Zephyr® Project
Developer Summit

Unreasonably Efficient AR

Using Zephyr to Achieve More with Simple Hardware

Aedan Cullen

#EmbeddedOSSummit

@aedancullen

How well Zephyr fits with

Rapidly prototyping a “crossover” system

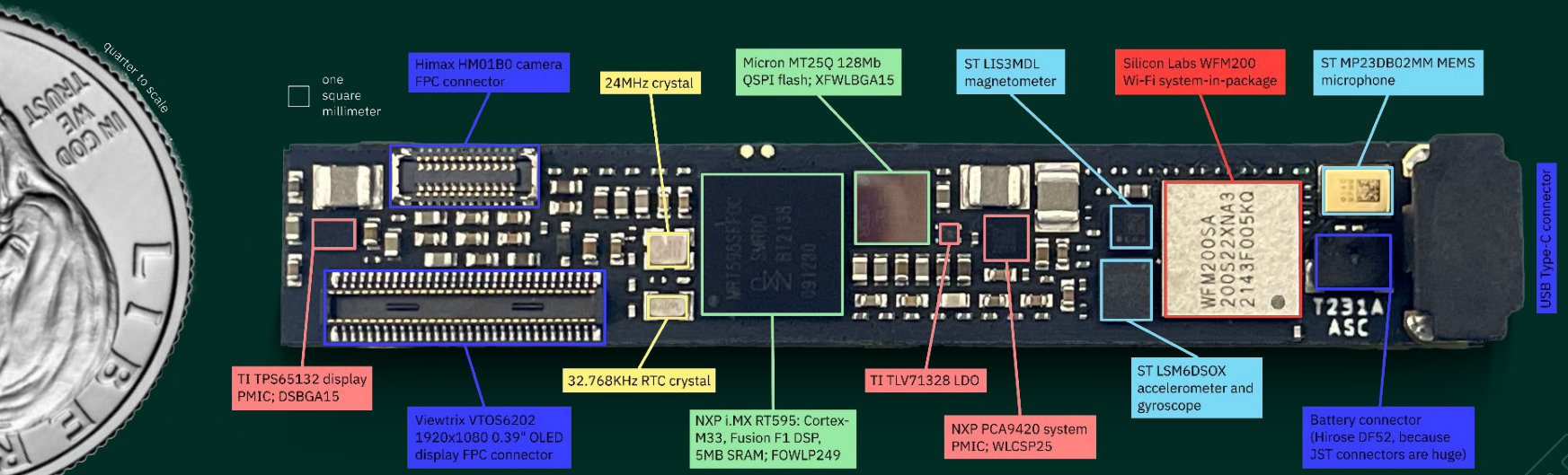
- Goals for this augmented reality project
  - Tiny, lightweight: small HDI board
  - All-day battery life: very low power
  - Phone-level display: min. 1080p
  - Initially monocular

- Pain points from earlier prototypes
  - DRAM power consumption
  - Nonvolatile memory footprint of Linux systems
  - Physical size - must use SiP or PoP
  - Optics

- After settling on an MCU, why choose Zephyr?
  - Quick experimentation and ease-of-use
    - Convenient build/debug workflow
    - Well-written collection of drivers, subsystems
    - Growing, approachable community
  - Long-term confidence
    - Progress made on ambitious/unusual systems stays relevant to others
    - Vendor-independence

What we're bringing up

The board (T231)



The main character

NXP i.MX RT595S

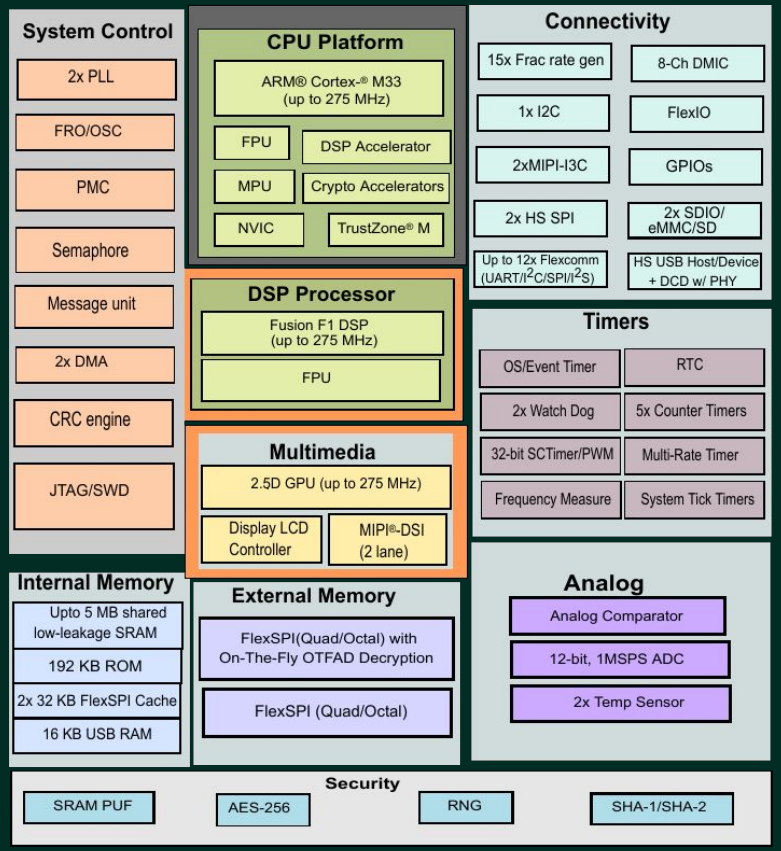
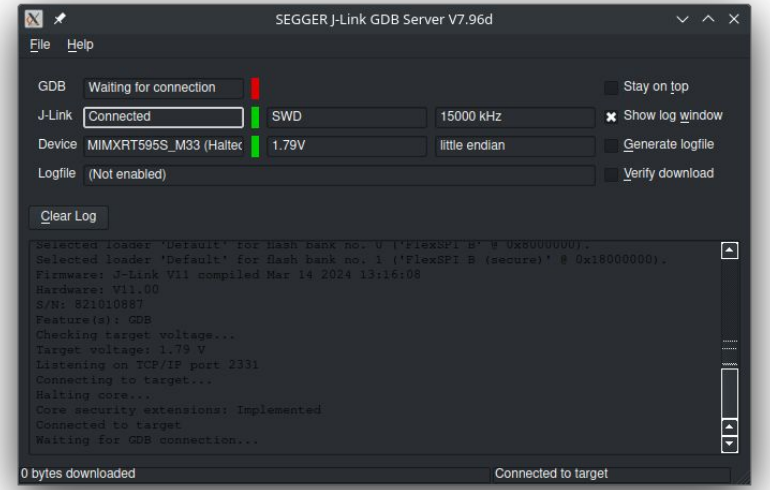


Image credit: NXP Semiconductors

Step zero

SWD ✓ XIP code from flash • Devicetree •



***flash_config.c** for EVK*

```
const flexspi_nor_config_t flash_config = {
    .memConfig =
        {
            .tag                = FLEXSPI_CFG_BLK_TAG,
            .version            = FLEXSPI_CFG_BLK_VERSION,
            .readSampleClkSrc   = kFlexSPIReadSampleClk_ExternalInputFromDqsPad,
            .csHoldTime         = 3,
            .csSetupTime        = 3,
            .deviceModeCfgEnable = 1,
            .deviceModeType     = kDeviceConfigCmdType_Spi2Xpi,
            .waitTimeCfgCommands = 1,
            .deviceModeSeq =
                {
                    .seqNum   = 1,
                    .seqId    = 6, /* See Lookup table for more details */
                    .reserved = 0,
                },
            .deviceModeArg = 2, /* Enable OPI DDR mode */
            .controllerMiscOption =
                (1u << kFlexSpiMiscOffset_SafeConfigFreqEnable) | (1u << kFlexSpiMiscOffset_DdrModeEnab
            .deviceType     = kFlexSpiDeviceType_SerialNOR,
            .sflashPadType  = kSerialFlash_8Pads,
            .serialClkFreq  = kFlexSpiSerialClk_60MHz,
            .sflashA1Size   = 64ul * 1024u * 1024u,
            .busyOffset     = 0u,
            .busyBitPolarity = 0u,
            .lookupTable =
                {
                    /* Read */
                    [0] = FLEXSPI_LUT_SEQ(CMD_DDR, FLEXSPI_8PAD, 0xEE, CMD_DDR, FLEXSPI_8PAD, 0x11),
                    [1] = FLEXSPI_LUT_SEQ(RADDR_DDR, FLEXSPI_8PAD, 0x20, DUMMY_DDR, FLEXSPI_8PAD, 0x04
                    [2] = FLEXSPI_LUT_SEQ(READ_DDR, FLEXSPI_8PAD, 0x04, STOP_EXE, FLEXSPI_1PAD, 0x00),

                    /* Read status SPI */
                    [4 * 1 + 0] = FLEXSPI_LUT_SEQ(CMD_SDR, FLEXSPI_1PAD, 0x05, READ_SDR, FLEXSPI_1PAD,

                    /* Read Status OPI */
                    [4 * 2 + 0] = FLEXSPI_LUT_SEQ(CMD_DDR, FLEXSPI_8PAD, 0x05, CMD_DDR, FLEXSPI_8PAD, 0
                    [4 * 2 + 1] = FLEXSPI_LUT_SEQ(RADDR_DDR, FLEXSPI_8PAD, 0x20, DUMMY_DDR, FLEXSPI_8PA
```

Basic hardware

QSPI flash part, I2C, I/O voltages

Micron

128Mb, 1.8V Multiple I/O Serial Flash Memory
Features

Micron Serial NOR Flash Memory

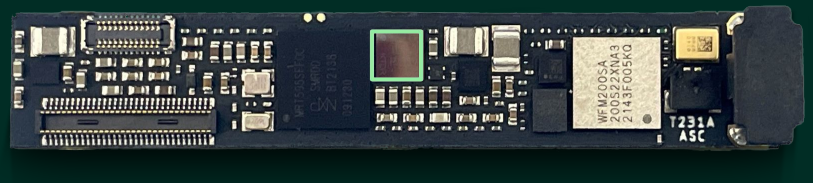
1.8V, Multiple I/O, 4KB, 32KB, 64KB, Sector Erase

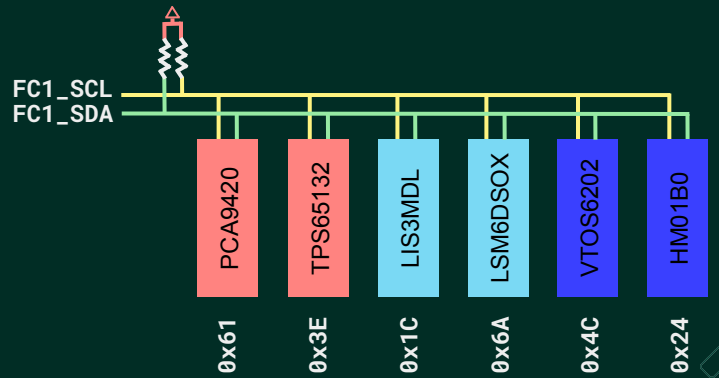
MT25QU128ABA

Features

- SPI-compatible serial bus interface
- Single and double transfer rate (STR/DTR)
- Clock frequency
  - 166 MHz (MAX) for all protocols in STR
  - 90 MHz (MAX) for all protocols in DTR
- Dual/quad I/O commands for increased through-put up to 90 MB/s
- Supported protocols in both STR and DTR
  - Extended I/O protocol
  - Dual I/O protocol
  - Quad I/O protocol
- Execute-in-place (XIP)
- PROGRAM/ERASE SUSPEND operations
- Volatile and nonvolatile configuration settings
- Software reset
- Additional reset pin for selected part numbers
- Dedicated 64-byte OTP area outside main memory
  - Readable and user-lockable
  - Permanent lock with PROGRAM OTP command
- Erase capability
  - Bulk erase
  - Sector erase 64KB uniform granularity
  - Subsector erase 4KB, 32KB granularity
- Security and write protection
  - Volatile and nonvolatile locking and software write protection for each 64KB sector
  - Nonvolatile configuration locking
  - Password protection
  - Hardware write protection: nonvolatile bits (BP[3:0] and TB) define protected area size

Options	Marking
• Voltage	
– 1.7–2.0V	U
• Density	
– 128Mb	128
• Device stacking	
– Monolithic	A
• Device generation	B
• Die revision	A
• Pin configuration	
– RESET# and HOLD#	8
• Sector Size	
– 64KB	E
• Packages – JEDEC-standard, RoHS-compliant	
– 24-ball T-PBGA, 05/6mm x 8mm (5 x 5 array)	12
– 24-ball T-PBGA 05/6mm x 8mm (4 x 6 array)	14
– Wafer level chip-scale package,15 balls , 9 active balls (XFWLBGA 0.5P)	54
– 8-pin SOP2, 208 mils body width (SO8W)	SE
– 16-pin SOP2, 300 mils body width (SO16W)	SF
– W-PDFN-8 6mm x 5mm (MLP8 6mm x 5mm)	W7
– W-PDFN-8 8mm x 6mm (MLP8 8mm x 6mm)	W9
• Standard security	0
• Special options	





- All PIO banks use 1.8V supplies

Step zero

SWD ✓ XIP code from flash ✓ Devicetree ✓

*custom **`flash_config.c`***

```
/* QSPI config for MT25Q series (tested on MT25QU128ABA8E54, MT25QL128ABA1ESE) */
const flexspi_nor_config_t flexspi_config = {
    .memConfig = {
        .tag                    = FLEXSPI_CFG_BLK_TAG,
        .version                = FLEXSPI_CFG_BLK_VERSION,
        .readSampleClkSrc       = kFlexSPIReadSampleClk_LoopbackInternally,
        .csHoldTime             = 3,
        .csSetupTime            = 3,
        .columnAddressWidth     = 0,
        .deviceModeCfgEnable    = 0,
        .deviceModeType         = 0,
        .waitTimeCfgCommands    = 0,
        .deviceModeSeq          = {
            .seqNum = 0,
            .seqId  = 0,
        },
        .deviceModeArg          = 0,
        .configCmdEnable        = 0,
        .configModeType         = {0},
        .configCmdSeqs          = {{0}},
        .configCmdArgs          = {0},
        .controllerMiscOption   = (0),
        .deviceType             = 1,
        .sflashPadType          = kSerialFlash_4Pads,
        .serialClkFreq          = kFlexSpiSerialClk_166MHz,
        .lutCustomSeqEnable     = 0,
        .sflashA1Size           = BOARD_FLASH_SIZE,
        .sflashA2Size           = 0,
        .sflashB1Size           = 0,
        .sflashB2Size           = 0,
        .csPadSettingOverride   = 0,
        .sclkPadSettingOverride = 0,
        .dataPadSettingOverride = 0,
        .dqsPadSettingOverride  = 0,
        .timeoutInMs            = 0,
        .commandInterval        = 0,
        .busyOffset             = 0,
        .busyBitPolarity        = 0,
        .lookupTable = {
            /* Fast Read */
            [4*0+0] = FLEXSPI_LUT_SEQ(CMD_SDR, FLEXSPI_1PAD, 0xEB, RADDR_SDR, FLEXSPI_4PAD, 0x18),
            [4*0+1] = FLEXSPI_LUT_SEQ(MODE4_SDR, FLEXSPI_4PAD, 0x00, DUMMY_SDR, FLEXSPI_4PAD, 0x09),
            [4*0+2] = FLEXSPI_LUT_SEQ(READ_SDR, FLEXSPI_4PAD, 0x04, STOP_EXE, FLEXSPI_1PAD, 0x00),

            /* Read Status */
            [4*1+0] = FLEXSPI_LUT_SEQ(CMD_SDR, FLEXSPI_1PAD, 0x05, READ_SDR, FLEXSPI_1PAD, 0x04),

            /* Write Enable */
            [4*3+0] = FLEXSPI_LUT_SEQ(CMD_SDR, FLEXSPI_1PAD, 0x06, STOP_EXE, FLEXSPI_1PAD, 0x00),

            /* Sector Erase */
            [4*5+0] = FLEXSPI_LUT_SEQ(CMD_SDR, FLEXSPI_1PAD, 0x20, RADDR_SDR, FLEXSPI_1PAD, 0x18),

            /* Block Erase */
```

*HAL config in **`board.c`**: I/O voltage, clocks*

```
static int t231_init(void)
{
    /* Set the correct voltage range according to the board. */
    power_pad_vrange_t vrange = {
        .Vdde0Range = kPadVol_171_198,
        .Vdde1Range = kPadVol_171_198,
        .Vdde2Range = kPadVol_171_198,
        .Vdde3Range = kPadVol_171_198,
        .Vdde4Range = kPadVol_171_198
    };

    POWER_SetPadVolRange(&vrange);

                ⋮

#if DT_NODE_HAS_COMPAT_STATUS(DT_NODELABEL(flexcomm1), nxp_lpc_i2c, okay)
    /* Switch FLEXCOMM1 to FRO_DIV4 */
    CLOCK_AttachClk(kFRO_DIV4_to_FLEXCOMM1);
#endif
```

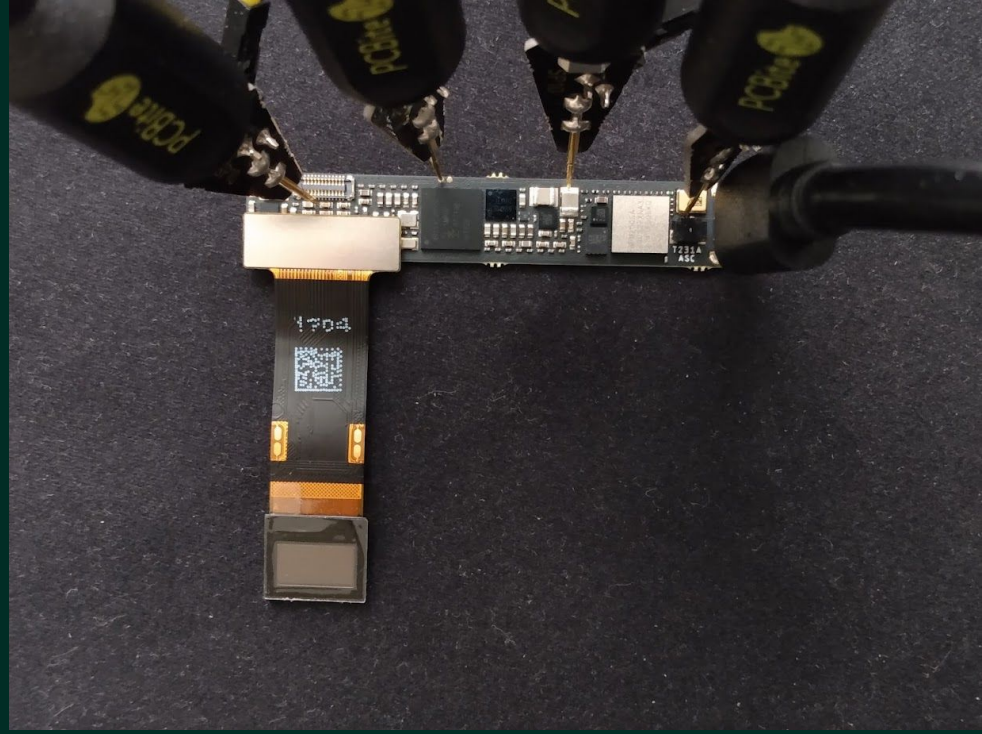
Basic devicetree configuration

```
gp_i2c: &flexcomm1 {
    compatible = "nxp,lpc-i2c";
    status = "okay";
    clock-frequency = <I2C_BITRATE_FAST>;
    #address-cells = <1>;
    #size-cells = <0>;
    pinctrl-0 = <&pinmux_flexcomm1_i2c>;
    pinctrl-names = "default";

    pca9420: pca9420@61 {

                ⋮

    lis3mdl: lis3mdl@1c {

                ⋮

    lsm6dso: lsm6dso@6a {

                ⋮
```

Now for a challenge

Viewtrix VTOS6202

- 1920x1080 0.39” OLED-on-silicon microdisplay
- 5,644 pixels per inch
- Single or **dual** 4-lane MIPI DSI interfaces (up to 8 lanes)
- 24bpp DSI video mode packets required
- Non-continuous HS clock required (important!)
- No internal memory
  - Must be constantly refreshed
  - Can’t deep-sleep between frames



- On the RT500, we have:
  - Only two MIPI DSI lanes
  - Maximum D-PHY bit clock of 895.1MHz
  - 5MiB SRAM (no external PSRAM in this design, to save space)

Limits are for pushing

Display enablement

- Things in our favor:
  - RT500: All the chips I tested are happy with a 1056MHz D-PHY clock at room temperature
  - VTOS6202: can select a smaller region of pixels to refresh, and set its position with register writes
  - VTOS6202: Possible to configure just 2 lanes of one DSI interface using undocumented registers*
- What fits in SRAM?
  - One **1920**x1080 16bpp framebuffer
  - Two **1080**x1080 16bpp framebuffers
- What fits over the 2-lane D-PHY?
  - **1920**x1080 @43Hz w/ 1056MHz PHY
  - **1080**x1080 @76Hz w/ 1056MHz PHY
  - **1080**x1080 @65Hz w/ 895MHz PHY
- Reduce porches/blanking to minimum supported by display

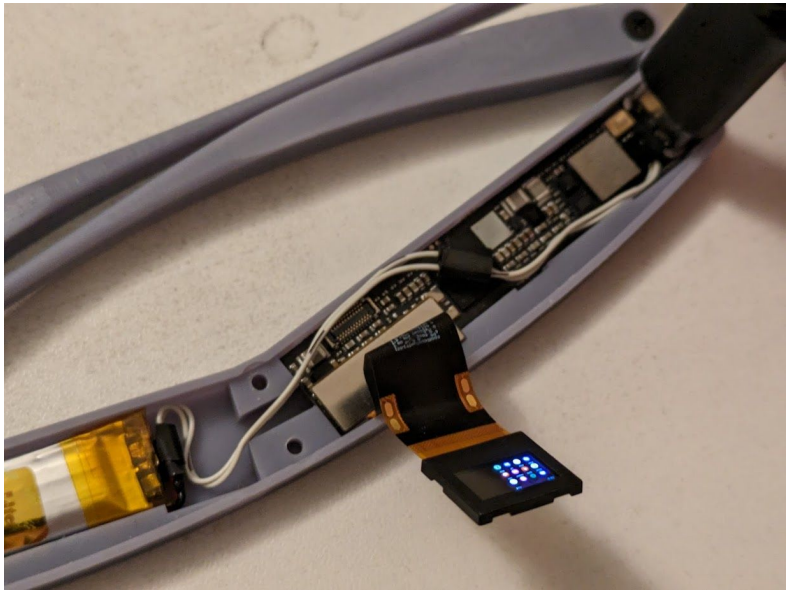
```
&mipi_dsi {
    status = "okay";
    nxp,lcdif = <&lcdif>;
    dpi-color-coding = "24-bit";
    dpi-pixel-packet = "24-bit";
    dpi-video-mode = "burst";
    dpi-bllp-mode = "low-power";
    noncontinuous-clk; /* Putting the clock lane in LP mode between HS bursts is
    autoinsert-eotp;
    /*
     * PHY clock is given by the following formula:
     * (pixel clock * bits per pixel) / MIPI data lanes
     */
    phy-clock = <1056000000>; /* 528MHz * 18 / dphy-clk-div (sets PFD3 in soc.c)

    vtos6202: vtos6202@0 {
        status = "okay";
        compatible = "viewtrix,vtos6202";
        reg = <0x0>;
        reset-gpios = <&gpio0 6 GPIO_ACTIVE_HIGH>;
        data-lanes = <2>;
        width = <1920>;
        height = <1080>;
        pixel-format = <MIPI_DSI_PIXFMT_RGB565>;
    };
};
```

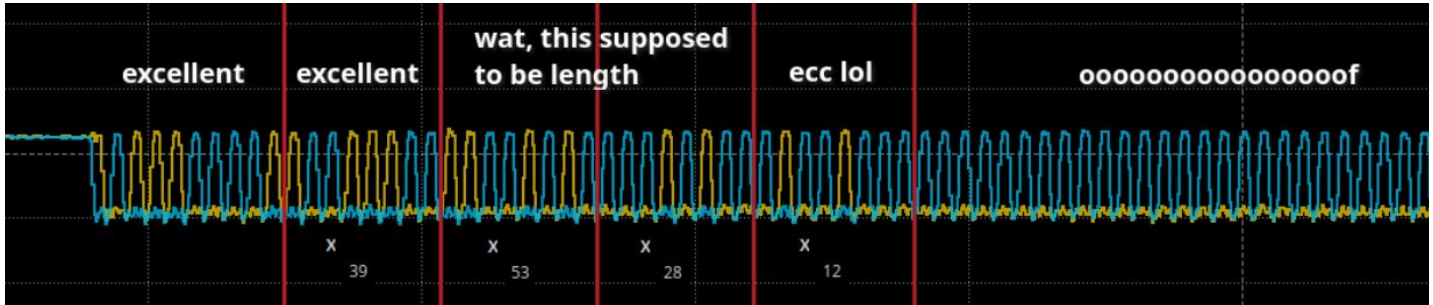

*very long story omitted

Limits are for pushing

Display enablement

- VTOS6202 requires a large initial set of register writes
  - Must use the 0x39 packet (DCS Long Write)
  - Reading power state after config writes shows the display never leaves sleep
  - Casually checking registers shows correct values
  - **Why?** NXP HAL incorrectly handles Long Writes with small payloads (<=2 bytes)
    - Two bytes of payload are treated as length
    - Dumps some memory over DSI :)
    - “Normal” displays use short writes here

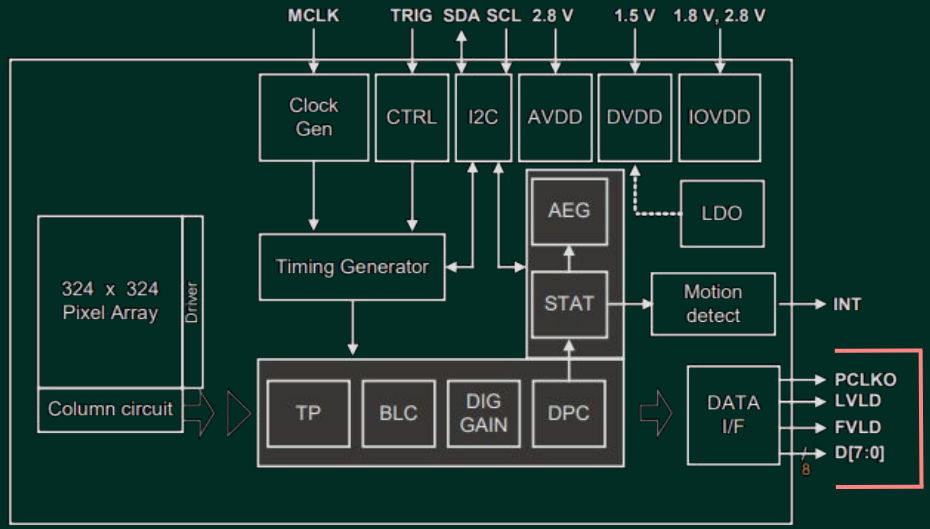




Hoping for CSI-2 RX in an RT500 successor…

For now, a parallel camera interface

- NXP FlexIO peripheral: a creative shift-register structure that can be cascaded
  - Handles this parallel interface nicely
  - Retrieving data from it is more nuanced
- Need to manage the individual shifter contents and organize them without CPU
  - NXP's solution: Use SmartDMA
  - Has APIs for certain high-level behaviors
- SmartDMA is just another DMA engine, right?
  - There's more to learn…



Himax HM01B0 block diagram

Please do not the binary blob

Discovering the EZH

- SmartDMA is a 32-bit programmable I/O processor
- Keypad scanning, encoders, etc.
- Elsewhere called:
  - EZH
  - I/O Handler Architecture B (or IOH)
- Standard programmers only see the particular APIs its firmware exposes
  - Useful application-specific functionality might not be available
  - Out of luck if you need something else… unless you're not a standard programmer

```
/*!
 * @brief The API index when using s_smartdmaDisplayFirmware.
 */
enum _smartdma_display_api
{
    kSMARTDMA_FlexIO_DMA_Endian_Swap = 0U,
    kSMARTDMA_FlexIO_DMA_Reverse32,
    kSMARTDMA_FlexIO_DMA,
    kSMARTDMA_FlexIO_DMA_Reverse,              /*!< Send data to FlexIO with r
    kSMARTDMA_RGB565To888,                     /*!< Convert RGB565 to RGB888 a
    smartdma_rgb565_rgb888_param_t. */
    kSMARTDMA_FlexIO_DMA_RGB565To888,          /*!< Convert RGB565 to RGB888 a
    smartdma_flexio_mculcd_param_t. */
    kSMARTDMA_FlexIO_DMA_ARGB2RGB,             /*!< Convert ARGB to RGB and se
    smartdma_flexio_mculcd_param_t. */
    kSMARTDMA_FlexIO_DMA_ARGB2RGB_Endian_Swap, /*!< Convert ARGB to RGB, then
    FlexIO, use         parameter smartdma_flexio_mculcd_param_t. */
    kSMARTDMA_FlexIO_DMA_ARGB2RGB_Endian_Swap_Reverse, /*!< Convert ARGB to RG
    to FlexIO, use parameter smartdma_flexio_mculcd_param_t. */
```

⋮

- It would seem beneficial for Zephyr users to be aware of these types of specialized processors when they exist

Please do not the binary blob

Discovering the EZH

```
const uint8_t s_smartdmaDisplayFirmware[] = {
    0x60U,  0x00U,  0x10U,  0x24U,  0x40U,  0x01U,
    0x04U,  0x02U,  0x10U,  0x24U,  0x68U,  0x03U,
    0xE8U,  0x0DU,  0x10U,  0x24U,  0xFCU,  0x0FU,
    0xE8U,  0x14U,  0x10U,  0x24U,  0xACU,  0x25U,
    0xCCU,  0x27U,  0x10U,  0x24U,  0x18U,  0x19U,
    0xE4U,  0x1BU,  0x10U,  0x24U,  0x54U,  0x1DU,
    0xDCU,  0x2EU,  0x10U,  0x24U,  0x68U,  0x33U,
    0x8CU,  0x3BU,  0x10U,  0x24U,  0xF8U,  0x22U,
    0x12U,  0x00U,  0x00U,  0x00U,  0x00U,  0x28U,
    0x04U,  0x18U,  0x04U,  0x27U,  0x01U,  0x80U,
    0x01U,  0xB0U,  0x05U,  0x02U,  0x10U,  0x04U,
    0x06U,  0x48U,  0xC7U,  0x08U,  0x1AU,  0x80U,
    0x1CU,  0xB4U,  0x00U,  0x00U,  0x18U,  0xECU,
    0x01U,  0x0CU,  0x1CU,  0x01U,  0x10U,  0x08U,
    0x11U,  0x08U,  0x38U,  0x00U,  0x01U,  0x0CU,
    0x05U,  0x04U,  0x22U,  0x32U,  0x11U,  0x08U,
    0x10U,  0x08U,  0x2CU,  0x10U,  0x05U,  0x08U,
    0x01U,  0x0CU,  0x1CU,  0x01U,  0x10U,  0x08U,
    0x11U,  0x08U,  0x38U,  0x00U,  0x01U,  0x0CU,
    0x05U,  0x10U,  0x22U,  0x32U,  0x11U,  0x08U,
    0x10U,  0x08U,  0x2CU,  0x10U,  0x05U,  0x14U,
    0x01U,  0x0CU,  0x1CU,  0x01U,  0x10U,  0x08U,
    0x11U,  0x08U,  0x38U,  0x00U,  0x01U,  0x0CU,
    0x05U,  0x1CU,  0x22U,  0x32U,  0x1CU,  0xB4U,
    0x00U,  0x28U,  0x04U,  0x00U,  0x00U,  0x00U,
    0x00U,  0x08U,  0x66U,  0x0BU,  0x0EU,  0x00U,
    0x1CU,  0xB4U,  0x00U,  0x00U,  0x15U,  0x42U,
    0x00U,  0x28U,  0x04U,  0x00U,  0x00U,  0x2CU,
```

⋮

ezhdis ➡

```
E_NOP                                                   // 0x00100054 (entry: kSMARTDMA_FlexIO_DMA_Endian_Swap)
E_COND_LOAD_SIMM(EU, CFS, 0, 0)                         // 0x00100058 (load shifted immediate)
E_COND_LOAD_SIMM(EU, CFM, 257, 0)                       // 0x0010005C
E_COND_PER_READ(EU, R6, SMARTDMA_ARM2SMARTDMA)          // 0x00100060 (read struct address from mailbox)
E_COND_LDR(EU, R0, R6, 0)                               // 0x00100064 (unpack buffer size and stack pointer)
E_COND_LDR(EU, R1, R6, 1)                               // 0x00100068  |
E_COND_LDR(EU, SP, R6, 2)                               // 0x0010006C  V
E_COND_ASR(EU, R1, R1, 5)                               // 0x00100070 (div. buf size by 32 bytes-per-iteration)
E_COND_SUB_IMM(EU, R1, R1, 1)                           // 0x00100074 (prep loop params)
E_COND_ADD_IMM(EU, R2, PC, 140)                         // 0x00100078
E_COND_TIGHT_LOOP(EU, R2, R1)                           // 0x0010007C (zero-overhead loop downwards?)
E_COND_LDR_POST(EU, R3, R0, 1)                          // 0x00100080
E_COND_HOLD(EU)                                         // 0x00100084 (wfi?)
E_COND_BCLR_IMM(EU, CFM, CFM, 0)                        // 0x00100088 (bit clear)
E_COND_FEND_ASR(EU, R2, R3, 0)                          // 0x0010008C (flip endian)
E_COND_LDR_POST(EU, R3, R0, 1)                          // 0x00100090 (load from buffer, post-increment)
E_COND_ROR(EU, R2, R2, 16)                              // 0x00100094 (rotate-right)
E_COND_PER_WRITE(EU, R2, 0x00032200)                    // 0x00100098 (AHB write to SHIFTBUF0)
E_COND_FEND_ASR(EU, R2, R3, 0)                          // 0x0010009C
E_COND_LDR_POST(EU, R3, R0, 1)                          // 0x001000A0
E_COND_ROR(EU, R2, R2, 16)                              // 0x001000A4
E_COND_PER_WRITE(EU, R2, 0x00032204)                    // 0x001000A8 (AHB write to SHIFTBUF1)
E_COND_FEND_ASR(EU, R2, R3, 0)                          // 0x001000AC
E_COND_LDR_POST(EU, R3, R0, 1)                          // 0x001000B0
E_COND_ROR(EU, R2, R2, 16)                              // 0x001000B4
E_COND_PER_WRITE(EU, R2, 0x00032208)                    // 0x001000B8 (AHB write to SHIFTBUF2)
E_COND_FEND_ASR(EU, R2, R3, 0)                          // 0x001000BC
E_COND_LDR_POST(EU, R3, R0, 1)                          // 0x001000C0
E_COND_ROR(EU, R2, R2, 16)                              // 0x001000C4
E_COND_PER_WRITE(EU, R2, 0x0003220C)                    // 0x001000C8 (AHB write to SHIFTBUF3)
E_COND_FEND_ASR(EU, R2, R3, 0)                          // 0x001000CC
E_COND_LDR_POST(EU, R3, R0, 1)                          // 0x001000D0
E_COND_ROR(EU, R2, R2, 16)                              // 0x001000D4
E_COND_PER_WRITE(EU, R2, 0x00032210)                    // 0x001000D8 (AHB write to SHIFTBUF4)
E_COND_FEND_ASR(EU, R2, R3, 0)                          // 0x001000DC
E_COND_LDR_POST(EU, R3, R0, 1)                          // 0x001000E0
E_COND_ROR(EU, R2, R2, 16)                              // 0x001000E4
E_COND_PER_WRITE(EU, R2, 0x00032214)                    // 0x001000E8 (AHB write to SHIFTBUF5)
E_COND_FEND_ASR(EU, R2, R3, 0)                          // 0x001000EC
E_COND_LDR_POST(EU, R3, R0, 1)                          // 0x001000F0
E_COND_ROR(EU, R2, R2, 16)                              // 0x001000F4
E_COND_PER_WRITE(EU, R2, 0x00032218)                    // 0x001000F8 (AHB write to SHIFTBUF6)
E_COND_FEND_ASR(EU, R2, R3, 0)                          // 0x001000FC
E_COND_LDR_POST(EU, R3, R0, 1)                          // 0x00100100
E_COND_ROR(EU, R2, R2, 16)                              // 0x00100104
E_COND_PER_WRITE(EU, R2, 0x0003221C)                    // 0x00100108 (AHB write to SHIFTBUF7)
E_COND_HOLD(EU)                                         // 0x0010010C
E_INT_TRIGGER(0x00000000)                               // 0x00100110
E_COND_LOAD_SIMM(EU, CFS, 0, 0)                         // 0x00100114 (packing bits 110 110 110... into CFM)
E_COND_LOAD_SIMM(EU, R0, 219, 24)                       // 0x00100118  |
E_COND_LOAD_SIMM(EU, R1, 109, 16)                       // 0x0010011C  |
E_COND_LOAD_SIMM(EU, R2, 182, 8)                        // 0x00100120  |
E_COND_XOR_LSL(EU, R0, R0, R1, 0)                       // 0x00100124  V
E_COND_XOR_LSL(EU, CFM, R0, R2, 0)                      // 0x00100128 (xor, post logical shift left)
E_COND_HOLD(EU)                                         // 0x0010012C
E_COND_GOTO(EU, 0x00100114)                             // 0x00100130
```

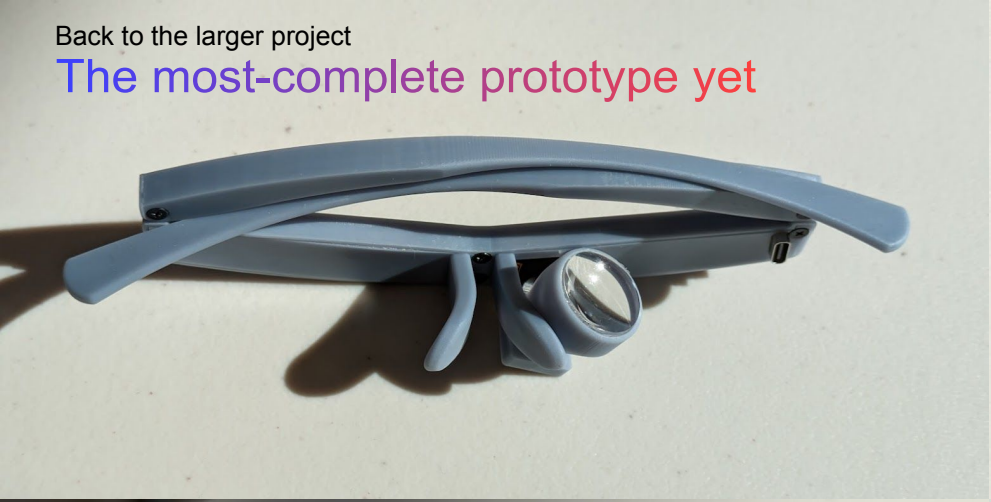
EMBEDDED OPEN SOURCE SUMMIT

github.com/aedancullen/ezhdis

`fsl_smartdma_prv.h`

Back to the larger project

The most-complete prototype yet



Back to the larger project

Other system features and overall usage

- Accel/mag/gyro (LSM6DSOX, LIS3MDL)
  - 9-DOF AHRS using NXP's Kalman filter implementation (NXPSensorFusion)
  - Possible to tap the edge of the device as a selection mechanism
- Wi-Fi (SiLabs WFM200): connect to an Android phone using Wi-Fi Direct
  - Stream display frames to the AR device, and sensor data in the opposite direction
  - Simple UI-oriented video codec - optimizations for scrolled content, regions of solid color, etc.
- Areas for improvement
  - Fusion F1 DSP codec implementation
  - Late-stage warping using the GPU
  - Optics design
  - Bi-ocular display system?

Enough abstraction to be convenient, enough metal to push the limits

- Some significant successes in low-overhead prototyping
  - Easy to understand RT500 display peripherals and drivers
  - Zephyr devicetree/Kconfig infrastructure eases:
    - Transition between platforms
    - Basic system bringup

- Specialized peripherals allow achieving more with the silicon you have
  - Only if you have the tools to utilize them well
  - Most users want the simple picture
  - Advanced users need the full picture

- New and emerging accelerators (AI/ML)
  - Abstraction vs. fine-grained HW control: even more important

Zephyr® Project

Developer Summit