

## **dombert SoC [PRELIMINARY FACT SHEET]**

### MJPEG Streaming System on Chip

The *dombert* System on Chip (SoC) is the successor of the MIPS based *dorothea* JPEG L2SoC encoder.

It is part of the cCAP (configureable Custom Application Processor) family concept by section5.ch. The cCAP excel in high configureability and low resource usage on FPGA silicon. Independent of the FPGA technology, peripherals and IP cores can be tailored flexibly, starting with low-level register addresses up to number of peripheral instances (such as UART or SPI interfaces) or custom IP cores.

The cCAP family was particularly developed to meet specific requirements of robust and proveable functionality (like for safety relevant environments) while being easy to handle for application programmers at the same time, in order to facilitate handling of FPGA technology. The compact instruction set of the core architecture enables very compact program code as well as customer specific micro code extensions with DSP emulation functionality.

### 1 Functional overview 'dombert'

- 32 Bit processor 'ZPUng' v1.1, three stage pipeline @50 Mhz
- Wishbone-Bus for address decoding and peripherals
- Autobuffer DMA for high speed data I/O
- Programmable in C (GCC)
- Hardware Debugger (JTAG), GDB
- Ethernet, UDP/IP stack

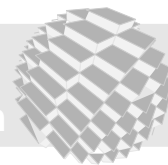
For technical specifications of the standard configuration, see Table 1.

Technical specifications		
<i>Memory configuration</i>		
ROM	256kB	Overlay program memory (SPI) read only
L1RAM	16kB	Level one program and data segment
L1CACHE	16kB	Program memory or ROM cache (configureable)
SRAM	2kB	Dedicated Stack Memory
SPAD	8kB	Two ScratchPad RAM buffers for peripheral data I/O
<i>Peripherals</i>		
SCACHE	1	SPI flash cache
SIC	1	System Interrupt Controller with four peripheral channels and configureable priorities
DMAA	2	Autobuffer DMA for high speed RX/TX
UART	1	UART console
TWI	1	Two wire interface (I2C compatible)
MAC	1	Media Access Control Core (RGMII/MDIO), 100M/1G
VIDEO	1	Optional "Video Hub" for RGB/YUV format conversion
JPEG_L2	1	Low latency JPEG 'two lane' encoder for 720p60 video or custom formats

Tab. 1: dombert standard configuration

### 2 JPEG encoder

The JPEG encoder engine is able to process YUV video at pixel frequencies up to 100 MHz in the *dombert* configuration and stream them to a host system that understands a RFC2435 compliant MJPEG RTP stream. It can operate on a 100M link at lower resolutions, however for full error free high quality transmission, a 1G setup is required. See details in Table 2.



Feature	Property
Input format	YUV422 or dual tap monochrome
Input width (max.)	2048
Buffering	8 scanlines per channel
Streaming	Raw JPEG (MJPEG/RTP and JFIF compliant)
JFIF (JPEG file format) generation	Microcontroller interface (non-hardcoded)
Huffman tables	fixed, non-configurable
Failure/Error resilience (Restart Marker)	N/A

Tab. 2: jpeg\_core 'dombert' configuration

### 3 Reference setup

The reference environment for the dombert SoC is based on a Linux gstreamer toolchain for low latency video streaming. Users can develop custom applications or image processing plugins based either on the gstreamer SDK or include gstreamer capabilities into own programs. Guidance for gstreamer based software development can be supplied on request.

### 4 Simulation

The entire *dombert* SoC can be co-simulated in a virtual machine. Advantages:

- Cycle-accurate verification of correct functionality
- Efficient visual debugging in trace waveforms, optional on-chip trace debugging via Ethernet
- Co-Simulation with Python-Scripts for functional verification

### 5 Example applications

- Motion JPEG over RTP/UDP streaming (IP camera)
- Real-Time Network stress testing

### 6 Reference-Platforms

The dombert SoC can be supplied as net list package for the following platforms:

- Lattice ECP5 Versa Eval board
- HDR60 ECP3 based video eval board
- TE0600 (Spartan6 based) core module

A few resource usage examples are displayed in the tables below. The number of used EBR (Block RAM instances) can be reduced if no extended buffering is required.

Module ID	MULT18X18D	ALU54B
JPEG	16	0
CPK	9	3
(Total SoC)	25	3

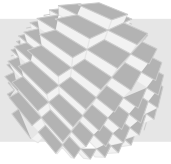
Tab. 3: Required DSP components

Lattice ECP5	
SLICE	6172
Registers	4896
LUT4	6803
EBR	107

Tab. 4: Resource usage ECP5 'test bench'

Lattice ECP3	
SLICE	5871
Registers	4698
LUT4	8576
EBR	108

Tab. 5: Resource usage ECP3 'ipcam'



## 7 HDR60 demo

This demo is based on the standard 'off the shelf' configuration of the HDR60. It demonstrates the compression and transmission over a RFC2435 compliant MJPEG UDP video protocol. A Linux set up is required for the default test.

Note that 1Gbit connections are not yet supported. Make sure your Ethernet port is running in 100MB mode. When the left LED on the connector is dark orange, the port is configured for 100 MB. When in 1G mode, it lights up green/yellow. To force an ethernet interface to 100M mode, you may have to install the **ethtool** and configure your MAC as follows:

```
sudo ethtool -s eth1 speed 100 duplex full autoneg off
```

### 7.1 Usage

1. Start gstreamer client on host (see configuration below)

2. Connect to UART console of target using minicom:

```
minicom -o -D /dev/ttyUSB1 -b 115200
```

3. Configure target host in the 192.168.0.x network, like

```
r 192.168.0.2
```

Make sure to check if the target host responds, like:

```
Send ARP request...
```

```
# Got ARP reply: 3c aa aa 15 ce 34
```

4. Run (preset) JPEG stream @20fps:

```
j 0
```

Currently, there are the following preset slots for the 'j' command:

- |   |  |
|---|--|
| 0 | 1024x768@20fps, quality 90%                            |
| 1 | 1024x768@10fps, quality 96%                            |
| 2 | 1024x768@35fps, quality 80% (network limit)            |
| 3 | 1280x960@35fps, quality 75% (network limit)            |
| 8 | 1280x960@10fps, quality 80%, test pattern (see Fig. 1) |

It is possible that some scenes cause FIFO overflows in the high bandwidth settings, for example the texture of a colourful fabric can have a similar effect as the pseudo random JPEG stress pattern.

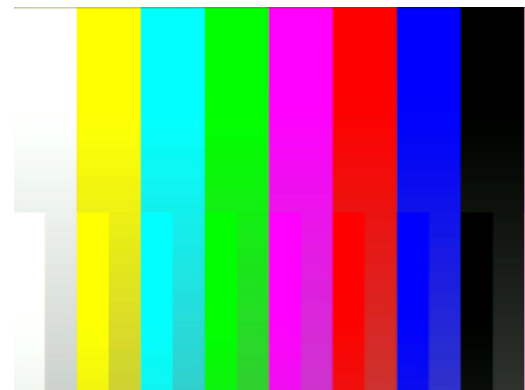


Fig. 1: 1280x960 sensor test pattern

### 7.1.1 GStreamer configuration

To display a test video stream, launch a gstreamer RTP pipeline:

```
#!/bin/bash

caps="application/x-rtp, media=(string\)video,"
caps="$caps clock-rate=(int\)90000,"
caps="$caps encoding-name=(string\)JPEG"

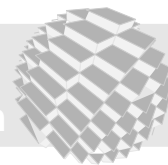
gst-launch-0.10 -v gstrtpbin name=rtpbin udpsrc \
  caps="$caps" \
  port=2020 \
  ! rtpbin.recv_rtp_sink_1 rtpbin. \
  ! rtpjpegdepay \
  ! jpegdec \
  ! fpsdisplaysink text-overlay=false \
  video-sink=autovideosink \
  signal-fps-measurements=true
```

To launch the test manually from the UART console shell on the FPGA SoC target, use the 'j' command. Note that arguments are in hexadecimal notation and should be a multiple of 16 (trailing '0'), otherwise they will be truncated. This command runs a JPEG stream at 1024x768:

```
j 400 300
```

You can tweak the framerate by changing the value of register `Reg_MFR_FRAME_LENGTH_LINES (0x300a)`, see also parameters below. Likewise, you can modify the `Reg_MFR_LINE_LENGTH_PCK (0x300c)` register to experiment with the framerate on a grainier level.

It may be required to restart the gstreamer script when the video parameters are changed. Some settings appear to cause the sensor to hang. If the system stops streaming and locks up, i.e. turning off and back on using 'j' has no effect, press the reset button on the HDR60 to reinitialize.



## 7.2 Sensor parameters

Using the 'i' command, you can control the MT9M024 sensor registers. When specifying only the register address as argument, its value will be dumped:

```
# i 3082  
> 0028
```

Examples:

```
i 3070 3  
Turns on the test pattern
```

```
i 3082 0  
Turn off HDR
```

```
i 3012 80  
Modify exposure time
```

```
i 300c 800  
Change line timing (frame rate). Too high frame rates may cause FIFO overflow warnings on the console:
```

```
[FIFO]
```

## 7.3 Command line parameters

*All numeric parameters are given in hexadecimal values*

```
r 192.168.0.<host_digit>  
Set receiver host by ip (must be in 192.168.0.x network)
```

```
q <quantisation factor in hex>  
Set quantisation parameter from 1 (best quality) to 0x50 (worst)
```

```
j <preset>  
Call preset from 1-5. These are predefined test values for quality and resolution. If no preset is entered, the default will be used.
```

```
j <x> <y>  
Run stream with resolution x and y with the current quality setting
```

*The resolution in x is limited to maximum 1024*

```
j Turn off stream when running
```

```
i <reg> [<data>]  
Get or set sensor value
```

```
is Initialize sensor default values (reset). Don't run while JPEG streaming.
```

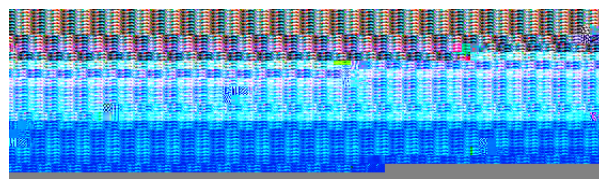


Fig. 2: FIFO full scenario

## 8 Diagnostics

The JPEG L2 SoC can be configured as test bench for FIFO and encoder stress tests to determine empirical compression limits. This is achieved by the so called 'stress pattern': A marching pseudo random pattern with a periodicity that results in a very bad compression rate and high load in the encoder pipeline. Details are found in the dombert SoC hardware reference.

### 8.1 Error scenarios

Errors may occur during streaming of the highly badly compressible LFSR pattern (stress test). If the bandwidth of the Ethernet port (when in 100M mode) is not sufficient, FIFO overflows will be signalled and the image will look broken as in example Fig. 2.

The displayed console (UART) errors in detail:

```
[FIFO]
```

Denotes that FIFO overrun flag was set. This occurs when the FIFO between JPEG and DMA can not take more data when the network MAC TX is busy. Increase the "quantisation" value ('q' command) or reduce resolution.

```
[DEMUX]
```

Can only fail on misconfiguration of the encoder or at the very beginning of a sequence, if the image stream is not started synchronously.

```
[DCT_ERR]
```

Can only fail when input data is faulty.

```
[DCT_OVER]
```

Can only show on misconfiguration of the encoder (wrong image width or wrong number of MCUs specified) or at the beginning of a broken frame.

### 8.2 Packet stream analysis

The JPEG L2 SoC testbench reference is configured with a few presets that demonstrate streaming via a 100MBit link, optionally 1GBit on the TE0600 based platforms. There are scenarios where the generated demo image may show broken for the following reasons:

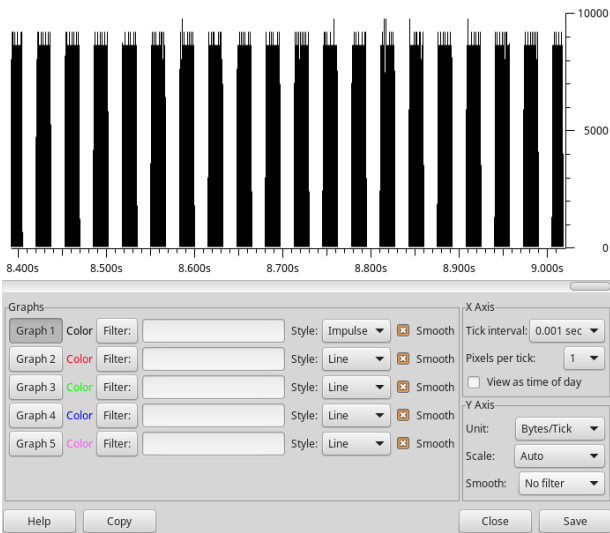
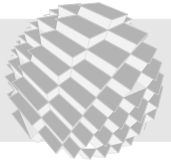


Fig. 3: Sane packet burst at approx 30 fps

- Packets are lost:
  - Packet drop on router side
  - Network congestion or interface issues
- FIFO overrun on the encoder side has occurred
  - Network is busy due to other network devices creating traffic
  - JPEG quality bandwidth excess (see previous paragraph)

To analyze the network packet stream, it is recommended to use the **Wireshark** (former Ethereal) analyzer on a Linux platform.

Fig. 3 shows a typical 'stress pattern' packet stream during a flawless JPEG transmission on a 100M link without interference by other network participants. The physical maximum of approx. 10'000 Bytes per tick (0.001s) during the packet bursts is not fully reached, leaving some headroom for the output FIFO.

When a width of a multiple of 336 (e.g. **j 150 200** on the console) is used, the stress pattern does not march and is the same for every line as display in Fig. 4, i.e. the compression is more effective. The corresponding packet sequence is display in Fig. 5.

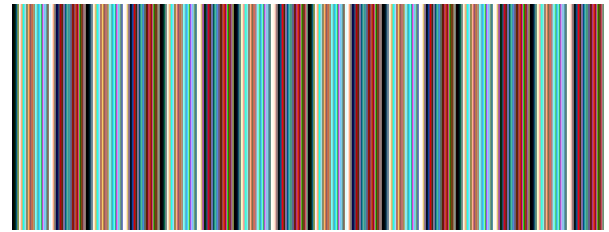


Fig. 4: Regular pattern at x=672 (0x2a0)

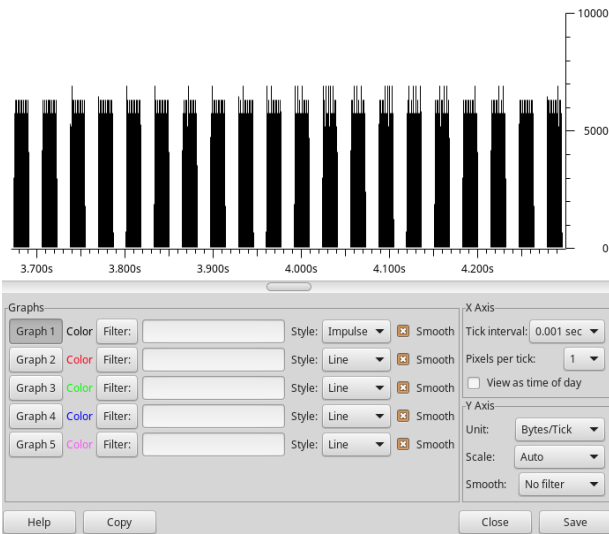


Fig. 5: Low bandwidth packet burst at approx 30 fps

## 9 Further pointers

[RFC2435] RTP Payload Format for JPEG-compressed Video

<http://www.faqs.org/rfcs/rfc2435.html>

Web link [german only]

<http://section5.ch/index.php/dienstleistungen/custom-cpu-solutions/>

dombert Hardware Reference soc-dombert.pdf  
On request only (NDA required)