Do It Faster:

How We Supercharged Linux to Work With Blazing Fast ADCs for IIO

Open Source Summit Europe, August 2025

Agenda

Intro

ADCs, Resolvers, and DACs (Oh, My!)

SPI Offloading

PWM Overload(ing)

ADCs at Full Speed

Reflecting: Developer Q&A

Conclusion

Intro

About BayLibre

- Services company specializing in all things embedded - e.g. the Linux Kernel, U-Boot, Toolchains, Yocto, Zephyr, hardware design
- Based in Nice, France, but have a global presence (US, Canada, UK, Germany, Italy, Taiwan, and more)
- Linux Foundation member (Silver)
- Many open-source contributors and maintainers
- https://baylibre.com/

About Me

- Embedded Linux developer
- Based in Ottawa area, Canada
- Maintainer for parts of Yocto Project
- Occasional Python and/or CI developer - now kernel work, too
- Joined BayLibre in April 2023

ADCs, Resolvers, and DACs (Oh My!)

Background

- **Mid 2023:** BayLibre begins contract with Analog Devices, Inc.
  - Driver development, subsystem improvements, documentation
  - Related support in Python, MATLAB
- **Team effort**, worked closely with ADI developers and upstream maintainers (10+ dev-years)
- **BayLibre is remote-first** - some hardware on individuals' desks, but we used shared hardware and lab infrastructure (located in Nice) to do the bulk of work
- **See also:**
  - **David's** ELC @ OSS NA 2024 talk - AD2S1210 driver work, **overview of aforementioned remote infrastructure**
  - **Uwe's** ELC @ OSS EU 2024 talk - **PWM subsystem** overview, challenges, roadmap
  - **David's** ELC @ OSS NA 2025 talk - **SPI Offloading development** (briefly covered here)

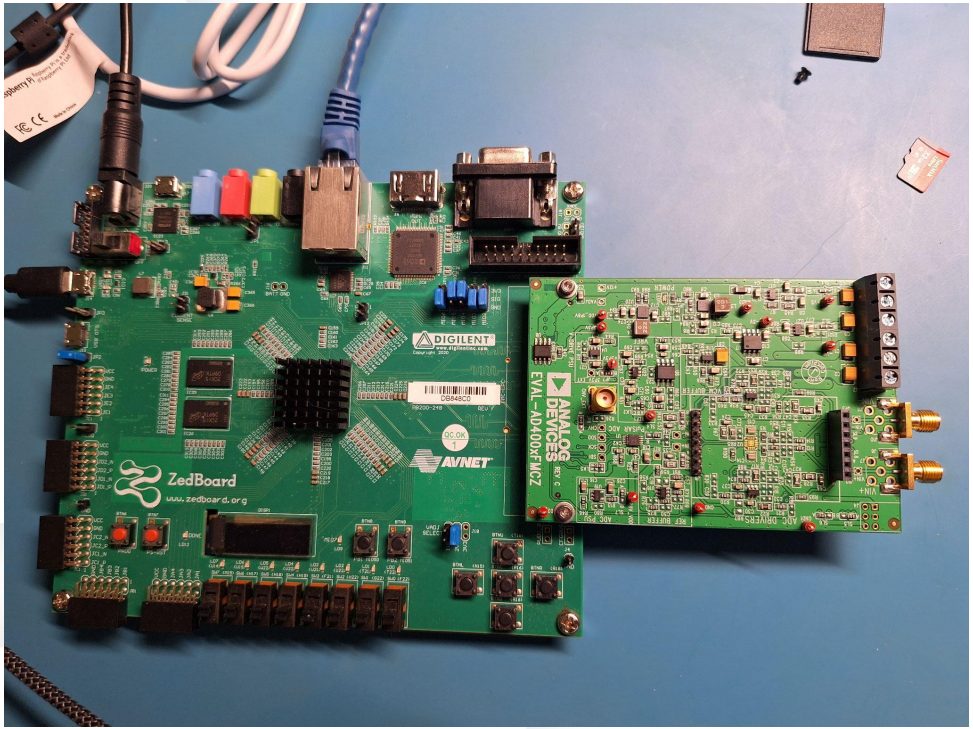
Target Devices

- Each device family featured a matrix of features, including:
  - Resolution (bits)
  - Max sample rate
  - Differentiality
  - Calibration
  - Clocks
  - etc.
- This is just a subset - **121 devices in all!**

				Sample data		sample rate	# ch		
Genus	Species	Part #	Eval board	bits	signed	max	in	out	differential
		AD7177-2		32		10 kSPS	4		mixed full differe
	Sigma-Delta ADC with PGA and Reference	AD7124-4		24	config	19.2 kSPS	4	2	mixed differential, pseudo
		AD7124-8		24		19.2 kSPS	8	2	differential,
	Sigma-Delta ADC with PGA	AD7190		24		4.8 kHz	8		differential / pseudo differential
		AD7192		24		4.8 kHz	8		
		AD7193		24	config	4.8 kHz	8		
		AD7194		24		4.8 kHz	8		
		AD7195		24		4.8 kHz	8		
SPI wihtout registers	Sigma-Delta ADC with PGA	AD7191		24	s	120 Hz	3		full differential
SPI wihtout registers	Simultaneous Sampling ADC	AD7605-4		16	s	300 kSPS	4		Single ended
SPI wihtout registers	Simultaneous Sampling ADC	AD7606	EVAL-AD7606CFMCZ	16	s	200 kSPS	8		Single ended
SPI wihtout registers	Simultaneous Sampling ADC	AD7606-4		16	s	200 kSPS	4		Single ended
SPI wihtout registers	Simultaneous Sampling ADC	AD7606-6		16	s	200 kSPS	6		Single ended
SPI wihtout registers	Simultaneous Sampling ADC	AD7607		14	s	200 kSPS	8		Single ended
SPI wihtout registers	Simultaneous Sampling ADC	AD7608		18	s	200 kSPS	8		Single ended
SPI wihtout registers	Simultaneous Sampling ADC	AD7609		18	s	200 kSPS	8		True Differential
SPI wihtout registers	PulSAR ADC in LFCSP	AD7944	EVAL-AD7944FMCZ	14	u	2.5 MSPS	1		Pseudo-differential/ground sense
SPI wihtout registers	PulSAR ADC in LFCSP	AD7985	EVAL-AD7985FMCZ	16	u	2.5 MSPS	1		Pseudo-differential/ground sense
SPI wihtout registers	PulSAR ADC in LFCSP	AD7986	EVAL-AD7986FMCZ	18	s	2 MSPS	1		True differential
LVDS	Fast PulSAR	AD7625	EVAL-AD7625FMCZ	16	s	6 MSPS	1		Differential (Antiphase)
LVDS	Fast PulSAR	AD7626	EVAL-AD7626FMCZ	16	s	10 MSPS	1		Differential (Antiphase)
LVDS	Fast PulSAR	AD7960		18	s	5 MSPS	1		Differential (Antiphase)
LVDS	Fast PulSAR	AD7961		16	s	5 MSPS	1		Differential (Antiphase)

Hardware Setup (Home)

- **Dev Board:** Digilent Zedboard running Kuiper Linux OS
- **Add-on:** Various ADI evaluation boards, e.g. EVAL-AD400x-FMCZ (shown)
- **Test Tools:** DS1054Z, SDG 1032X, Saleae Logic 8
- **Power Control (Post-Project):** HS103 controlled via labgrid fork (see also my next talk in G104)
- **Note:** Some other devs also worked on the DE10 Nano, used Debian



Hardware Setup (Office)

- **Dev Board:** Digilent Zedboard running Kuiper Linux OS
- **Add-on:** Various ADI evaluation boards
- **Test Tools:** ADALM2000 - handy all-in-one oscilloscope, logic analyzer, signal generator, etc.
- **Power:** BayLibre Copilot
- Also tested some devices (e.g. AD2S1210) on a RaspberryPi (see here)

Software Setup

- **Build Host:** Debian 12 on a Ryzen 7900-based machine
- **HDL (Hardware Description Language) Builds**: Vivado 2023.2
- **Measurement:** IIO Oscilloscope, Scopy (used with ADALM2000)
- Some Bash scripts for setting and getting sysfs device properties during bring-up
- An early version of Uwe's libpwm tooling
- Labgrid (tgamblin/kasa-driver branch) for automatic control of the setup - I added this later (specifically to help prep this talk)

AXI HDL in 60 Seconds

- **AXI: A**dvanced E**x**tensible **I**nterface (see here)
  - Basically, protocol/standard for memory-mapped communication
- **Implement interfaces with HDL in an FPGA - SPI controller, PWM generator, etc.**
- **ADI** maintains a repository of HDL source code for FPGAs written to serve as the bridge between drivers and target devices (i.e. ADCs/DACs), with a setup/usage guide
  - It also contains implementations for things like axi_pwm_gen (more on that soon), which some devices depend on for full performance
- Building a particular project produces files that can be converted into a **BOOT.BIN**, which - along with a **uImage** and a **dtb** - make up the boot files for bringing the Zedboard up

The IIO Subsystem

- **Documentation:** here
- **Maintainer:** Jonathan Cameron
- **Drivers, helpers, core logic for ADCs, DACs**, and other **sensors** (e.g. accelerometers, temperature sensors)
- Often use one or more clocks, PWM signals, SPI/I2C bus, or a combination thereof for full functionality
- **Most devices' state/parameters can be accessed and controlled via sysfs**, e.g. via paths like `/sys/bus/iio/devices/iio\:device0/sampling_frequency`
- The majority of BayLibre's work on device drivers for ADI targeted IIO, but some significant additions had to be made elsewhere…

SPI Offloading

SPI Offloading (Also in 60 Seconds)

- Reminder: David's talk
- **TL;DW: Goal:** reduce burden on Linux host by offloading transactions to dedicated hardware
- **Example:** AD469x ADCs list short time scales (order: **ns**) for commands - e.g. to read a sample and trigger the next one
  - hard to achieve consistently via the CPU if it's doing anything else!
- ADI has the AXI SPI Engine interface (FPGA-based SPI controller) for this - **took ~1 year to upstream all of the matching kernel code**
- With this, we **can achieve 1+ MSPS (million samples per second)**, with idle CPU and direct SPI communication, using the ADCs
  - For faster speeds, use an FPGA + the IIO backend framework

SPI Offloading - Examples of Upstream Usage

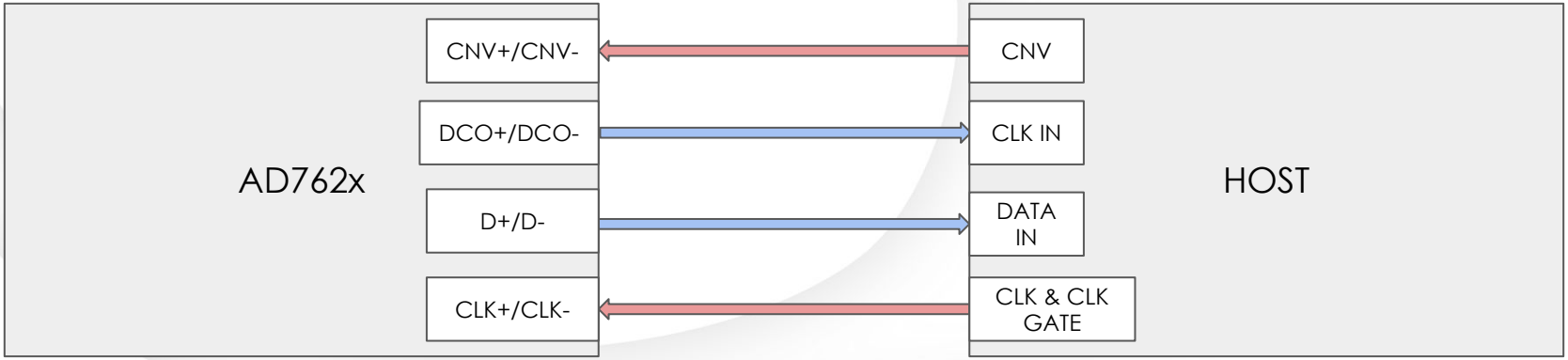
Driver	Number of Devices Supported	Max Sample Rate (MSPS)	Kernel Version	Commit ID*
ad4000.c	14	2	6.16	59b51edf717b
ad4695.c	4	1	6.15	f09f140e3ea8
ad7380.c	18	4	6.15	bbeaec81a03e
ad7606.c	11	1	6.16	e96d35faf357
ad7944.c	3	2.5	6.15	cbc986cda57a

* When SPI Offload support first added to driver

PWM Overload(ing)

Why Care About PWM?

- The AD762x/AD796x device family, which has one of the highest maximum sample rates (**10 MSPS on the AD7626**) - uses PWM signals as **conversion triggers**
- Specifically, it requires two PWMs operating on the **CNV+/CNV-** and **CLK+/CLK-** pins:



- **The kicker: CLK GATE signal needs to be offset from CNV**

AD7626 Echoed-Clock Timing Diagram

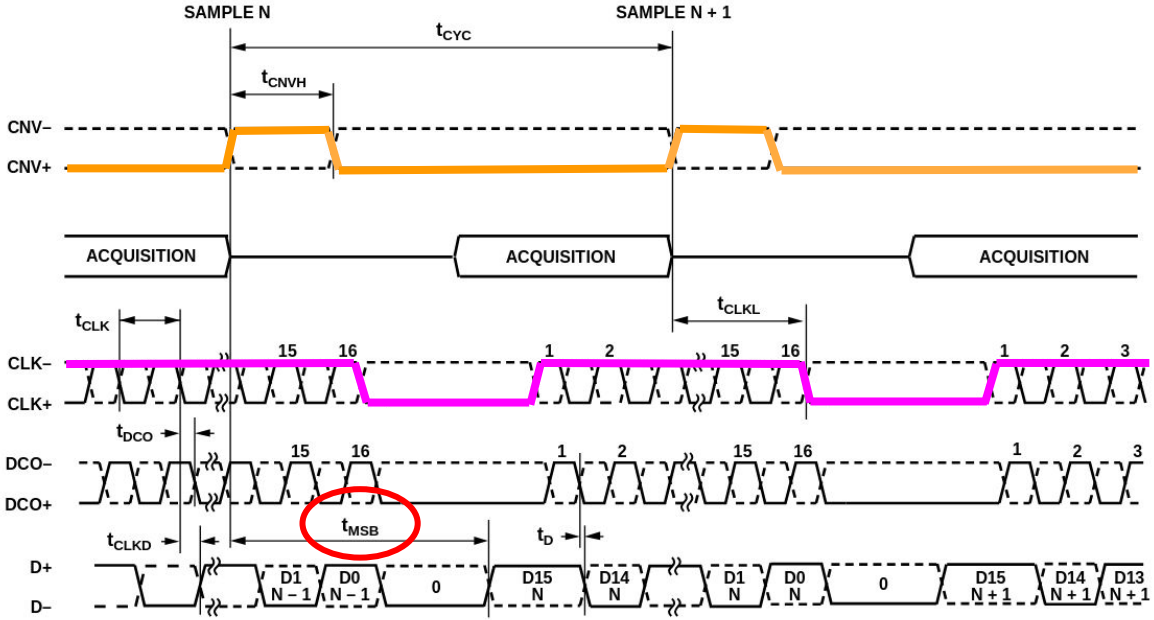


Figure 41. Echoed Clock Interface Mode Timing Diagram

Source: [AD7626 datasheet, page 23]

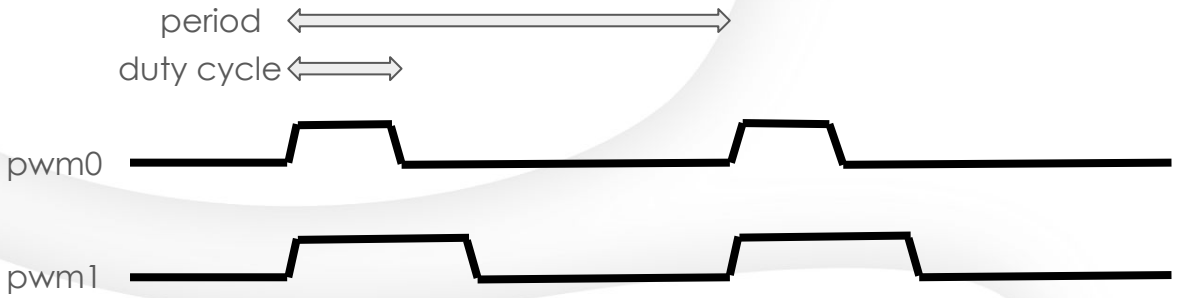
PWM API: State

Two main attributes:

1. **Period**
2. **Duty cycle**

Three secondary attributes:

1. Polarity (normal or inverted)
2. Enable/disable
3. Usage power



PWM: Getting Started

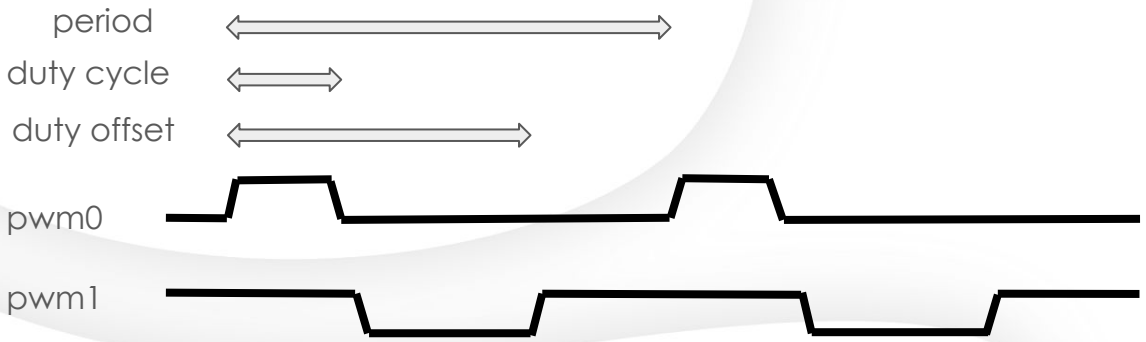
- The steps to support the AD762x/AD796x devices:
  a. Develop a basic driver for the axi_pwm_gen core
  b. Add support for PWM offsets in the PWM subsystem upstream
  c. Refactor the axi_pwm_gen driver to use the new PWM functionality
  d. Finally, write the ad7625 device driver
- **Basic driver landed in v6.11** (commit 41814fe5c782) - followed by a couple of fixes in v6.11 and v6.12
- **Next: modifying the PWM core** - I made an RFC and a v2, before Uwe (PWM subsystem maintainer) joined BayLibre and wrote a more complete overhaul for **v6.13**
  a. This included an **update** to the **axi-pwm-gen** driver with duty offset support

PWM API: Waveform (as of v6.13)

Three attributes:

1. **Period**
2. **Duty cycle**
3. **Duty offset**

Polarity and enable/disable handled within these parameters (disable with period == 0)



PWM waveforms no longer have to line up

Kernel API: Two Ways to Represent PWMs

Reference: pwm.h

```
struct pwm_state {
        u64 period;
        u64 duty_cycle;
        enum pwm_polarity polarity;
        bool enabled;
        bool usage_power;
};
```

```
struct pwm_waveform {
        u64 period_length_ns;
        u64 duty_length_ns;
        u64 duty_offset_ns;
};
```

Both available, but the latter is more expressive!

PWM: Rounding It Out

- See also: Uwe's talk at OSS EU 2024
- PWM/CLK signals are derived from the system clock. **Major hurdle:** rounding waveform parameters correctly based on available hardware
- From patch 2 of 8:

```
The rounding rules that are expected to be implemented in the
round_waveform_tohw() are: First pick the biggest possible period not
bigger than wf->period_length_ns. For that period pick the biggest
possible duty setting not bigger than wf->duty_length_ns. Third pick the
biggest possible offset not bigger than wf->duty_offset_ns. If the
requested period is too small for the hardware, it's expected that a
setting with the minimal period and duty_length_ns = duty_offset_ns = 0
is returned and this fact is signaled by a return value of 1.
```

PWM: Testing Duty Offset

- Method:
  a. Build Zedboard bootfiles (i.e. kernel with axi-pwm-gen driver, updated dtb to match, **BOOT.BIN routing PWM signals to PMOD header, see here**)
  b. Set waveform parameters (with offsets) on the Zedboard using libpwm
  c. Check the outputs with an oscilloscope - **importantly, that waveforms are actually offset!**
- **On the next slide:**
  a. CH1 (yellow) attached to JB2 (pwm0), CH2 (blue) attached to JB3 (pwm1)
  b. **Both PWMs** with **period_length = 20000 ns** (freq = 50kHz), **duty_length = 5000 ns**
  c. **pwm0 duty_offset = 0, pwm1 duty_offset = 8000 ns**

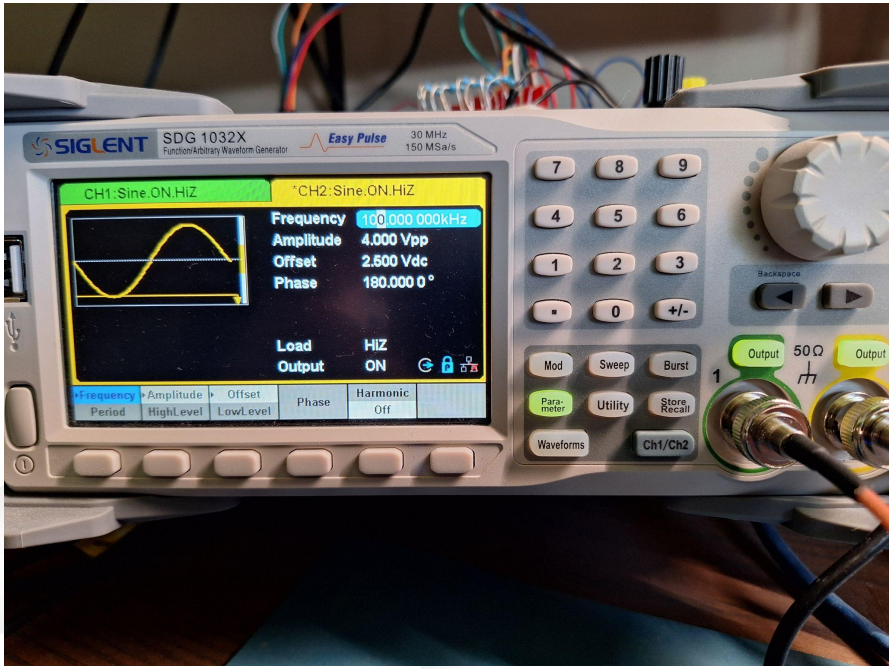
PWM: Testing Duty Offset (Example)

8000ns offset

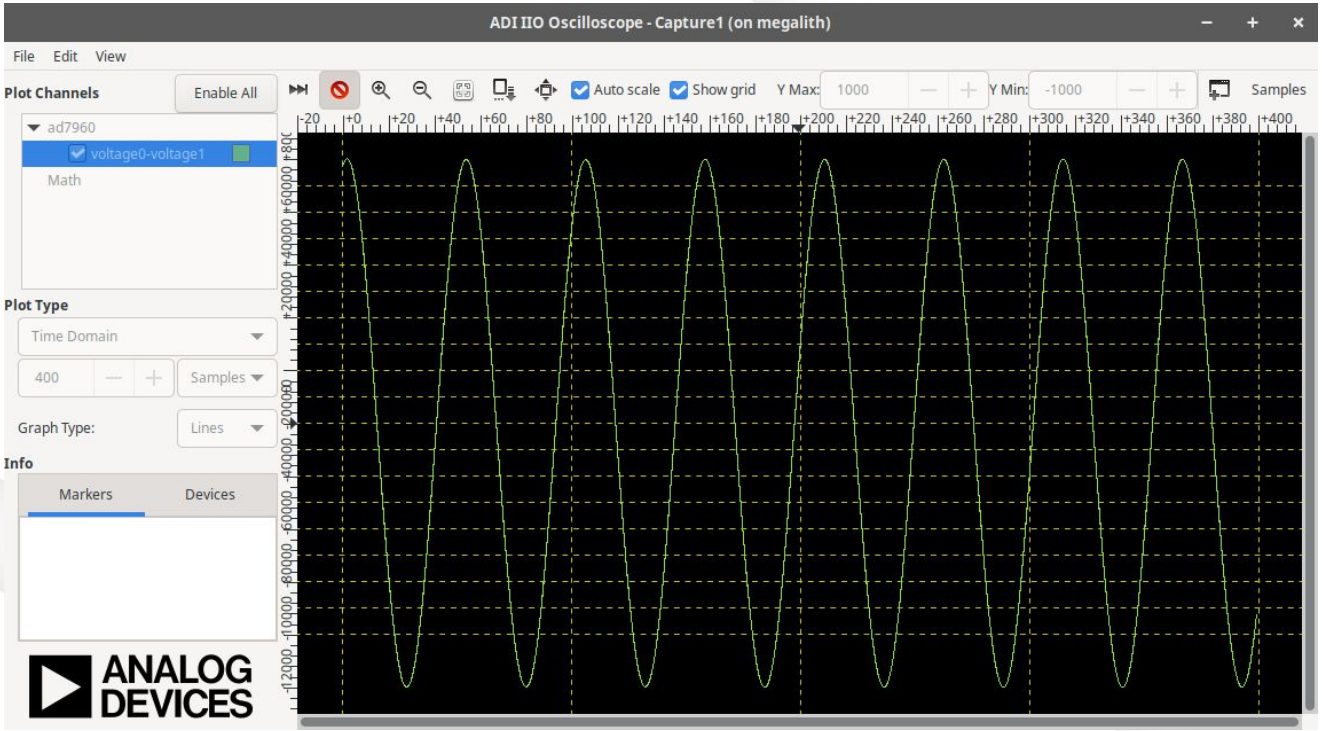
ADCs at Full Speed

Back to the AD762x/AD796x

- With duty_offset support, we could finally write ad7625 driver
- Final patch series (v5) made it into **v6.13**
- **Next slide:** iio_oscilloscope screen capture from 2024, with my my SDG 1032X outputting test signals: 100 kHz, $4V_{p-p}$, 2.5V offset, antiphase



The AD7960 at Work



AD7960: GPIOs and Special Modes

- **Another aspect:** handling voltage references and modes
- **AD762x was simpler** - only had EN0/EN1 (voltage configs)
- Depends on hardware wiring, handled this in the dtb bindings

Table 8. Voltage Reference Options

EN3	EN2	EN1	EN0	REFIN	Reference Mode Description
X[1]	0	0	0	X[1]	Power-down mode. Everything is powered down, including the LVDS interface.
X[1]	0	0	1	0 V	Interface powered up. Reference buffer disabled. An external 5 V reference is applied to the REF pin. Connect REFIN to 0 V in this mode. The bandwidth of the input sampling network is set to 28 MHz.
X[1]	0	0	1	2.048 V	Internal reference buffer enabled. An external 2.048 V reference applied to REFIN pin is required. A buffered 4.096 V reference is available on the REF pin. The bandwidth of the input sampling network is set to 28 MHz.
X[1]	0	1	0	0 V	Internal reference buffer disabled. Drive the REF pins with a 4.096 V external reference. Connect REFIN to 0 V in this mode. The bandwidth of the input sampling network is set to 28 MHz.
X[1]	0	1	1	0 V	Snooze mode.[2] LVDS powers down. The chip is unresponsive to CNV± start pulses. The wake-up time is fast (5 µs) when EN3 to EN0 are set to XX01 or XX10. Ensure that the CNV± start pulse is low when transitioning in and out of this mode.
0	1	0	0	X[1]	Test patterns output on LVDS. The ADC output is not available on the interface.
1	1	0	0	X[1]	Invalid mode.
X[1]	1	0	1	0 V	Reference buffer disabled. Drive the REF pins with a 5 V external reference. The bandwidth of the input sampling network is set to narrow (9 MHz).
X[1]	1	0	1	2.048 V	Internal reference buffer enabled and driving REF pin to 4.096 V. The bandwidth of the input sampling network is set to narrow (9 MHz).
X[1]	1	1	0	0 V	Reference buffer disabled. Drive the REF pins with a 4.096 V external reference. The bandwidth of the input sampling network is set to narrow (9 MHz).
X[1]	1	1	1	0 V	Snooze mode.[2] LVDS powers down. The chip is unresponsive to CNV± start pulses. The wake-up time is fast (5 µs) when EN3 to EN0 are set to XX01 or XX10.

[1] X = don't care.
[2] The snooze mode is not useful when the internal reference buffer is used because the fast wake-up is not possible due to the settling of the internal reference buffer.

AD7960: Handling Modes in the Driver

- Corresponding code in ad7625.c: several booleans for states the device can be in, based on dtb settings

```
        /*
         * Optional GPIOs for controlling device state. EN0 and EN1
         * determine voltage reference configuration and on/off state.
         * EN2 controls the device -3dB bandwidth (and by extension, max
         * sample rate). EN3 controls the VCM reference output. EN2 and
         * EN3 are only present for the AD796x devices.
         */
        struct gpio_desc *en_gpios[4];
        bool can_power_down;
        bool can_refin;
        bool can_ref_4v096;
```

- And some functions for checking these before setting the device up, e.g.:

```
static int ad7625_parse_mode(struct device *dev, struct ad7625_state *st,
                             int num_gpios)
{
```

Reflecting: Developer Q&A

The Query

I asked some of my coworkers on the project (**David Lechner, Uwe Kleine-König, Esteban Blanc**) four questions:

1. **What kernel drivers did you add/modify?**
2. **Did you contribute to the kernel in other ways as part of this project?**
3. **What was the most memorable part of the process?**
4. **What was the biggest challenge for you during development?**

Here's what they said…

Response Summary (Part 1)

What kernel drivers did you add/modify?

- **New drivers:** ad4030, ad4630, ad7124 (DE10 Nano), ad7944, ad7380 (assist: Julien Stephan), ad4695 (assist: Trevor Gamblin)
- **New features/improvements:** ad7606, ad7173, ad_sigma_delta

Did you contribute to the kernel in other ways as part of this project?

- **SPI offloading support** and some other core SPI stuff
- **PWM Core** updates
- Added some new **kernel-wide convenience APIs** for regulators and gpio arrays
- Various janitorial efforts

Response Summary (Part 2)

What was the most memorable part of the process?

- Getting involved with the kernel community again **(David)**
- Lessons Learned **(Uwe):**
  - There are corners in Linux that don't live up to standard of being solid, well-reviewed
  - Devices that work well on one platform might not on another
- Uncertainty with common byte channel implementation for the ad4030/ad4630 **(Esteban)**

What was the biggest challenge for you during development?

- Identifying similarities between parts to make the best development choices **(David)**
- IIO subsystem was new to me **(Uwe)**
- Keeping morale up when my driver's performance was below theoretical maximum **(Esteban)**

Future Work

- More ADI device support - some work still ongoing
- **Personal goal:** Enabling more streamlined kernel dev workflow using the BeaglePlay and MIKROE boards like the PWM 2 Click and Waveform Click (also see my next talk in G104)
- **Personal goal #2:** More kernel work in general
- Further improvements/additions to subsystems - e.g. IIO trigger API

Special Thanks

- Analog Devices (especially Nuno Sá, Michael Hennerich)
- David Lechner, Uwe Kleine-König
- The BayLibre ADI team

Thank You

Questions?