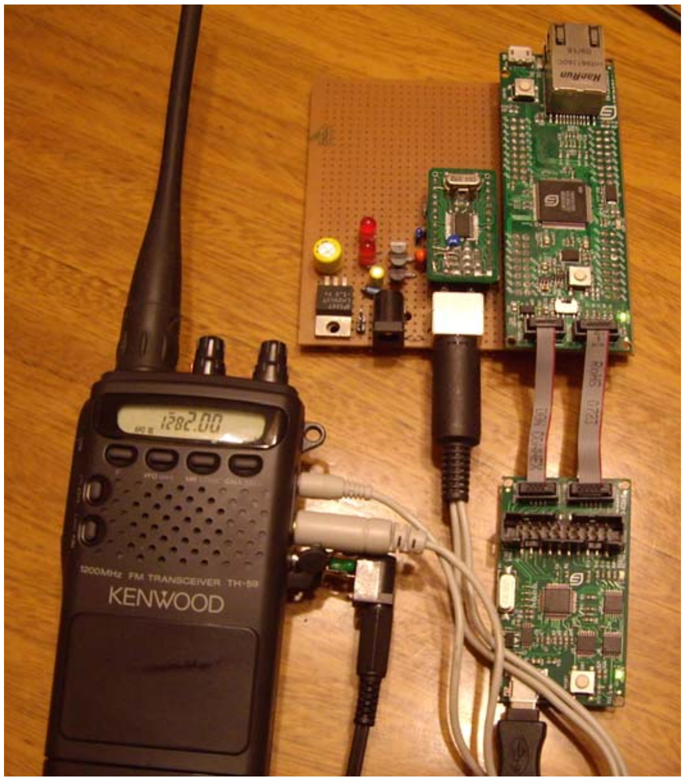


Modem Schematic diagram



Abstract

Evaluation Kit mounted with Modem board and Radio



Code Sample:

```
//---------------------------------------------------------------------------
void SWIntHandler(void)
{
	portBASE_TYPE xHigherPriorityTaskWoken = pdTRUE;
	ROM_GPIOPinIntClear(USER_SW);
	taskYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}

//---------------------------------------------------------------------------
static void LedTask(void *pvParameters)
{
	portTickType xLastTime;
	xLastTime = xTaskGetTickCount();
	while(1)
	{
		xTaskDelayUntil (&xLastTime, LEDON_PERIOD);
		taskENTER_CRITICAL();
		ROM_GPIOPinWrite(USER_LED, 0);
		taskEXIT_CRITICAL();
		xTaskDelayUntil (&xLastTime, LEDON_PERIOD * 14);
		taskENTER_CRITICAL();
		ROM_GPIOPinWrite(USER_LED, GPIO_PIN_0);
		taskEXIT_CRITICAL();
	}
}
```

Abstract

```
//-------------------------------------------------------------------------
static void prvErrorHook(xTaskHandle xHandleOfTaskWithError,
            signed portCHAR *pcErrorString, portBASE_TYPE xErrorCode)
{
    out_text ("Fatal SafeRTOS error!\n");
            while(1);
}

//-------------------------------------------------------------------------
static void prvIdleHook(void)
{
            while(1);
}

//-------------------------------------------------------------------------
static void prvTaskDeleteHook(xTaskHandle xTaskToDelete)
{
}

//-------------------------------------------------------------------------
```

Abstract