ZCU104 & Zybo FPGA 프로젝트 상세 보고서

이 보고서는 `dinh/` 디렉토리에 포함된 3개의 FPGA 프로젝트에 대한 종합적인 분석을 제공합니다.

목차

---

1. 프로젝트 개요

프로젝트	타겟 보드	목적	핵심 기술
zcu104-baremetal-imgio	ZCU104 (Zynq UltraScale+ MPSoC)	SD 카드 BMP 이미지 로더	FatFs, A53 베어메탈
zcu104-a53-mobilenetv1-baseline	ZCU104 (Zynq UltraScale+ MPSoC)	양자화된 MobileNet 숫자 인식	INT8 양자화, DW/PW Conv
zybo_memcopy_accel_interrupt	Zybo Z7-10 (Zynq-7000)	HLS 메모리 복사 가속기	Vitis HLS, 인터럽트, AXI

---

2. zcu104-baremetal-imgio

2.1 프로젝트 설명

ZCU104 보드의 **Cortex-A53** 베어메탈 환경에서 **SD 카드의 24비트 BMP 이미지**를 로드하는 최소 스타터 프로젝트입니다. 이 프로젝트는 `zcu104-a53-mobilenetv1-baseline` 의 기반이 되는 SD I/O 패턴을 제공합니다.

2.2 주요 기능

- **FatFs (xilffs)** 라이브러리를 사용한 SD 카드 마운트 및 파일 읽기
- **24비트 비압축 BMP (BI_RGB)** 형식 파싱
- BGR → RGB 색상 순서 변환
- Bottom-up → Top-down 이미지 방향 변환
- 행 스트라이드 4바이트 정렬 처리
- UART 로그 출력 (115200 8N1)

2.3 디렉토리 구조

```
zcu104-baremetal-imgio/
├── sd_bmp_loader.cpp      # 메인 소스 코드
├── bmp_24.bmp             # 테스트용 BMP 이미지
├── README.md              # 프로젝트 문서
└── zcu104-baremetal-imgio.md  # 상세 코드 리뷰 보고서
```

2.4 핵심 코드 분석

SD 카드 마운트

```
static bool sd_mount() {
    FRESULT rc = f_mount(&fatfs, "0:/", 1);  // 드라이브 0, 즉시 마운트
    if (rc != FR_OK) {
        xil_printf("f_mount failed: %dWrWn", rc);
        return false;
    }
    return true;
}
```

BMP 헤더 파싱

```
#pragma pack(push, 1)  // 1바이트 정렬로 파일 형식과 일치
struct BMPFileHeader {
    uint16_t bfType;      // 'BM' = 0x4D42
    uint32_t bfSize;      // 파일 전체 크기
```

```
    uint16_t bfReserved1;
    uint16_t bfReserved2;
    uint32_t bfOffBits;   // 픽셀 데이터 시작 오프셋
};
#pragma pack(pop)
```

이미지 구조체 (NHWC 레이아웃)

```
struct Image {
    int width = 0;
    int height = 0;
    int channels = 0;           // 3 for RGB
    vector<uint8_t> pixels;     // [height × width × channels]
};
// 인덱싱: pixels[(y * width + x) * channels + c]
```

2.5 빌드 및 실행

필수 조건:

- Vivado 2022.1 + Vitis 2022.1
- ZCU104 보드 (USB-UART + JTAG 연결)
- FAT/FAT32 포맷 microSD 카드

Vitis 프로젝트 설정:

1. XSA 생성: Zynq MPSoC, ZCU104 프리셋 적용 (PS DDR, UART, SD 활성화)
2. 플랫폼 생성: `psu_cortexa53_0` , OS = standalone
3. BSP에 **xilffs** 라이브러리 추가
4. 링커 스크립트: Heap 4MB, Stack 64KB (DDR 영역)

SD 카드 레이아웃:

```
0:/test.bmp   (24비트 비압축 BMP)
```

예상 출력:

```
SD BMP Loader (bare-metal)
SD mounted: 0:/
File size: 120054 bytes
Loaded BMP: 200x200, 3 channels
Top-left pixel RGB = (255,0,0)
Done.
```

---

3. zcu104-a53-mobilenetv1-baseline

3.1 프로젝트 설명

zcu104-baremetal-imgio를 기반으로 확장된 프로젝트로, SD 카드에서 **32×32 RGB BMP**를 로드하고 **양자화된 MobileNet 스타일** 추론 파이프라인을 실행하여 **0-9 숫자 분류**를 수행합니다.

3.2 아키텍처

```
입력 BMP (32×32×3)
    ↓
전처리 (RGB → NHWC uint8)
    ↓
Depthwise Conv 3×3 + ReLU6
    ↓
Pointwise Conv 1×1
    ↓
Global Average Pooling
    ↓
Softmax
    ↓
Top-K 예측 출력
```

3.3 디렉토리 구조

```
zcu104-a53-mobilenetv1-baseline/
├── vitis_src/
|   ├── mobilenet_bm.cpp       # 메인 베어메탈 애플리케이션
|   ├── ref_kernels.c          # 양자화 커널 구현
|   ├── ref_kernels.h          # 커널 헤더
|   └── vitis_src.md           # 코드 리뷰 보고서
├── tools/
|   ├── synth_digits_train.py  # 합성 숫자 데이터 학습
|   ├── export_bins.py         # 가중치 양자화 및 내보내기
|   └── tools.md               # 도구 설명 보고서
├── assets/
|   ├── dw3x3_c3.bin           # DW 가중치 (27 bytes)
|   ├── pw1x1_c10x3.bin        # PW 가중치 (30 bytes)
|   ├── labels.txt             # 0-9 레이블
|   └── samples/               # digit_0.bmp ~ digit_9.bmp
├── platform/
|   └── standalone_zynq_core.xsa  # 사전 빌드 하드웨어
└── README.md
```

3.4 양자화 커널 구현

텐서 구조체

```c
typedef struct {
    int H;          // 높이
    int W;          // 너비
    int C;          // 채널
    uint8_t *data;  // 픽셀 데이터
    float scale;    // 양자화 스케일
    int zp;         // 영점 (zero-point)
} tensor_u8_nhwc_t;
```

역양자화/재양자화 헬퍼

```c
// 역양자화: uint8 → float
static inline float deq(uint8_t q, float s, int zp) {
    return s * ((int)q - zp);
}

// 재양자화: float → uint8
static inline uint8_t req(float r, float s, int zp) {
    int q = zp + (int)lrintf(r / s);
    return (uint8_t)clamp(q, 0, 255);
}
```

Depthwise Convolution 3×3

```c
void dwconv3x3_nhwc_u8(
    const tensor_u8_nhwc_t *in,    // 입력 텐서
    const uint8_t *k3x3,           // 3×3 커널 [C × 9]
    const int32_t *bias,           // 편향 (선택)
    float w_scale, int w_zp,       // 가중치 양자화 파라미터
    tensor_u8_nhwc_t *out,         // 출력 텐서
    int apply_relu6                // ReLU6 활성화 적용 여부
);
```

핵심 연산:

- 각 채널에 대해 독립적인 3×3 필터 적용
- 경계 처리: 범위 외 픽셀 건너뛰기 (제로 패딩 효과)
- ReLU6: `max(0, min(6, x))` 클램핑

Pointwise Convolution 1×1

```c
void pwconv1x1_nhwc_u8(
    const tensor_u8_nhwc_t *in,    // 입력 [H × W × Cin]
    const uint8_t *k1x1,           // 1×1 커널 [Cout × Cin]
```

```
    const int32_t *bias,
    float w_scale, int w_zp,
    tensor_u8_nhwc_t *out          // 출력 [H × W × Cout]
);
```

핵심 연산:

- 모든 입력 채널의 가중합으로 출력 채널 생성
- GEMM (General Matrix Multiply) 형태로 동작

3.5 Python 학습 도구

모델 정의 (PyTorch)

```
class TinyDigitDSCNN(nn.Module):
    def __init__(self, cout=10):
        super().__init__()
        self.dw = nn.Conv2d(3, 3, kernel_size=3, padding=1, groups=3, bias=False)
        self.pw = nn.Conv2d(3, cout, kernel_size=1, bias=False)

    def forward(self, x):
        x = self.dw(x)                  # Depthwise
        x = torch.clamp(x, 0.0, 6.0)    # ReLU6
        x = self.pw(x)                  # Pointwise
        x = x.mean(dim=(2, 3))          # Global Average Pool
        return x                         # logits [N, 10]
```

양자화 파라미터

파라미터	값	설명
in_scale	0.02	입력 스케일
in_zp	128	입력 영점
w_scale	0.0019579321	가중치 스케일 (학습 후 계산)
w_zp	128	가중치 영점

가중치 내보내기

```
# DW: c*9 + (ky+1)*3 + (kx+1) 레이아웃
# PW: co*Cin + ci 레이아웃
with open("dw3x3_c3.bin", "wb") as f: f.write(bytearray(dw_flat))
with open("pw1x1_c10x3.bin", "wb") as f: f.write(bytearray(pw_flat))
```

3.6 SD 카드 레이아웃

```
0:/assets/dw3x3_c3.bin         (27 bytes)
0:/assets/pw1x1_c10x3.bin      (30 bytes)
0:/assets/labels.txt           (10 lines: 0..9)
0:/assets/samples/digit_X.bmp  (32×32 RGB)
```

3.7 실행 결과 예시

```
mobilenet_bm: bare-metal demo
SD mounted: 0:/
Image loaded: 32x32
DWConv: 15 ms (15234 us)
PWConv: 8 ms (8421 us)
AvgPool: 2 ms (2103 us)
Softmax: 0 ms (156 us)
Top-5:
 1) 1 : 241/255 (94%)
 2) 7 : 8/255 (3%)
 3) 4 : 4/255 (2%)
 4) 9 : 2/255 (1%)
 5) 2 : 0/255 (0%)
Done.
```

3.8 다음 단계: HLS 가속기

README에서 제안하는 HLS IP 아키텍처:

```
MM2S (AXI DMA) → DW3x3 IP → PW1x1 IP → S2MM (AXI DMA)
                 (AXI-Lite)   (AXI-Lite)
```

주요 최적화:

- DW 3×3: 스트리밍 스텐실 + 라인 버퍼 + 채널 병렬성
- PW 1×1: 시스톨릭 어레이 + 타일링된 GEMM
- DW→PW 퓨전: 중간 DRAM 트래픽 제거

---

4. zybo_memcopy_accel_interrupt

4.1 프로젝트 설명

Zybo Z7-10 보드에서 **Vitis HLS**로 구현한 **메모리 복사 가속기**입니다. **인터럽트 기반** 동기화와 **AXI4 마스터 버스트 전송**을 사용하여 DDR 메모리 간 고속 데이터 복사를 수행합니다.

4.2 디렉토리 구조

```
zybo_memcopy_accel_interrupt/
├── src/
│   ├── Vitis-HLS/
│   │   ├── memcopy_accel.cpp       # HLS 가속기 구현
│   │   ├── memcopy_accel_test.cpp  # C 시뮬레이션 테스트벤치
│   │   ├── memcopy_accel_tb_data.h # 테스트 데이터 헤더
│   │   └── Vitis-HLS.md            # 코드 리뷰 보고서
│   ├── Vitis-BareMetal/
│   │   ├── main.c                  # 베어메탈 데모 애플리케이션
│   │   ├── memcopy_accel.c         # 가속기 드라이버
│   │   ├── memcopy_accel.h         # 드라이버 헤더
│   │   └── Vitis-BareMetal.md      # 코드 리뷰 보고서
│   └── GPIO-demo/
│       ├── zybo_GPIO.c             # Zybo GPIO 데모
│       ├── zedboard_GPIO.c         # ZedBoard GPIO 데모
│       └── GPIO-demo.md            # 코드 리뷰 보고서
├── docs/
│   └── Memcopy-Guide.pdf           # 가이드 문서
└── README.md
```

4.3 HLS 가속기 구현

인터페이스 설계

```
void memcopy_accel(
    uint32_t* src,   // AXI4 Master: 소스 DDR
    uint32_t* dst,   // AXI4 Master: 목적지 DDR
    uint32_t len     // 복사할 바이트 수
) {
#pragma HLS INTERFACE m_axi port=src offset=slave bundle=AXI_SRC depth=1024
#pragma HLS INTERFACE m_axi port=dst offset=slave bundle=AXI_DST depth=1024
#pragma HLS INTERFACE s_axilite port=src bundle=CTRL_BUS
#pragma HLS INTERFACE s_axilite port=dst bundle=CTRL_BUS
#pragma HLS INTERFACE s_axilite port=len bundle=CTRL_BUS
#pragma HLS INTERFACE s_axilite port=return bundle=CTRL_BUS
```

인터페이스 특징:

포트	인터페이스	번들	설명
src	m_axi + s_axilite	AXI_SRC + CTRL_BUS	소스 주소 (AXI4 마스터 읽기)
dst	m_axi + s_axilite	AXI_DST + CTRL_BUS	목적지 주소 (AXI4 마스터 쓰기)
len	s_axilite	CTRL_BUS	바이트 수
return	s_axilite	CTRL_BUS	제어 (ap_start, ap_done)

브랜치리스 버스트 전송

```
#define BURST_LEN 32  // 32 워드 = 128 바이트 버스트

uint32_t buffer[BURST_LEN];
#pragma HLS ARRAY_PARTITION variable=buffer complete dim=1

copy_loop:
for (uint32_t i = 0; i < num_words; i += BURST_LEN) {
#pragma HLS PIPELINE II=1

    // 브랜치리스 청크 계산 (조건문 제거)
    uint32_t diff = num_words - i;
    uint32_t mask = (diff >= BURST_LEN);
    mask = -mask;  // 0xFFFFFFFF 또는 0x0
    uint32_t chunk = (mask & BURST_LEN) | (~mask & diff);

read_loop:
    for (uint32_t j = 0; j < BURST_LEN; j++) {
#pragma HLS UNROLL
        bool valid = (j < chunk);
        uint32_t vmask = -((uint32_t)valid);
        buffer[j] = src[i + j] & vmask;
    }

write_loop:
    for (uint32_t j = 0; j < BURST_LEN; j++) {
#pragma HLS UNROLL
        bool valid = (j < chunk);
        uint32_t vmask = -((uint32_t)valid);
        dst[i + j] = buffer[j] & vmask;
    }
}
```

HLS 최적화:

지시자	효과
ARRAY_PARTITION complete	버퍼 32개 레지스터로 완전 분할
PIPELINE II=1	매 사이클 새 버스트 시작
UNROLL	읽기/쓰기 루프 완전 전개

4.4 베어메탈 드라이버

레지스터 맵

```
#define MEMCOPY_ACCEL_BASEADDR   0x40000000u
#define MEMCOPY_ACCEL_CTRL_OFFSET   0x00u  // Control (ap_start, ap_done)
#define MEMCOPY_ACCEL_GIE_OFFSET    0x04u  // Global Interrupt Enable
#define MEMCOPY_ACCEL_IER_OFFSET    0x08u  // IP Interrupt Enable
#define MEMCOPY_ACCEL_ISR_OFFSET    0x0Cu  // IP Interrupt Status
#define MEMCOPY_ACCEL_SRC_OFFSET    0x10u  // 소스 주소
#define MEMCOPY_ACCEL_DST_OFFSET    0x1cu  // 목적지 주소
#define MEMCOPY_ACCEL_LEN_OFFSET    0x28u  // 바이트 수
```

가속기 시작

```
void memcopy_accel_start(uint32_t src_addr, uint32_t dst_addr, uint32_t len) {
    Xil_Out32(base_addr + MEMCOPY_ACCEL_SRC_OFFSET, src_addr);
    Xil_Out32(base_addr + MEMCOPY_ACCEL_DST_OFFSET, dst_addr);
    Xil_Out32(base_addr + MEMCOPY_ACCEL_LEN_OFFSET, len);
    Xil_Out32(base_addr + MEMCOPY_ACCEL_CTRL_OFFSET, MEMCOPY_AP_START_MASK);
}
```

인터럽트 핸들러

```
void memcopy_isr(void *CallbackRef) {
    memcopy_accel_interrupt_clear();

    // 중요: AXI 쓰기 플러시를 위한 더미 읽기
    // 이것이 없으면 인터럽트 스톰 발생
    (void)Xil_In32(MEMCOPY_BASE + MEMCOPY_ACCEL_ISR_OFFSET);

    memcopy_done = 1;
}
```

인터럽트 스톰 방지:

- AXI-Lite 쓰기는 포스트 트랜잭션
- ISR 클리어가 완료되기 전에 반환하면 재진입 발생
- 더미 읽기로 쓰기 버퍼 플러시

4.5 메인 애플리케이션 흐름

```
int main() {
    // 1. 드라이버 및 인터럽트 초기화
    memcopy_accel_init(MEMCOPY_BASE);
    setup_interrupt_system();

    // 2. 버퍼 할당 (32KB)
    uint32_t *src_buf = malloc(BYTE_LEN);
    uint32_t *dst_buf = malloc(BYTE_LEN);

    // 3. 소스 데이터 초기화 (0xA5A50000 | index 패턴)
    for (uint32_t i = 0; i < NUM_WORDS; ++i)
        src_buf[i] = 0xA5A50000u | i;

    // 4. 캐시 플러시 (PL이 최신 데이터 읽도록)
    Xil_DCacheFlushRange((unsigned int)src_buf, BYTE_LEN);
    Xil_DCacheFlushRange((unsigned int)dst_buf, BYTE_LEN);

    // 5. 가속기 시작 및 WFI 대기
    memcopy_done = 0;
    memcopy_accel_start((uint32_t)src_buf, (uint32_t)dst_buf, BYTE_LEN);
    do {
        __asm__ volatile ("wfi");  // 인터럽트까지 저전력 대기
    } while (!memcopy_done);

    // 6. 캐시 무효화 (CPU가 최신 데이터 읽도록)
    Xil_DCacheInvalidateRange((unsigned int)dst_buf, BYTE_LEN);

    // 7. 검증 및 성능 비교
    // ...
}
```

4.6 GPIO 데모

Zybo Z7-10 버튼-LED 데모

```
// 4개 버튼 (BTN0-BTN3), 4개 LED (LD0-LD3)
#define BTN_MASK 0b1111
#define LED_MASK 0b1111

while (1) {
    data = XGpio_DiscreteRead(&btn_device, BTN_CHANNEL);
    if (data != 0)
        data = LED_MASK;  // 버튼 눌림 → 모든 LED 켜기
    else
        data = 0;
    XGpio_DiscreteWrite(&led_device, LED_CHANNEL, data);
}
```

ZedBoard 스위치-LED 미러 데모

```c
// 8개 딥 스위치 (SW0-SW7), 8개 LED (LD0-LD7)
#define SW_MASK  0xFF
#define LED_MASK 0xFF

while (1) {
    sw_val = XGpio_DiscreteRead(&sw_gpio, SW_CHANNEL) & SW_MASK;
    XGpio_DiscreteWrite(&led_gpio, LED_CHANNEL, sw_val);  // 직접 미러링
}
```

---

5. 프로젝트 간 관계

공통 패턴

패턴	zcu104-baremetal-imgio	zcu104-a53-mobilenetv1-baseline	zybo_memcopy_accel_interrupt
SD 카드 FatFs	✓	✓	✗
BMP 파싱	✓	✓	✗
시간 측정 (XTime)	✗	✓	✓
캐시 관리	△	△	✓
인터럽트	✗	✗	✓
HLS IP	✗	✗ (참조만)	✓

---

6. 기술 스택 요약

하드웨어 플랫폼

보드	프로세서	FPGA	프로젝트
ZCU104	Cortex-A53 (64-bit)	Zynq UltraScale+ MPSoC	imgio, mobilenet
Zybo Z7-10	Cortex-A9 (32-bit)	Zynq-7000	memcopy

개발 도구

도구	버전	용도
Vivado	2022.1	하드웨어 설계, 비트스트림 생성
Vitis	2022.1	소프트웨어 개발, HLS
Vitis HLS	2022.1	C/C++ → RTL 합성
Python	3.x	모델 학습 (PyTorch), 가중치 내보내기

핵심 라이브러리

라이브러리	용도	프로젝트
xilffs (FatFs)	SD 카드 파일 시스템	imgio, mobilenet
xtime_l	고정밀 시간 측정	mobilenet, memcopy
xscugic	GIC 인터럽트 컨트롤러	memcopy
xgpio	AXI GPIO 드라이버	memcopy (GPIO-demo)
PyTorch	딥러닝 프레임워크	mobilenet (tools/)

인터페이스 프로토콜

프로토콜	설명	사용처
AXI4-Lite	제어 레지스터 접근	HLS IP 제어
AXI4 Master	메모리 버스트 전송	HLS IP 데이터 경로
AXI4-Stream	스트리밍 데이터 전송	DW/PW IP (계획)
UART	직렬 통신 (115200 8N1)	디버그 출력

---

7. 활용 가이드

7.1 학습 순서 권장

```
1단계: zcu104-baremetal-imgio
├── 목표: 베어메탈 환경 이해, SD 카드 I/O
├── 핵심: FatFs 마운트, BMP 파싱, NHWC 레이아웃
└── 시간: 1-2일

2단계: zybo_memcopy_accel_interrupt
├── 목표: HLS 가속기 개발 패턴 학습
├── 핵심: HLS 인터페이스, 버스트 전송, 인터럽트
└── 시간: 3-5일

3단계: zcu104-a53-mobilenetv1-baseline
├── 목표: 양자화 신경망 추론 이해
├── 핵심: INT8 양자화, DW/PW Conv, PyTorch 학습
└── 시간: 5-7일

4단계: HLS 가속기 구현 (도전)
├── 목표: mobilenet의 ref_kernels를 HLS IP로 변환
├── 참조: memcopy의 인터페이스 패턴 적용
└── 시간: 2-4주
```

7.2 확장 아이디어

단기 (1-2주)

- [ ] mobilenet: 다양한 이미지 크기 지원 (리사이즈 추가)
- [ ] mobilenet: JPEG 지원 (stb_image 라이브러리)
- [ ] memcopy: 64비트 주소 지원

중기 (1-2개월)

- [ ] mobilenet: DW 3×3 HLS IP 구현 (스트리밍 스텐실)

- [ ] mobilenet: PW 1×1 HLS IP 구현 (시스톨릭 어레이)
- [ ] mobilenet: DW→PW 퓨전 (DRAM 트래픽 제거)

장기 (3-6개월)

- [ ] 실제 MobileNetV1 전체 레이어 구현
- [ ] ImageNet 분류 지원
- [ ] DPU (Deep Processing Unit) 통합
- [ ] Vitis AI 프레임워크 활용

7.3 참고 자료

Xilinx 공식 문서:

- UG1137: Zynq UltraScale+ MPSoC Software Developer Guide
- UG643: Standalone BSP Libraries
- UG1399: Vitis HLS User Guide

외부 리소스:

- MobileNet 논문
- 양자화 신경망 가이드
- Xilinx GitHub Examples

부록: 파일 크기 참조

프로젝트	파일	크기	설명
mobilenet	dw3x3_c3.bin	27 bytes	3채널 × 9 = 27 가중치
mobilenet	pw1x1_c10x3.bin	30 bytes	10출력 × 3입력 = 30 가중치
mobilenet	labels.txt	~20 bytes	0-9 숫자 레이블
mobilenet	digit_X.bmp	~3KB	32×32 RGB BMP
memcopy	테스트 버퍼	32KB	8192 워드 × 4바이트

이 보고서는 2026년 2월 5일 기준으로 작성되었습니다.