Whiznium Code Generation/Iteration Services

Use Case: FabSight Monitoring Of Industrial Appliances

Quick facts

- This example highlights the vast capabilities of WhizniumDBE and WhizniumSBE covering all aspects of modern IIoT software development from the lowest hardware level to data collection in the cloud for big data analytics.
- A non-invasive current/voltage (I/V) probe on the power line of a portable beer cooling box along with a standard FPGA board configured as an oscillicsope are used to perform transient and steady-state analysis of the device's state.
- Together, the FabSight.Device WhizniumDBE and FabSight.BeerCooler WhizniumSBE projects collect raw data on an ARM-based edge device running Linux, and perform basic pattern recognition. The derived insights are stored in a local database and are made available through an interactive web-based dashboard. To help during the development stage, real-time interaction with a Windows/.NET machine learning toolkit is implemented using the API capabilities of WhizniumSBE.
- FabSight.Analytics, the third Whiznium-developed tool in this context, serves as cloud-based counterpart to FabSight.BeerCooler. Via its API, it accepts secure HTTPS connections from potentially multiple FabSight edge devices, allowing them to synchronize their historical data e.g. for big data analytics.

Introduction

Numerous challenges need to be overcome to get many of today's IIoT applications off the ground, from non-invasiveness to the system under investigation, to limited edge device computing power and low bandwidth of the data link to connected cloud services.

The FabSight project as shown in Figure 1 demonstrates how each of these possible obstacles and bottlenecks can be circumvented.

The project's overall mission is to determine a target device state only by measuring the current and voltage on its power supply line, signals which are readily available even for legacy industrial appliances. Detection of transients and spikes requires sampling and A/D conversion at up to 1MSPS per line. The corresponding data rate to the host embedded system is greatly reduced by performing functions such as triggering, peak detection and spectral decomposition already on FPGA level. WhizniumDBE is used for the corresponding FabSight.Device RTL project *Devsfcd*, implementing a device command set which allows to parametrically adjust the FPGA acquisition process to each specific deployment.

The host system runs the *combined daemon* FabSight.BeerCooler / *Sfbccmbd*, a dedicated multi-threaded Linux executable developed using WhizniumSBE. It acquires e.g. transient I/V time series or spectra, and stores its various features into a local SQLite database as *raw data*. Owing to the event-driven architecture of the code, storing sets of *raw data* triggers the execution of various state detection algorithms which write their results as time-stamped *insights* into the same database. FabSight.BeerCooler communicates to the outside world via a web-based HMI and an API library, standard features in any WhizniumSBE project.

Finally, FabSight.Analytics / *Sfbacmbd* is a cloud-based WhizniumSBE-developed *combined daemon* which replicates the data model of FabSight.BeerCooler, with only slight adaptations such as multi-source capability and MySQL storage. The HTTPS edge-to-cloud synchronization process between both tools can be configured to periodically push just enough *raw data* or *insights*, so that mea-

Table 1: The FabSight.Device DBE project in numbers

Metric	Value
modules	21
… of which controller	2
… of which LogiCORE	3
… of which memory	7
source files	
RTL project	16
device access library	2
FPGA utilization	
LUT	5968
LUTRAM	343
FF	7188
BRAM	8.5
DSP	16
dev. access library size	500kB

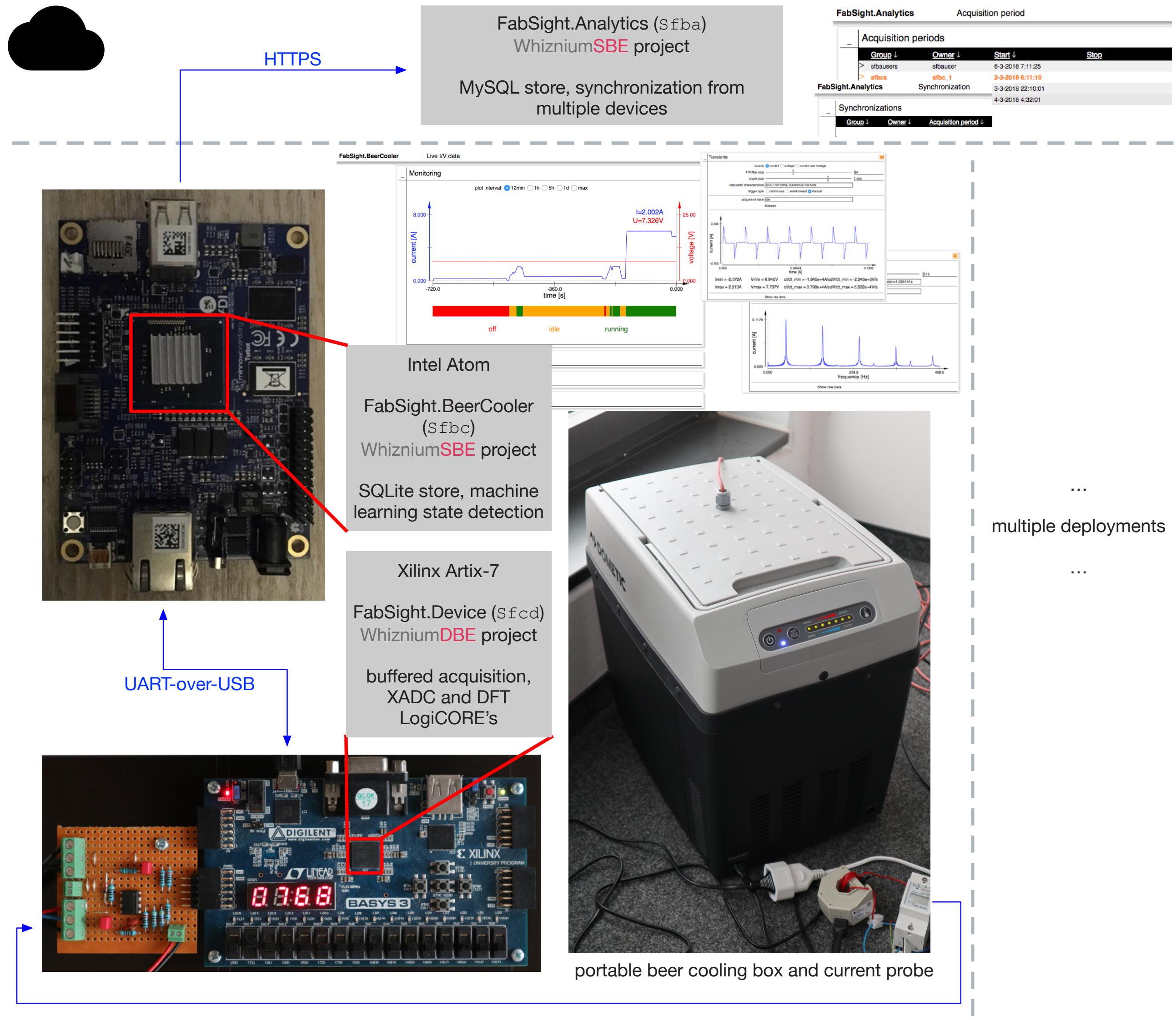

Figure 1: The individual FabSight hardware and software components of the project

ningful big data analytics can be performed.

Programmable logic

The commercial evaluation board chosen (Digilent Basys3) is built around a Xilinx Artix-7 series FPGA. This type of device features on-chip ADC's, so that the only external circuitry required is for I/V level translation and anti-alias filtering.

Following the Whiznium**DBE** development methodology, the FPGA is subdivided into *controllers*, functional entities serving specific purposes. Only two *controllers* are needed here, `tkclksrc` to provide accurate time stamps based on a 10kHz clock and `xadcacq`, performing the actual acquisition task. Moreover, six read-only 2kB buffers based on FPGA BlockRAM are defined to provide bulk data transfer of acquired data to the host system.

To complete the *basic device definition* and with that the VHDL module hierarchy (below `xadcacq`), further relevant instantiations include Xilinx's XADC and DFT LogiCORE's.

As part of the *detailed device description*, a command set is specified (see Figure 2) and a number of finite state machines (FSM's) are included in `xadcacq`, see Table 2 for their purpose.

While FSM implementation is a manual task, representing the project-specific IP, Whiznium**DBE** uses the model information provided to generate all FPGA-side wiring. This includes a UART, CRC secured, host interface along with the *Devsfcd* C++ device access library ("easy" implementation) for the host.

```
void tkclksrc_getTkst(uint& tkst); // get 10kHz time stamp

// get low-pass filtered long-duration average I/V value
void xadcacq_getLpVal(usmallint& lpvalI, usmallint& lpvalV);

// set spectrum acquisition characteristics: rng={on/off}, FIR filter, I vs. V
void xadcacq_setSpec(const bool rng, const utinyint tixVFir, const bool vNotI);

// read from current (I) trace buffer, B part of A/B "ping-pong"
size_t readTrcibbufFromXadcacq(unsigned char* buf, size_t reqlen);
```

Figure 2: *Devsfcd* command set (excerpt) as seen from host

Table 2: `xadcacq` controller FSM's along with their respective tasks

State machine	Purpose
`acq`	interface to XADC LogiCORE ; 2-channel read-out
`lp`	low-pass filtered / long-duration average I/V value
`spec`	interface to DFT LogiCORE ; write to spec{re/im}buf
`spec{re/im}bufB`	burst read to host interface from spec{re/im}buf
`trc`	trace recording to trc{i/v}{a/b}buf; continuous vs. auto/manual triggers; peak detection
`trcbuf`	trc{i/v}{a/b}buf "ping-pong" buffer management
`trc{i/v}bufB`	burst read to host interface from trc{i/v}{a/b}buf

Data model

A versatile data model was conceived which can be used for any application that performs analytics based on live sensor data. Figure 3 shows how *raw data* (`TblSfbcMData`), filled in by acquisition *jobs* (see below), is cleanly separated from *insights* (`TblSfbcMInsight`), filled in by analytics *jobs*. Actual information, both single text/ numeric values or Base64-encoded binary data can be stored along with a μs-precision time stamp underneath *raw data* and *insights*, respectively. WhizniumSBE uses the data model to implement SQLite and MySQL database wrappers along with a default web-based HMI for data view and manipulation.

Sfbccmbd job hierarchy

As with all WhizniumSBE-developed projects, *jobs* responsible for HMI features (named `Crd…`/`Pnl…`/`Dlg…`) are generated automatically. All other functionality is handled by customly specified *jobs* which appear both in the source code tree and in the run-time *job* hierarchy of *Sfbccmbd*. In this project, those *jobs* are categorized further into *source jobs*, *acquisition jobs* and *analytics jobs*. They are related as shown in Figure 4. Most custom *jobs* use WhizniumSBE's master/ slave (M/S) feature, permitting multiple run-time instances of the same *job* with only one being in charge.

`JobSfbcSrcSfcd` uses *Devsfcd* to access the FPGA hardware. Some low-level functions such as time stamp read-out are made available directly, and others are adjusted only by calibration information, performing the translation from raw ADC values to currents/ voltages. The *job* also offers high-level functions such as waiting for and retrieving time-stamped I/V traces.

The trigger source *job* `JobSfbcSrcTrigger` uses the system clock to initiate low-speed acquisition tasks, such as `JobSfbcAcqIv` which retrieves and stores a time-averaged I/V pair on each trigger, typically using a 1s interval.

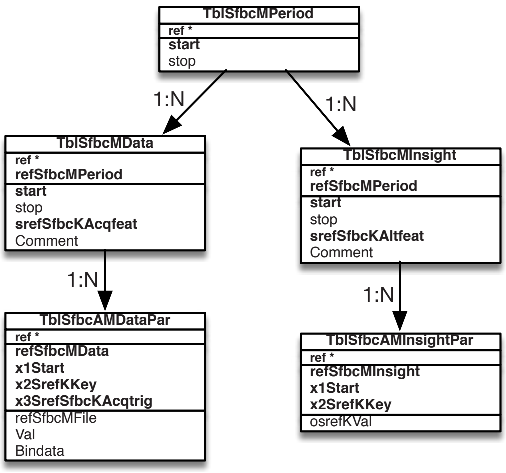

Figure 3: Data model (relevant tables)

The key *job* for obtaining I/V "oscilloscope" traces is `JobSfbcAcqIvTrans`. It can be used to configure various triggers (threshold values for levels and step heights) along with trace lengths and on-FPGA FIR averaging. Once a trace is obtained, statistical features (min/max/avg/var/…) are extracted: this information in many cases is sufficient for state characterization.

`JobSfbcAcqIvSpect` receives I/V spectra as real/imaginary ADC raw values and transforms them into calibrated amplitude vs. frequency series. Again, on-FPGA FIR averaging can be configured.

Finally, `JobSfbcAcqIvStream` allows recording seamless long-duration traces. This functionality is useful for the training of machine learning algorithms.

Currently a single analytics job, `JobSfbcAltIvState`, gets notified on arrival of new I/V trace data (acquired by `JobSfbcAcqIvTrans`) and uses various indicators in both time and frequency domains to determine the machine's state.

`JobSfbcSfbasync` fulfills a special role outside of the source-acquisition-analytics schema: it makes use of the FabSight.Analytics API library to match new data with already synchronized data, and perform the required updates remotely.

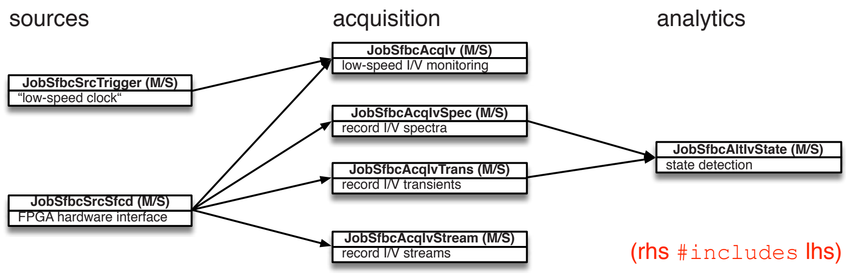

Figure 4: Dependencies between source, acquisition and analytics *jobs*

Table 3: The FabSight.BeerCooler SBE project in numbers

Metric	Value
database tables	43
… of which model	24
… of which query	19
UI modules	2
UI cards	11
source files	
database	93
combined engine	425
web-based UI	673
API	233
binary sizes	
database library	19.4 MB
combined engine	48.5 MB
API library	55.7 MB

Live data display

Instant visual feedback of the machine's I/V input and state are given in the form of a web-based HMI, also included in Figure 1. It combines standard WhizniumSBE features with custom HTML/SVG graphs.

Updates to the HMI views are event-triggered by the acquisition *jobs* described above. Manual acquisition of traces, spectra and long-duration streams can be commanded from within the UI as well.

Windows/.NET interaction

As part of the customer's specifications, an external Windows analytics tool was required to get access to live data. WhizniumSBE's accessor app development feature was used to generate the needed .NET C++/CLI code.

WhizniumSBE's API philisophy is that each HMI and M2M interaction are equivalent in terms of XML data exchanged.

This fact, along with the clean separation between acquired *raw data* on one hand and *insights* on the other hand imply a simple workflow for any analytics task external to *Sfbccmbd*:

1. log in / start a session, 2. "observe" the automatically generated value panel on the *raw data* card waiting for new rows, 3. analyze that data and 4. write back derived *insights* via the "add data" dialog on the *insight* card.

This last point is illustrated in Figure 5, where JSON *insight* values are added manualy, a task performed by the .NET accessor app through *Sfbccmbd*'s API in the application discussed here.

Cloud synchronization

Finally, *raw data* and *insight* collection from possibly multiple embedded deployments is one of the prerequisites for big data analytics, which is handled by copying information from *Sfbccmbd* to an instance of *Sfbacmbd* running in the aws cloud.

In analogy to above .NET case, the API workflow follows the manual workflow with *raw data* and *insight* value add dialogs, and *Sfbccmbd* `#includes` the *Sfbacmbd* API library.

In remote or autonomous deployment scenarios, the cominication channel can be constrained by sporadic availability, data rate and cost for bandwidth. This is particularly the case if mobile, e.g. 4G/LTE services are required.

For this reason, and as not all raw data and insights are required for analytics, the synchrnoization procedure can be configured with detailed filtering options and variable periodicity.

Table 4: The FabSight.Analytics SBE project in numbers

Metric	Value
database tables	48
… of which model	25
… of which query	23
UI modules	2
UI cards	9
source files	
database	104
combined engine	374
web-based UI	619
API	231
binary sizes	
database library	18.5 MB
combined engine	36.9 MB
API library	52.7 MB

Figure 5: JSON entry of insight values, template for API-based procedure

More information

- FabSight YouTube video, MPSI Technologies 2018.

MPSI TECHNOLOGIES
contact@mpsitechnologies.com
+49 175 918 5480
www.mpsitechnologies.com
Helene-Mayer-Ring 4, 04.19
80809 Munich
Germany