Evgeny Sobolev, 41 years. old, phone.: +79003030374, e-mail: hwswdevsev@gmail.com

Location: Voronezh, Russia

Aducation:

VSTU, Radiotechnics , 2001-2006 years.
Specialization: Antennas and microwave devices.

Work experience: JSC "Concern "Sozvezdie", from 2006 till 2015 years
Embedded software engineer, cheif of laboratory

Qualinet Systems, from 2012 till 2017
Embedded Software & Hardware Engineer

Antilatency, from 2017 till 2020
Software Engineer

Auriga Inc. from 2020 till 2020
Software Engineer (Embedded Software & HW Simulators)

AutoVAZ from 2021 till 2021
Chief Specialist (RFQ)

Skills:

- Embedded software development
- Programming languages C, C++, Assembler, less experience with Java
- RTOS usage: uC/COS, FreeRTOS, VxWorks, ECOS.
- Development experience with the following architecture: x86, AVR, ARM (7, 9, Cortex M)
- Android software development (ADK + NDK + eclipse)
- Embedded software development experience using hardware development tools like JTAG
- Linux kernel building, modification. Simple diver writing
- VxWorks, Windows Embedded CE 6.0, BSP development experience
- 2G / 3G / LTE mobile networks logs analyzing experience (L3, RRC, RP, CP, UICC).
- Embedded hardware interfaces UART, UICC, LIN, RS485, SPI, I2C, USB, Ethernet, CAN-HS/CAN-FT, LIN, CPRI, Timers/DMA/etc..
- Reverse engineering architectures: Arm, x86, PowerPC, 8051
- Reverse engineering software: IDA, Ghidra, WinDBG
- Worked with the following international partners as technical specialist: Sequans Communications, Runcom, SEM4G, DesignArt, ASTRI.

Last job skills:

- Different bootloaders
- Embedded software of ALT (Antilatency Tracker)
- Different calibration machines software/firmware
- Hardware development consultations

Courses:

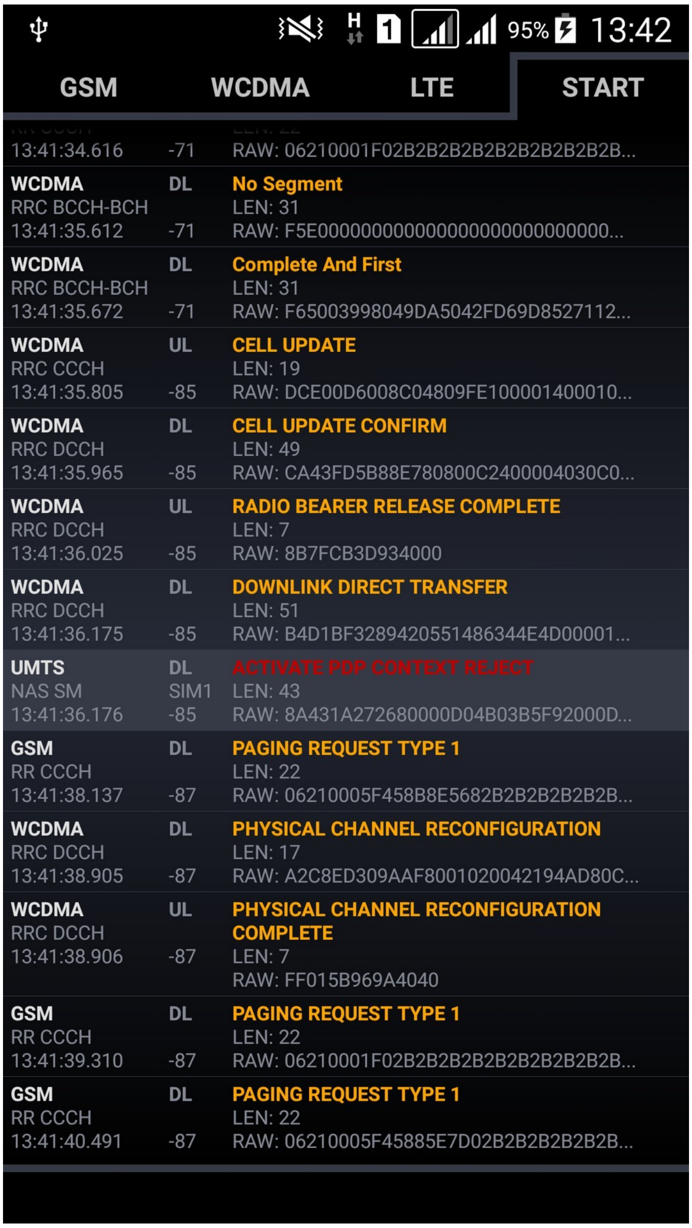
Moscow, Quarta Technologies, Microsoft Windows Embedded Training, 2007 year.
Rishon-Lezion, Runcom Technologies, WiMAX NOC Training, 2011 year.
Remote, HackerU, Information Security (Pen-Test) Training, 2020 year.

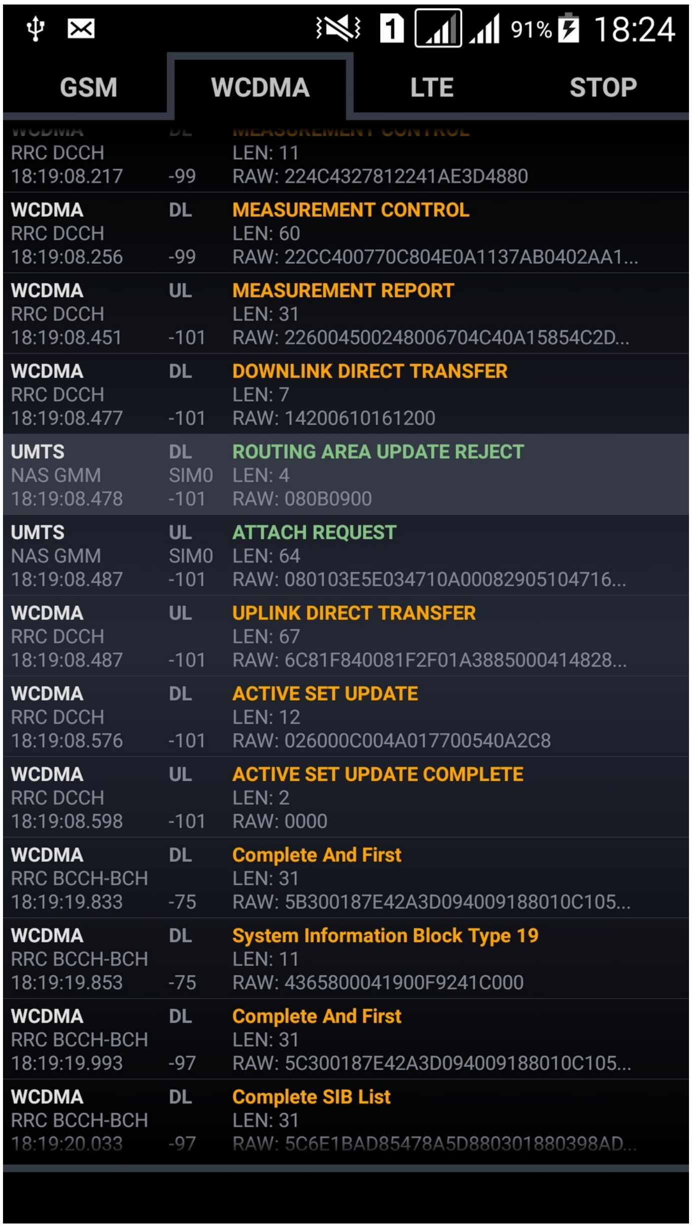
Additional information:

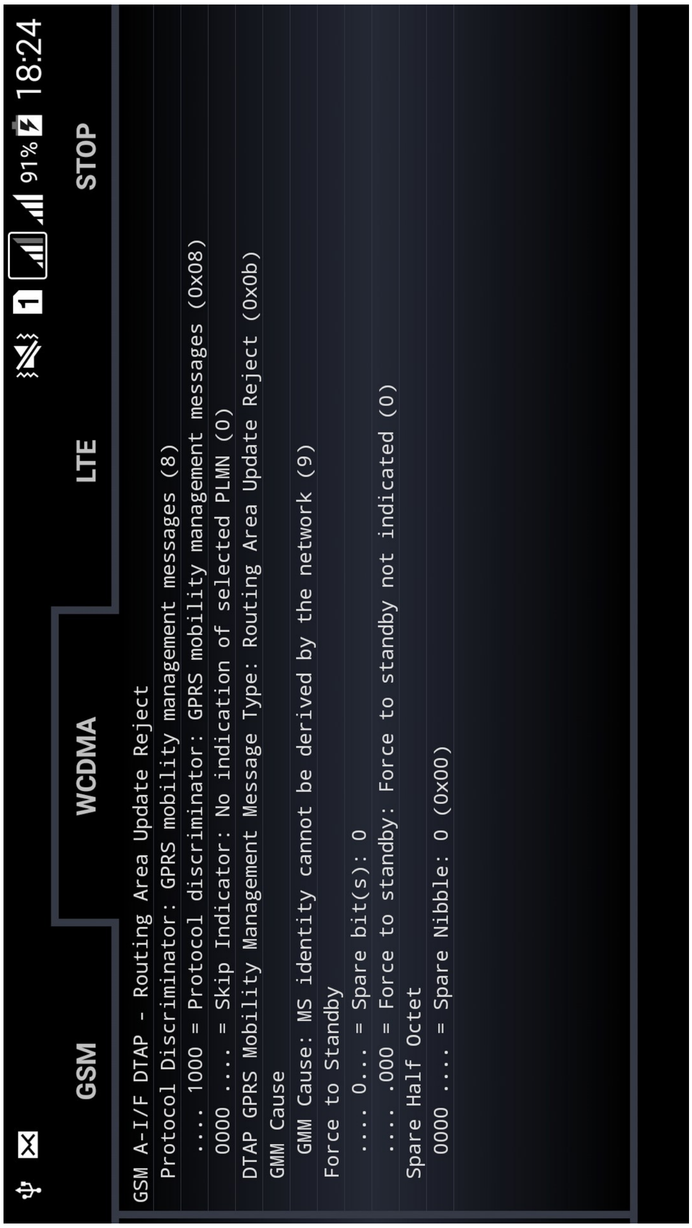
Worked with the following SOC, Microcontrollers, CPUs, DSP:
STM32F465, STM32F407, STM32F103, STM32L432, SQN3120, AR9331, AT91SAM7SXXX, AT91RM9200, AT91SAM9263, IXP465, PXA270, KS8695P, LPC2306, STM32F103, STM32F407, TMS320C6474, STM8S003, AT90S2313/ATTINY2313, ATTINY13, ATMEGA8, ATMEGA16, ATMEGA128, PIC16F84, Z80.

Hardware at Home Lab: Oscilloscope 1GS/100MHz-2CH (SDS7102); Logic analyzer 500MHz 16Ch; SMD Rework Station (Luckey 852D+); Power Supply (HY1803D), Debugging tools J-LINK v8, ST-LINK v2, XDS510-USB, AVRJTAGIC; Xilinx, Altera, Lattice programmers; SPI/I2C programmer.

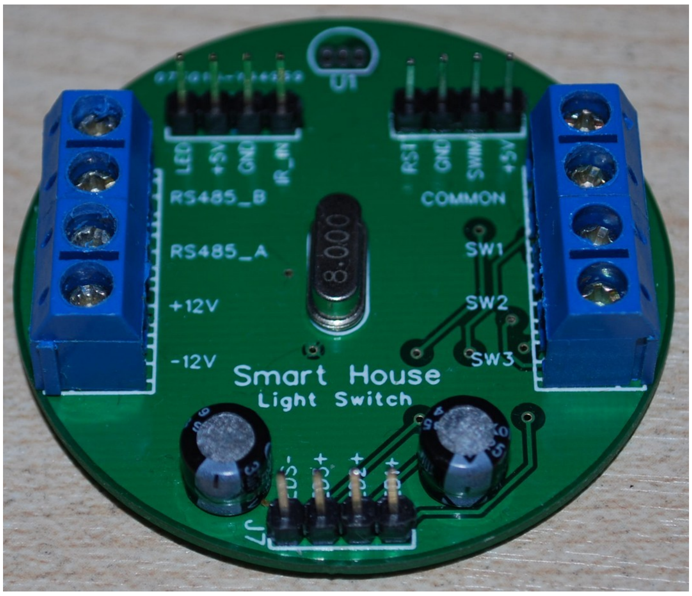
Android software development example.
This application allows to intercept local Layer3 messages between modem of mobile phone and base station. Java + Native. Reverse engineering skills is required to make this project possible. This project is something like simple version of Nemo Handy.

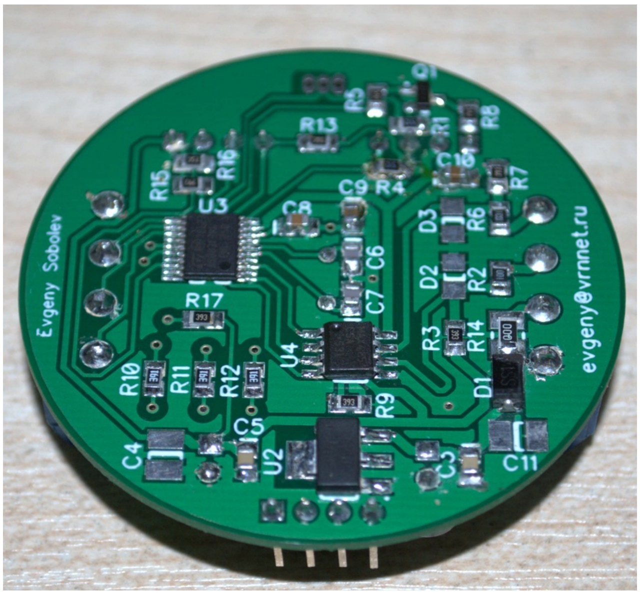



91% 18:24
GSM
WCDMA
LTE
STOP
RRC DCCH LEN: 11
18:19:08.217 -99 RAW: 224C4327812241AE3D4880
WCDMA DL MEASUREMENT CONTROL
RRC DCCH LEN: 60
18:19:08.256 -99 RAW: 22CC400770C804E0A1137AB0402AA1...
WCDMA UL MEASUREMENT REPORT
RRC DCCH LEN: 31
18:19:08.451 -101 RAW: 2260045002480067O4C40A15854C2D...
WCDMA DL DOWNLINK DIRECT TRANSFER
RRC DCCH LEN: 7
18:19:08.477 -101 RAW: 14200610161200
UMTS DL ROUTING AREA UPDATE REJECT
NAS GMM SIM0 LEN: 4
18:19:08.478 -101 RAW: 080B0900
UMTS UL ATTACH REQUEST
NAS GMM SIM0 LEN: 64
18:19:08.487 -101 RAW: 080103E5E034710A00082905104716...
WCDMA UL UPLINK DIRECT TRANSFER
RRC DCCH LEN: 67
18:19:08.487 -101 RAW: 6C81F840081F2F01A3885000414828...
WCDMA DL ACTIVE SET UPDATE
RRC DCCH LEN: 12
18:19:08.576 -101 RAW: 026000C004A017700540A2C8
WCDMA UL ACTIVE SET UPDATE COMPLETE
RRC DCCH LEN: 2
18:19:08.598 -101 RAW: 0000
WCDMA DL Complete And First
RRC BCCH-BCH LEN: 31
18:19:19.833 -75 RAW: 5B300187E42A3D094009188010C105...
WCDMA DL System Information Block Type 19
RRC BCCH-BCH LEN: 11
18:19:19.853 -75 RAW: 4365800041900F9241C000
WCDMA DL Complete And First
RRC BCCH-BCH LEN: 31
18:19:19.993 -97 RAW: 5C300187E42A3D094009188010C105...
WCDMA DL Complete SIB List
RRC BCCH-BCH LEN: 31
18:19:20.033 -97 RAW: 5C6E1BAD85478A5D880301880398AD...


18:24 91%
STOP
LTE
WCDMA
GSM
GSM A-I/F DTAP - Routing Area Update Reject
Protocol Discriminator: GPRS mobility management messages (8)
.... 1000 = Protocol discriminator: GPRS mobility management messages (0x08)
0000 .... = Skip Indicator: No indication of selected PLMN (0)
DTAP GPRS Mobility Management Message Type: Routing Area Update Reject (0x0b)
GMM Cause
GMM Cause: MS identity cannot be derived by the network (9)
Force to Standby
.... 0... = Spare bit(s): 0
.... .000 = Force to standby: Force to standby not indicated (0)
Spare Half Octet
0000 .... = Spare Nibble: 0 (0x00)

Simple hardware development example (Hobby project: https://github.com/hwswdevelop/ModbusIrControl). This is simple project to show that I can develop hardware. This project is developed and soldered by myself. Hardware development skills is much more than presented by this project, but it is under NDA.





Home Lab.

LA5016
Kingst Logic Analyzer
LATTICE
ispDownload Cable
XDS510-USB2.0
TI DSP EMULATOR
XILINX
USB Blaster
PICkit 3
852D
230

ARM Assembler example, quad root calculator

https://github.com/hwswdev/FixedPoint/blob/main/Math/Sqrt.s

```
/*
 ******************************************************************************************
  * @file      Sqrt.s dedicated to STM32Fxx device
  * @author    Evgeny Sobolev
  * @version   V1.0.0
  * @date      2025-03-04
  *
  * @description  Function unt32_t sqrt( uint32_t x )
  *
  * @description  0           =>  0x00000000,
  * @description  65536       =>  0x00010000,
  * @description  2^32-1      =>  0xFFFFFFFF
  * @description  valid  x => [ 0x00000000 .. 0xFFFFFFFF ]
  *
  * @description  Return 32 bit value, where 16bit is decimal & 16bit is fractional
  *
  * @description  Where result, is scaled as
  * @description  0.0         =>  0x00000000,
  * @description  0.5         =>  0x00008000,
  * @description  0.25        =>  0x00004000,
  * @description  1.0         =>  0x00010000,
  * @description  32768.25    =>  0x80004000,
  * @description  65535.9999847 =>  0xFFFFFFFF
  *
  * @description  accuracy is about 1.5e-5, i.e. 1LSB
  *
 *****************************************************************************************
 */

.syntax unified
.cpu cortex-m3
.thumb

.global sqrtFixed
.type sqrtFixed, %function
.align 4
sqrtFixed:
		push { r4-r9, lr }
		// Check if zero, don't calculate
		cmp r0, #0
		beq sqrtFixedEnd

		// Get total bit count of X value
		mov r4, r0
		mov r8, #32
sqrtCalcMsbLoop:
   lsls r4, r4, #1
   bcs sqrtCalcMsbLoopExit
		subs r8, r8, #1
   bne sqrtCalcMsbLoop
sqrtCalcMsbLoopExit:

		// Now R0 <= X, R8 <= MSB, i.e. most significant bit set on value
		// **************************************************
		// Calculate linear approxymation
		// **************************************************
		rsb r7, r8, #32
		add r6, r7, #1
		lsl r9, r0, r6				// R9 <= X << (ToMaxVal+1)
		tst r8, #1
		ittee eq
		ldreq r6, =0xb504f333		// Y0_32bit
		ldreq r5, =0x4afb0ccc// dY_32bit
		ldrne r6, =0x80000000	// Y0_31bit
		ldrne r5, =0x3504f333	// dY_31bit
		umull r4, r5, r9, r5	// R5 <= (dY * X - X0) << MAX
		add r4, r5, r6				// R4 now is linear approxymation of SQRT(x) << MAX
		lsr r6, r7, #1				// Calculate result shift		Shift = 16 - MaxBitCount / 2
		lsr r6, r4, r6				// R6 <= Result of linear approxymation

		// **************************************************
		// Calculate quad correction
		// **************************************************
		// Calculate X-shift offset parable
		mov r4, r9
		tst r4, #0x80000000
		it eq
		rsbeq r4, r4, #0xFFFFFFFF
		sub r4, r4, #0x80000000
		lsl r4, r4, #1
		umull r4, r5, r4, r4
		rsb r4, r5, #0xFFFFFFFF
```

```
// Get X-shift scaled parable value
        ldr r5, =#0x57d86660// Maximum X correction shifted by 4 bit's
        umull r4, r5, r4, r5    // R5 <= X correction shifted by 4bit's and by max bits
        lsr r4, r5, r7                          // R4 <= X correction shifted by 4bit's
        lsr r4, r4, #4                          // R4 <= X correction value
        // Shift X value, to get max sacled parable
        add r4, r4, r0                          // R4 <= X + Xcorrection
        add r5, r7, #1
        lsl r4, r4, r5                          // R4 <= (X + Xcorrection) << (ToMaxVal+1)
        // Now, X-corection applied
        // Calculate Y correction parable, based on corrected X-value
        tst r4, #0x80000000
        it eq
        rsbeq r4, r4, #0xFFFFFFFF
        sub r4, r4, #0x80000000
        lsl r4, r4, #1
        umull r4, r5, r4, r4
        rsb r4, r5, #0xFFFFFFFF
        // Get Y-scale value, depend on sqrt(2) or (2) is scale factor
        tst r8, #0x01
        ite eq
        ldreq r5, =#1726663841          // CorrY * 2 << N
        ldrne r5, =#1220935711          // CorrY * sqrt(2) << N
        umull r4, r5, r4, r5
        lsr r4, r7, #1
        lsr r4, r5, r4
        lsr r4, r4, #5
        add r6, r4, r6                          // R6 <= Result of quad approxymation

        // **************************************************
        // Calculate double parable correction
        // **************************************************
        // R9 is now not corrected, but scaled parable value
        ldr r5, = #0x7504f333           // X-value, where is maximum of quad error correction
        cmp r9, r5                              // Xcorrection central point
        ittee ls
        subls r4, r5, r9                // R4 <= value to calculate parable
        ldrls r5, =#0x8c02d41d          // R5 <= Scale of parable, left side
        subhi r4, r9, r5                // R4 <= value to calculate parable
        ldrhi r5, =#0x75e30c0c          // R5 <= Scale of parable, right side
        umull r4, r5, r4, r5    // R5 <= Scaled parable argument value (shifted by 2)
        lsl r4, r5, #2                          // R4 <= Scaled parable argument value
        // So, now I can calculate parable value
        umull r4, r5, r4, r4    // R5 <= parable value
        rsb r4, r5, #0xFFFFFFFF         // R4 <= inverse parable value of X-value
        // So, now I have to calculate Y-parable, based on X-value
        // But it's the same as before
        tst r4, #0x80000000
        it eq
        rsbeq r4, r4, #0xFFFFFFFF
        sub r4, r4, #0x80000000
        lsl r4, r4, #1
        umull r4, r5, r4, r4
        rsb r4, r5, #0xFFFFFFFF
        // So, now I have to multiply by Scale factor.
        and r7, r8, #1          // R7 <= index of Factor << N, or Factor * sqrt(2) << N
        ldr r5, =#0x7504f333
        cmp r9, r5
        ite hi
        addhi r5, r7, #2
        movls r5, r7
        ldr r7, =sqrtDoubleParableScale
        ldr r5, [ r7, r5, lsl 2 ]
        umull r4, r5, r4, r5
        // Scale correction factor
        rsb r7, r8, #32
        lsr r7, r7, #1
        lsr r4, r5, r7
        // Add correction to the value
        add r6, r6, r4

        // SQRT iterative correction
        ldr r4, =sqrtMaxErrorPowArray
        ldrb r4, [r4, r8]
        mov r7, #1
        lsl r7, r7, r4

sqrtCorrLoop:
        cmp r7, #0
        beq sqrtEndCorrLoop                     // Skip if nothing to do
        umull r4, r5, r6, r6
        rsbs r8, r4, #0
        bne sqrtMulLpNotZero
        sbcs r8, r0, r5
        beq sqrtEndCorrLoop
        rsbs r8, r4, #0
```

```
sqrtMulLpNotZero:
                sbcs r8, r0, r5
                it cc
                subcc r6, r6, r7
                bcc sqrtSkipSum
                rsb r4, r7, #0xFFFFFFFF
                cmp r4, r6
                ite hi
                addhi r6, r6, r7
                ldrls r6, =#0xFFFFFFFF
sqrtSkipSum:
                lsrs r7, r7, #1
                b sqrtCorrLoop
sqrtEndCorrLoop:
                mov r0, r6
sqrtFixedEnd:
                pop { r4-r9, lr }
                bx lr

.align 4
sqrtDoubleParableScale:
.word 0x00064b12     // L
.word 0x00047333     // L
.word 0x0005d399     // R
.word 0x00041eb8     // R

.align 4
sqrtMaxErrorPowArray:
.byte 0x00 // err = 0.0000, bitCount=0
.byte 0x00 // err = 0.0000, bitCount=1
.byte 0x07 // err = 0.0022, bitCount=2
.byte 0x07 // err = 0.0034, bitCount=3
.byte 0x08 // err = 0.0048, bitCount=4
.byte 0x08 // err = 0.0069, bitCount=5
.byte 0x08 // err = 0.0068, bitCount=6
.byte 0x08 // err = 0.0056, bitCount=7
.byte 0x08 // err = 0.0039, bitCount=8
.byte 0x07 // err = 0.0029, bitCount=9
.byte 0x07 // err = 0.0022, bitCount=10
.byte 0x06 // err = 0.0016, bitCount=11
.byte 0x06 // err = 0.0012, bitCount=12
.byte 0x06 // err = 0.0010, bitCount=13
.byte 0x05 // err = 0.0009, bitCount=14
.byte 0x05 // err = 0.0009, bitCount=15
.byte 0x06 // err = 0.0011, bitCount=16
.byte 0x06 // err = 0.0013, bitCount=17
.byte 0x06 // err = 0.0017, bitCount=18
.byte 0x07 // err = 0.0023, bitCount=19
.byte 0x07 // err = 0.0033, bitCount=20
.byte 0x08 // err = 0.0046, bitCount=21
.byte 0x08 // err = 0.0065, bitCount=22
.byte 0x09 // err = 0.0092, bitCount=23
.byte 0x09 // err = 0.0130, bitCount=24
.byte 0x0A // err = 0.0184, bitCount=25
.byte 0x0A // err = 0.0260, bitCount=26
.byte 0x0B // err = 0.0367, bitCount=27
.byte 0x0B // err = 0.0519, bitCount=28
.byte 0x0C // err = 0.0733, bitCount=29
.byte 0x0C // err = 0.1037, bitCount=30
.byte 0x0C
.byte 0x0D
```

Simple Verilog Sample
https://github.com/hwswdev/VerilogLearningSoC/blob/main/rtl/uart_tx.v

```
// ******************************************************************************
// * © Evgeny Sobolev, passport 76 1375783,
// * disallowed to any type  of use by thirdparty
// * uart_tx, - UART module
// * i_rst - input reset signal
// * i_clk - input reset clock signal
// * i_baud8_clk - input baud clock multiplyed by 8
// * i_wr - input data (byte) write signal
// * i_data - input data (byte)
// * o_txe  - output flag, new data can be uploaded
// * o_txc  - output flag, byte transmit complete
// * o_tx   - output tx pin
// *******************************************************************************
module uart_tx (
		i_rst,				// Module reset
		i_clk,				// System clock
		i_baud8_clk,			// Baud clock multiplyed by 8, the same as receiver
		i_wr,				// Write data strobe
		i_data,				// Data 8-bit
		o_tx,				// USART TX pin
		o_txe,				// Tx empty flag
		o_txc				// Tx complete strobe
		);

input  wire i_rst;
input  wire i_clk;
input  wire i_baud8_clk;
input  wire i_wr;
input  wire [7:0]i_data;
output reg o_bsy;
output reg o_txr;

output wire o_rdy;
output wire o_tx;

output reg o_txe;
output reg o_txc;

reg [1:0]r_baud8_clk;
reg r_baud_clk_posedge;
reg [6:0]r_baud8_counter;
reg [10:0]r_data;

assign o_tx = r_data[0];

// Generate baud clock posedge
// It's delayed from original i_baud8_clk, by 3 cycles
// But it's doesn't metters
always @( posedge i_clk or posedge i_rst ) begin
	if ( i_rst ) begin
		r_baud8_clk <= 2'b00;
		r_baud_clk_posedge <= 1'b0;
	end else begin
		r_baud8_clk <= { r_baud8_clk[0], i_baud8_clk };
		r_baud_clk_posedge <= ~r_baud8_clk[0] & ( r_baud8_clk[1] );
	end
end


wire baud_counter_on;
assign baud_counter_on  = ( |(r_baud8_counter) );

wire tx_start_or_data;
assign tx_start_or_data		= ( |(r_baud8_counter[6:2]) );


always @( posedge i_clk or posedge i_rst ) begin
	if ( i_rst ) begin
		r_baud8_counter <= 7'h00;
	end else begin
		if ( baud_counter_on ) begin
			if ( r_baud_clk_posedge ) begin
				r_baud8_counter <= r_baud8_counter + 7'h7F;
			end else begin
				r_baud8_counter <= r_baud8_counter;
			end
		end else begin
			if ( ~r_data[1] ) begin
				r_baud8_counter <= 7'h50;
			end else begin
				r_baud8_counter <= r_baud8_counter;
			end
		end
	end
end
```

```
reg r_txe;
reg r_txc;
reg r_txc_1ck_late;

always @( posedge i_clk or posedge i_rst ) begin
	if ( i_rst ) begin
		o_bsy <= 1'b0;
		r_txe <= 1'b0;
		r_txc <= 1'b0;
		r_txc_1ck_late <= 1'b0;
		o_txe <= 1'b0;
		o_txc <= 1'b0;
	end else begin
		o_bsy  <= baud_counter_on;
		r_txe <= ~tx_start_or_data;
		o_txe <= r_txe & (r_data[1]);// & ~i_wr; // data[1] is zero when new byte is loaded
		r_txc <= ~baud_counter_on;
		r_txc_1ck_late <= r_txc;
		o_txc <= (~r_txc_1ck_late) & r_txc;
	end
end

wire next_bit_strobe;
assign next_bit_strobe = (&(r_baud8_counter[2:0])) & r_baud_clk_posedge;

always @( posedge i_clk or posedge i_rst ) begin
	if ( i_rst ) begin
		r_data <= 10'h3FF;
	end else begin
		if ( i_wr & r_txe ) begin
			r_data <= { 1'b1, i_data, 2'b01 };
		end else begin
			if ( next_bit_strobe ) begin
				r_data <= { 1'b1, r_data[10:1] };
			end else begin
				r_data <= r_data;
			end
		end
	end
end


endmodule
```

https://github.com/hwswdev/VerilogLearningSoC/blob/main/rtl/uart_rx.v

```
// ***************************************************************************
// * © Evgeny Sobolev, passport 76 1375783,
// * disallowed to any type  of use by thirdparty
// * uart_rx, - UART module
// ***************************************************************************

module uart_rx (
	i_rst,					// Module reset
	i_clk,					// System clock
	i_baud8_clk,			// Baud clock multiplyed by 8, the same as receiver
	i_rx,					// RX input pin
	o_data,					// Data 8-bit
	o_rdy,					// Means, bus is busy
	o_bsy
	);

input  wire i_rst;
input  wire i_clk;
input  wire i_baud8_clk;
input  wire i_rx;
output wire [7:0]o_data;
output reg o_rdy;
output wire o_bsy;

reg [1:0]r_baud8_clk;			// i_baud8_clk, i.e. baud clock multipied by 8.
reg r_baud8_clk_posedge;		// delayed by 3 i_clk, i_baud_clk pulse each posedge
reg r_baud8_clk_posedge_1ck; // delayed by 4 i_clk, i_baud_clk pulse each posedge

reg [1:0]rx_shift_reg;
reg rx_bit_edge;
reg [6:0]r_rx_cnt;
reg rx_processing;
reg rx_processing_1ck;
reg r_rx_bit_strobe;
reg [9:0]r_rx_data;
reg [7:0]r_o_data;

assign o_bsy  = (rx_processing & rx_processing_1ck);
assign o_data = r_o_data;
```

```verilog
// Shift i_baud8_clk into r_baud8_clk[1:0]
always @( posedge i_clk or posedge i_rst ) begin
	if ( i_rst ) begin
		r_baud8_clk[1:0] <= 2'b00;
	end else begin
		// Shift data from i_baud8_clk
		r_baud8_clk[1:0] <= { r_baud8_clk[0], i_baud8_clk };
	end
end

// Generate posedge of i_baud8_clk, single i_clk duration
// Signal r_baud8_clk_posedge is shifted by 3 i_clk clock signal
always @( posedge i_clk or posedge i_rst ) begin
	if ( i_rst ) begin
		r_baud8_clk_posedge <= 1'b0;
		r_baud8_clk_posedge_1ck <= 1'b0;
	end else begin
		r_baud8_clk_posedge <= (~r_baud8_clk[1]) & r_baud8_clk[0];
		r_baud8_clk_posedge_1ck <= r_baud8_clk_posedge;
	end
end

// Get USART rx line stream
always @( posedge i_clk or posedge i_rst ) begin
	if ( i_rst ) begin
		rx_shift_reg <= 8'b0;
	end else begin
		rx_shift_reg <= { rx_shift_reg[0], i_rx };
	end
end

// Make USART edge detection
always @( posedge i_clk or posedge i_rst ) begin
	if ( i_rst ) begin
		rx_bit_edge <= 1'b0;
	end else begin
		rx_bit_edge <= (rx_shift_reg[1]) & (~rx_shift_reg[0]);
	end
end

wire w_rx_start;
assign w_rx_start = (rx_bit_edge & (~rx_processing));

// Read USART bit
always @( posedge i_clk or posedge i_rst ) begin
	if ( i_rst ) begin
		rx_processing <= 1'b0;
		rx_processing_1ck <= 1'b0;
		r_rx_cnt <= 7'h00;
	end else begin
		rx_processing <= ( rx_processing & (|(r_rx_cnt[6:0]))) | w_rx_start;
		rx_processing_1ck <= rx_processing;
		if ( rx_processing ) begin
			if ( r_baud8_clk_posedge ) begin
				r_rx_cnt <= r_rx_cnt + 7'h7F;
			end else begin
				r_rx_cnt <= r_rx_cnt;
			end
		end	else begin
			if ( w_rx_start ) begin
				r_rx_cnt <= 7'h4C;
			end else begin
				r_rx_cnt <= r_rx_cnt;
			end
		end
	end
end


always @( posedge i_clk or posedge i_rst ) begin
	if ( i_rst ) begin
		r_rx_bit_strobe <= 1'b0;
	end else begin
		r_rx_bit_strobe <= ( (~r_rx_cnt[2]) & (~r_rx_cnt[1]) & (r_rx_cnt[0]) ) & r_baud8_clk_posedge_1ck;
	end
end

always @( posedge i_clk or posedge i_rst ) begin
	if ( i_rst ) begin
		r_rx_data <= 10'h00;
	end else begin
		if ( r_rx_bit_strobe ) begin
			r_rx_data <= { i_rx, r_rx_data[9:1] };
		end else begin
			r_rx_data <= r_rx_data;
		end
	end
end
```

```
always @( posedge i_clk or posedge i_rst ) begin
		if ( i_rst ) begin
			r_o_data <= 1'b0;
		end else begin
			if ( rx_processing_1ck & (~rx_processing) ) begin
				r_o_data <= r_rx_data[8:1];
			end else begin
				r_o_data <= r_o_data;
			end
		end
end

always @( posedge i_clk or posedge i_rst ) begin
		if ( i_rst ) begin
			o_rdy <= 1'b0;
		end else begin
			o_rdy <= rx_processing_1ck & (~rx_processing);
		end

end

endmodule
```

Photo 06.10.2025