AN0063: Driving Electronic Paper Displays (E-paper)

This application note shows how to drive an Electronic Paper Display (EPD) with an EFR32xG22-based Silicon Labs Wireless Kit. The application note makes use of an EPD extension board made by E Ink and is connected to the xG22 Wireless Kit (BRD4182A and BRD4001A boards). The xG22 Wireless Kit with the E Ink panel is controlled wirelessly with another wireless kit.

This application note includes:

- Simplicity Studio IDE projects
- Source files (in the project)
- Example C-code (in the project)
- Software examples (in GitHub)

KEY POINTS

- Electronic Paper Displays (EPDs) are reflective and bistable, which means they rely solely on ambient light and can retain an image even when power is not connected.
- EPDs are ideal for applications such as e-readers, industrial signage, and electronic shelf labels.
- EPDs draw no current when displaying a static image, so they maintain a long battery life.

1. Electronic Paper Displays

Electronic Paper Displays (EPDs) are reflective and bistable displays. Reflective in this case means they rely solely on ambient light and do not use a back light. Bistable is the property of retaining an image even when no power is connected.

EPDs are commonly used in e-readers, industrial signage, and electronic shelf labels. Their properties are ideal for applications which do not update the image frequently. Because the display draws no current when showing a static image, they allow a very long battery lifetime.

1.1 EPD Cell - Three Pigment Ink System

The pixels in an EPD are made up of millions of tiny microcapsules, each about the diameter of a human hair. E Ink's Spectra product line utilizes a 3-pigment ink system in a microcup structure. This ink was engineered specifically for Electronic Shelf Labels (ESL) and is offered in black, white, and red. This ink system works similarly to the dual pigment system, in that a charge is applied to the pigments and to a top and bottom electrode to facilitate movement. However, instead of using microcapsules, this system utilizes Microcups®, which are filled with liquid and sealed. For more info about the technology, visit the E Ink website.

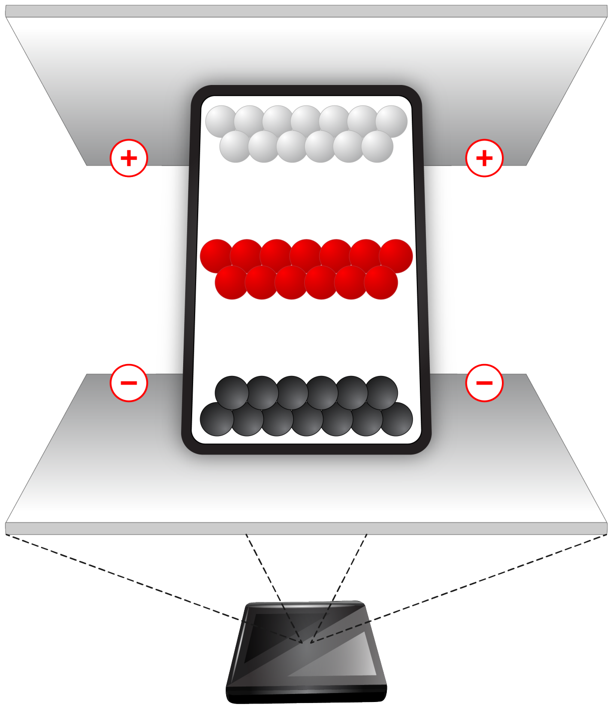


Figure 1.1. EPD Cells

1.2 Benefits of EFR32xG22

Because EPDs have the ability to draw no current, the current consumption of the MCU and the rest of the application becomes very low in the long term. The EFR32 flexible energy modes allow the MCU to always draw as little current as possible. In many EPD applications, Energy Mode 4 can be used in which the MCU consumption can be as low as 170 nA.

Memory is important to save frame buffers and images. The EFR32 has large memory options, both for Flash and SRAM. At the time of this writing, the largest options include 512 kB of Flash and 32 kB of SRAM.

1.3 Considerations

Even though the EPDs draw no current while showing a static image, they require a significant amount of current while updating the display. The display update itself also takes a long time (typically 12 seconds for black and white, 18 seconds for red dots). EPDs are therefore not suited to applications that require a high update frequency.

2. EPD Extension Board

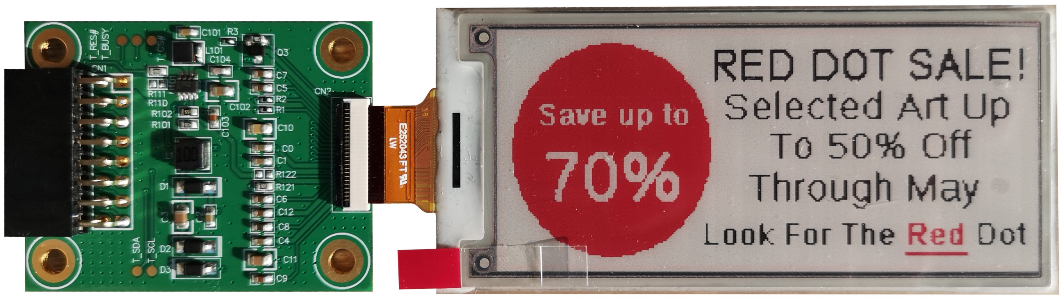


Figure 2.1. E Ink Display Extension Board with Connected 2.9" Display

This application note makes use of the E Ink EPD extension board to illustrate how to drive an e-paper display with an EFR32 micro-controller. The EPD extension board is available for sale with the HULK Driving Board and the full schematics and documentation reference code is attached to this document. Furthermore, these documents can be requested for free via the E Ink customer support. Documentation and ordering information can also be found on their webpage.

The extension board includes the necessary E Ink EPD driver components and a 16-pin header to connect to the MCU. It is based on the reference circuit, so it is an excellent base for new designs.

The flat panel connector is compatible with most of the EPD panels. Additionally, the EPD panel-related flat connector is very sensitive to physical forces, like bending, twisting or stretching, so a temporary tool should be used during the development phase to prevent permanent damage. The signals required to drive the panel are routed to the 16-pin header. The kit comes without standard jump wires, so you'll need to acquire these cables for initial testing, or you can create a probe panel where the signal and power lines are connected between the WSTK and E Ink display driver (see the figure below). Table 2.1 shows how to connect the E Ink display driver board to the EFR32XG22-based Wireless Starter Kit.

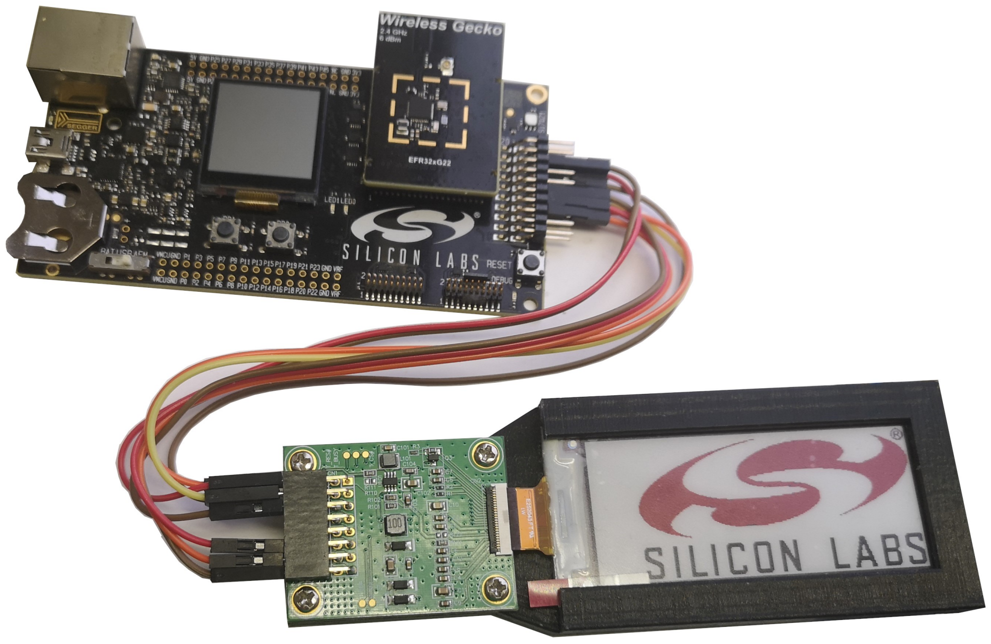
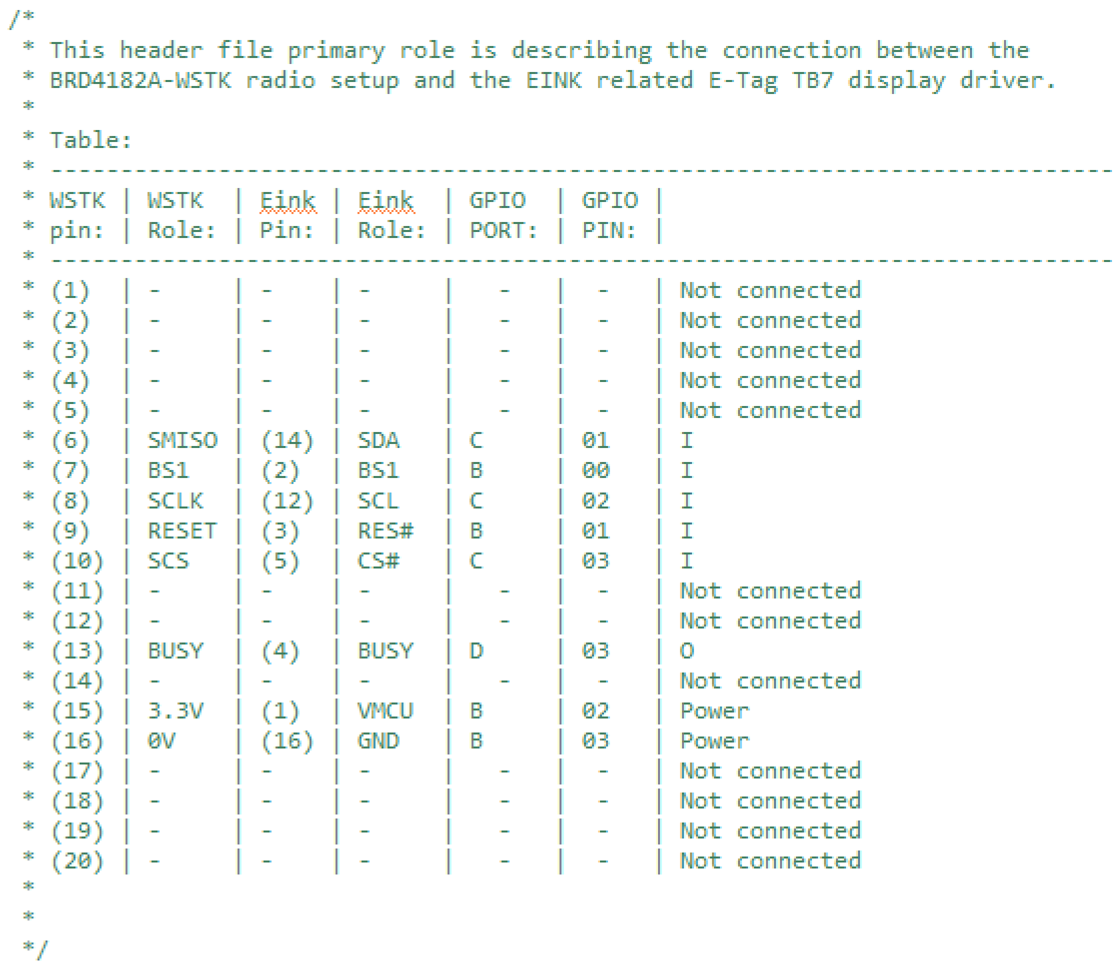


Figure 2.2. E Ink Driver and WSTK Connected

Table 2.1. Connection Table



```
/*
 * This header file primary role is describing the connection between the
 * BRD4182A-WSTK radio setup and the EINK related E-Tag TB7 display driver.
 *
 * Table:
 * ---------------------------------------------------------------------------
 * WSTK | WSTK   | Eink | Eink   | GPIO  | GPIO |
 * pin: | Role:  | Pin: | Role:  | PORT: | PIN: |
 * ---------------------------------------------------------------------------
 * (1)  | -      | -    | -      |   -   |  -   | Not connected
 * (2)  | -      | -    | -      |   -   |  -   | Not connected
 * (3)  | -      | -    | -      |   -   |  -   | Not connected
 * (4)  | -      | -    | -      |   -   |  -   | Not connected
 * (5)  | -      | -    | -      |   -   |  -   | Not connected
 * (6)  | SMISO  | (14) | SDA    | C     | 01   | I
 * (7)  | BS1    | (2)  | BS1    | B     | 00   | I
 * (8)  | SCLK   | (12) | SCL    | C     | 02   | I
 * (9)  | RESET  | (3)  | RES#   | B     | 01   | I
 * (10) | SCS    | (5)  | CS#    | C     | 03   | I
 * (11) | -      | -    | -      |   -   |  -   | Not connected
 * (12) | -      | -    | -      |   -   |  -   | Not connected
 * (13) | BUSY   | (4)  | BUSY   | D     | 03   | O
 * (14) | -      | -    | -      |   -   |  -   | Not connected
 * (15) | 3.3V   | (1)  | VMCU   | B     | 02   | Power
 * (16) | 0V     | (16) | GND    | B     | 03   | Power
 * (17) | -      | -    | -      |   -   |  -   | Not connected
 * (18) | -      | -    | -      |   -   |  -   | Not connected
 * (19) | -      | -    | -      |   -   |  -   | Not connected
 * (20) | -      | -    | -      |   -   |  -   | Not connected
 *
 *
 */
```

Note: The WSTK has power lines, such as GND and VMCU, that have dedicated pins on an extension header, namely WSTK pin 1 and WSTK pin 2. These pins can also power up the E Ink driver board. Based on the EPD, the driver does not have a "power switch" to shut itself down; therefore, if the previous power lines are used, then there will be no way to shut down the E Ink driver panel. It will constantly consume a significant amount of current, which is why GPIO pins are used to power up the EPD.

For an easier interpretation, the following figures also describe the wires among the PCB boards. The pin numbers on the WSTK are highlighted.

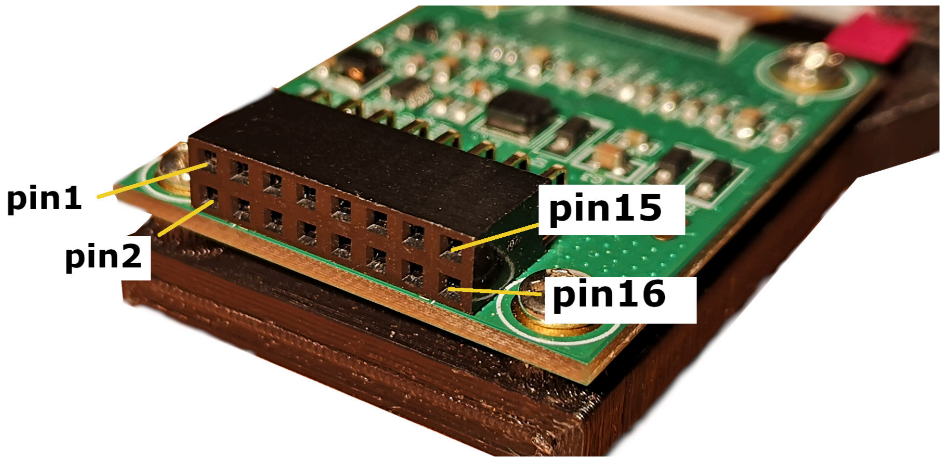


Figure 2.3. E Ink Driver Pinout Marking

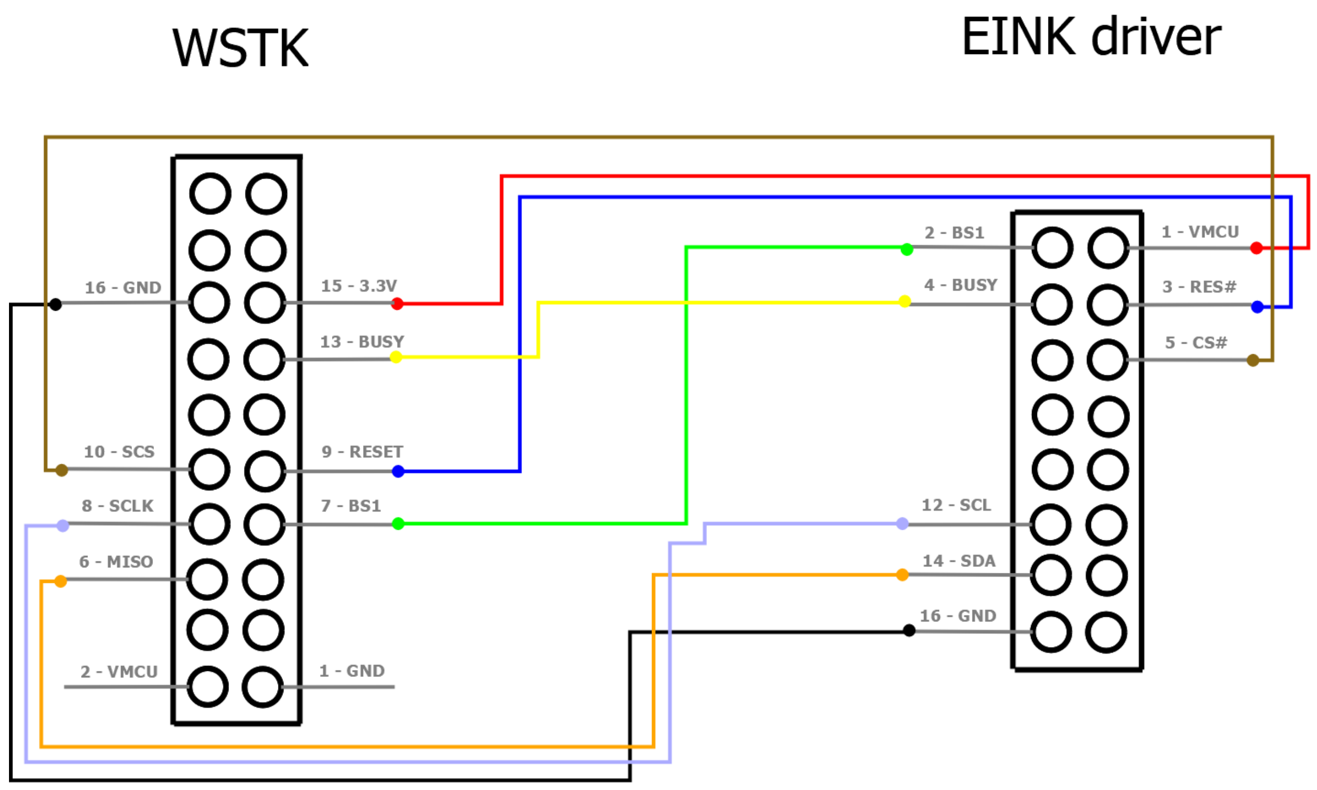


Figure 2.4. WSTK and E Ink Driver Wiring

3. Driving an EPD with EFR32

This chapter will explain what the MCU needs to do to update the EPD panel. All the information in this chapter is based on the documentation released by E Ink. See this documentation for details about each step, including SPI commands and timing.

3.1 Chip-on-Glass Driver

The EPD panels from E Ink come with an integrated Chip-on-Glass (COG) driver that controls the behavior of the panel. The COG has a 3-line serial peripheral interface (SPI) (or 4-line, depending on the configuration) that is used to accept commands from the EFR32.

3.2 Charge Pump

When writing a new image to the display, the panel itself requires a large voltage across it to drive the pixels. To achieve the display without an external voltage supply, use a charge-pump circuit. The E Ink display has an on-chip oscillator, on-chip booster, and regulator control for generating VCOM, Gate, and source driving voltage for its proper operation.

3.3 Image Update Sequence

To draw an image on the EPD panel, the MCU must:

1. Power up and initialize the COG.
2. Write the image data to the DTM1 and DTM2 buffers according to the desired color for each pixel.
3. Power off the COG driver.

During initialization, the MCU communicates with the COG by sending SPI commands and must adhere to specific timings laid out by the COG documentation. The MCU should keep only one frame in its buffer memory: the frame of the new image.

The following image update process happens automatically. It is driven by the EPD display-related COG.

The E Ink display related COG has two buffers for the images: one for the new image to be displayed, and the old image currently visible on the panel. During image update, the panel is updated in four stages:

1. The current image is inverted.
2. The entire panel is drawn white.
3. The inverse of the new image is drawn.
4. The new image is drawn.

During each stage, the same frame is written multiple times to the display. The number of times a frame should be rewritten is dependent on both the type of display and the current temperature. A colder environment makes the particles within the pixel cells move more slowly and therefore requires the panel to be rewritten more. If the application should work in varying temperature conditions, the temperature should be measured before writing to the display; however, E Ink display has a built-in temperature sensor, so this happens automatically. The EFR32 also has an internal temperature sensor that can be used.

After the frame has been written, the COG should be powered down. This process also has a specific sequence of commands and timings the MCU must obey. During this sequence, the pixel registers are first cleared and then the charge-pump capacitors are discharged. After the power-down sequence is finished, all power to the panel can be removed and the image will be kept on the panel.

3.4 Frame Buffer

The COG expects each frame in a special format. A pixel is labeled (x,y) where (00) is the lower right of the display. The update of the EPD display happens from column to column, starting from the bottom of each column, so the developer should handle this display frame as a long 1-dimensional array. Each pixel is described by 1 bit, and the color of this pixel is based on pixel actual value and the previous value. This process is visualized in 1.1 EPD Cell - Three Pigment Ink System.

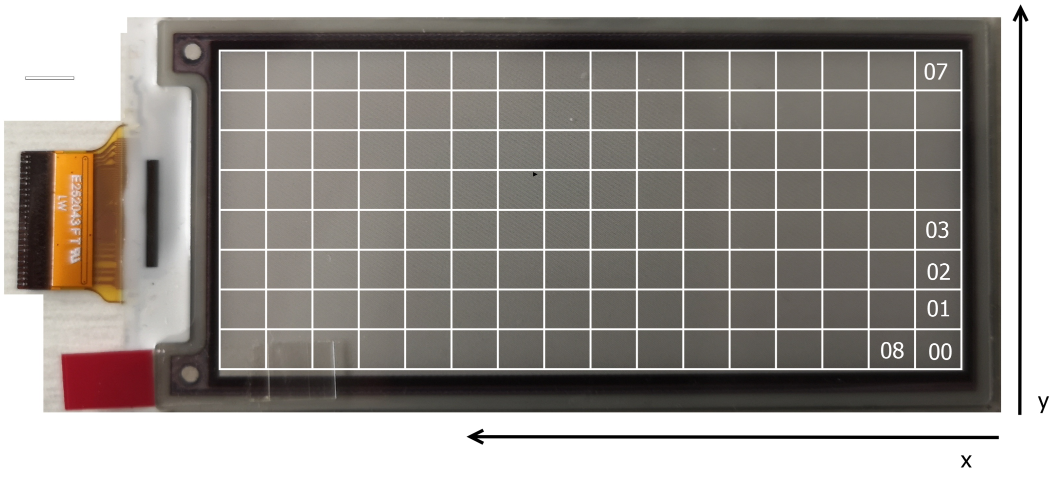


Figure 3.1. EPD Frame

The update mechanism is handled automatically by the E Ink display-related COG. On the registers and the embedded software level, the developer gets two buffers: DTM1 as the previous value and DTM2 as the next value. Based on this simple process, the color of the pixels (bits) can be defined by the following mapping table.

Table 3.1. E Ink Image Processing Tool

DDX[1:0]	DTM2	DTM1	Display
11	0	0	Red
	0	1	
	1	0	Black
	1	1	White

3.5 Rendering with E Ink Image Processing Tool

The E Ink image processing tool is a python-based tool developed by Silicon Labs. The tool is completely open source, so users can change it according to their needs. The tool can be found under the **pc_tool** folder and, with a simple double-click to the *start_image-Consol.bat*, the application will start. No further adjustments are required to generate the sufficient frame format for the E Ink 2.9" tri-color display.

3.5.1 Converting Images

The E Ink image processing tool can open common image formats, such as GIF, BMP, and PNG, and save the output as a normal 1-dimensiona; array that can be included and compiled into the MCU application.

When using the E Ink image processing tool to prepare an image for use on an EPD, first open the desired image [File -> Open File]. Then, two picture frames will appear on the screen: the bigger one is the original picture with the original screen aspect ratio. If the original picture is bigger than 640x480pixels, then the original picture will be reduced to 640x480pixels while holding the original aspect ratio. The right, smaller picture is a 294x128 resolution picture, which shows the expected picture on the E Ink display. It can be adjusted further using the rotating option [Process -> Rotate right/ Rotate Left].

The output byte stream and bit-stream can be found under the E Ink image processing tool-related *python\tmp\*folder. The byte stream should be used in the MCU application as the content of a 1-dimensional array. These files are generated automatically after any image processing option, such as open or rotation in any direction.

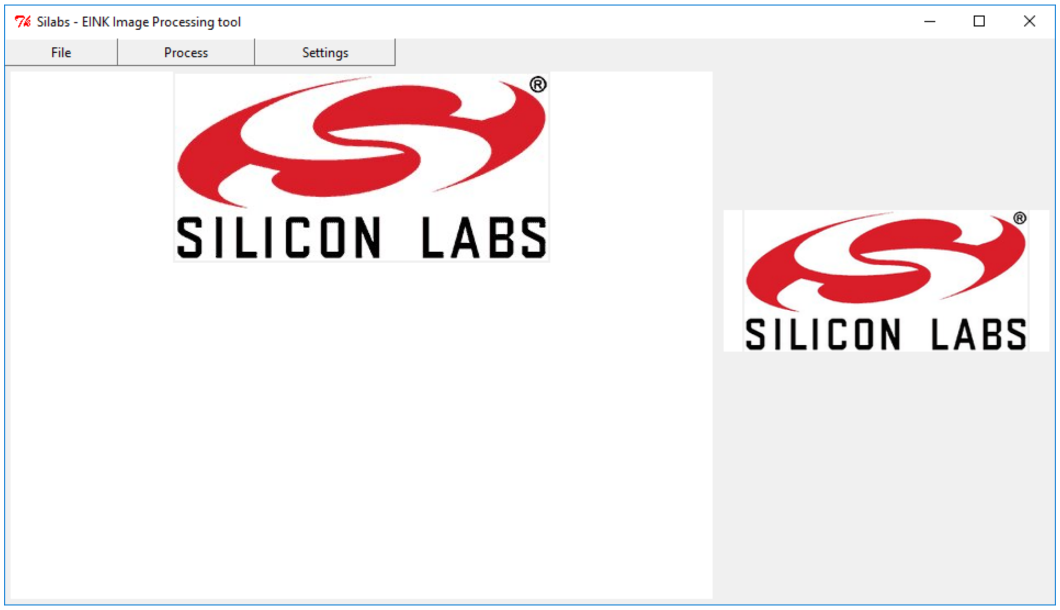


Figure 3.2. E Ink Image Processing Tool

Note that, on the bit level, the image shape is still visible.

bitstream_DTM2.txt

Figure 3.3. Bit-stream

4. Power Consumption

The EPD only consumes power during a display update. A display update, however, takes a significant amount of time, typically between 12-18 seconds at room temperature. In addition to this time, the MCU must complete the power-up and power-down sequences and transmit frames to the panel.

To optimize power consumption during a display update, ensure that the MCU is in its optimal energy mode at all times. When performing CPU intensive tasks, such as rendering a new image, the MCU should run at a high frequency. During the update stages, when the frame is transmitted over SPI to the EPD panel, the clock frequency can be slower because the stage should run for a predetermined amount of time. During the periods when the MCU is waiting for the COG to be ready, the MCU can be in a sleep mode.

The main parameter that can be optimized to save power during a display update is the BUSY period. The E Ink display uses the BUSY pin to signal that no EFR32 interaction is allowed. During these busy periods, the MCU can go to sleep, decreasing the overall current consumption.

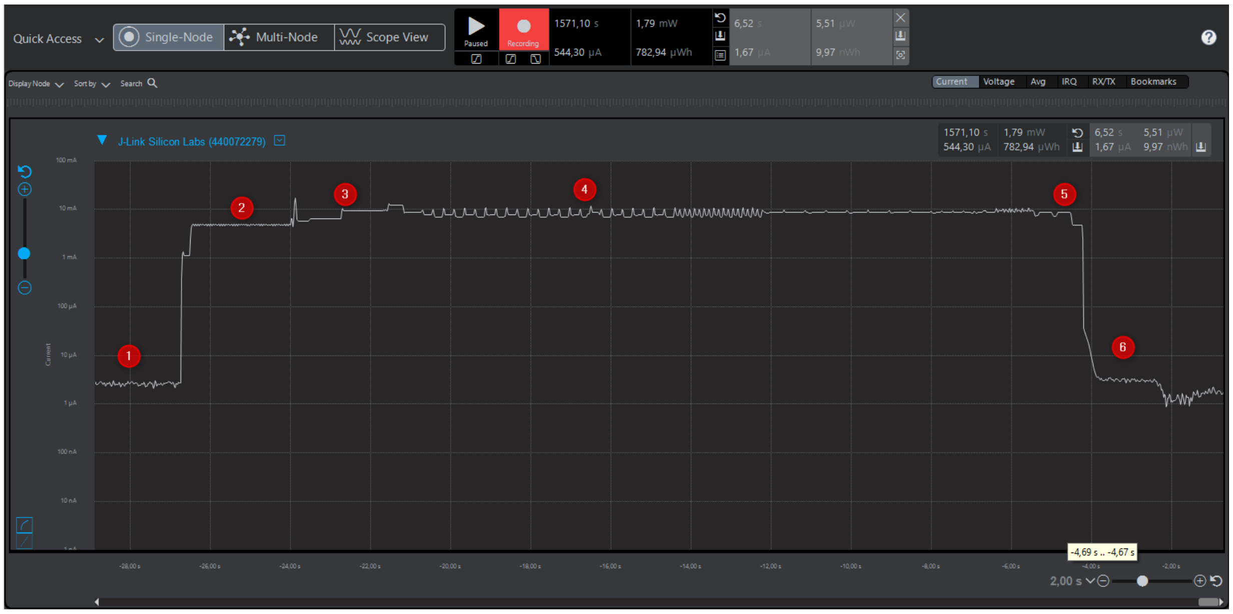


Figure 4.1. EPD+ MCU Power Consumption

The figure above shows a breakdown of the MCU + EPD power consumption during a display update. This figure is taken with the Energy Profiler while the MCU is running the E Ink driver example with wireless update example shown in section 5.2 E Ink Driver Example with Wireless Frame Update.

Referring to the numbers in the figure above:

1. EFR32 in EM4 power state sleeps and waits for a wake-up event.
2. The EFR32 receives the new frame and stores it in the memory.
3. EFR32 powers up the E Ink display driver, powers up the COG and charge-pump, and transfers the new frame to the E Ink.
4. E Ink COG updates the display with a new frame.
5. COG shutdown sequence begins and the EFR32 releases the power, SPI, etc., lines between the E Ink driver and the WSTK.
6. Power is cut to the panel. The MCU is in EM4.

Note: The measured EM4 current is much higher than the expected n*100 nA. It is based on the architecture of the EPD driver board. The absence of a "power switch" on the EPD board causes leakage current on the GPIO pins. However, the average leakage current on most of the GPIO pins is ~2 nA (at room temperature). On the other side, there are special function pins where leakage current can exceed the n*100 nA. Because there are not many EFR32 pins routed to the WSTK's expansion header, few of these high leakage current pins are used in this application.

4.1 Choosing a Sleep Mode

Because EPDs do not require power or interaction while displaying a static image, EM4 can be used as a sleep mode. Whether it makes sense to use EM4 depends on the application, specifically if other tasks need to be performed during sleep, and the update frequency.

In EM4, the EFR32 can draw as little as 0.17 µA of current and wakes up on a GPIO interrupt (NB: only on selected pins, see reference manual). However, when waking up from EM4, the MCU has to go through a full reset and run the startup/initialization code gain, which will add to the average current consumption. Therefore, it is dependent on the application that has long sleep periods and gains the most by using EM4.

The theoretical advantage of using EM4 is shown in the figure below. The graph is calculated assuming a simple application that stays in a sleep mode and updates the display based on a periodic GPIO interrupt. The horizontal axis is the update period, and the vertical axis shows the average current consumption. At short update periods (high update frequency), the current consumption is dominated by the MCU's active mode current consumption and the EPD's current draw when updating the image. In this case, the sleep mode current can be ignored. However, as the update period gets longer, the average current consumption becomes dominated by the sleep mode current consumption.

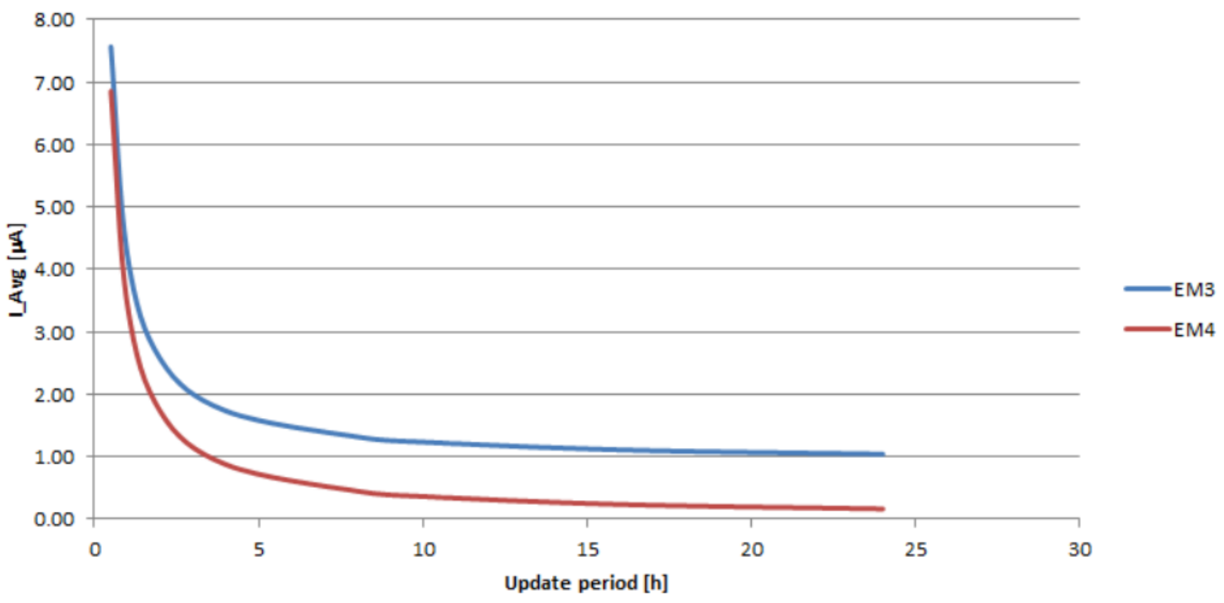


Figure 4.2. Average Current Consumption (MCU + EPD)

Note: This graphic is a theoretical approximation to visualize the correlation among energy modes, current consumption, and the frequency of the ELD frame's update

5. Software Examples

This application note comes with two software examples.

5.1 E Ink Driver Example

The E Ink EPD display demo application will be driven directly with the EFR32xG22 MCU and with the related development boards.

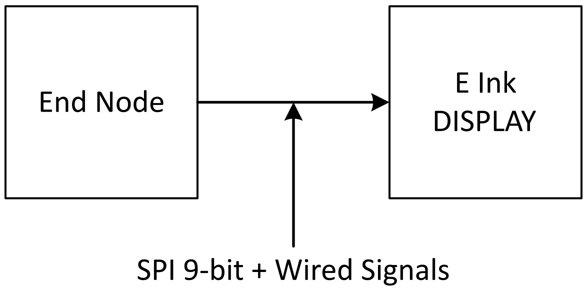


Figure 5.1. E Ink Display and EFR Driver

In this example, the EFR32XG22 development board drives the E Ink panel directly without any wireless connection. The new image frame will be compiled into the source code.

The following boards and parts are required for this demo:

- Silicon Labs BRD4001A WSTK main board
- Silicon Labs BRD4182A radio board
- E Ink E-tag TB7 display driver board
- EINK EA2200-BJA display
- Jumper cables

Mount the BRD4182A radio board to the BRD4001A WSTK mainboard and, based on Figure 2.4 WSTK and E Ink Driver Wiring on page 5, connect the E Ink E-tag TB7 display driver board to the WSTK main board's extended header. The E Ink EA2200-BJA display also should be connected to the display driver.

The E Ink image processing tool can be used to generate the byte streams for the DTM1 and DTM2 registers. To generate the streams, start the application, then select "File" -> "Open File" to choose the desired picture. With "Process" -> rotate options, you can change the picture indentation to have a better look. Using no "Generate" button, the DTM1 and DTM2-related values can be found in the pc_tool/tmp/ bytestream_DTM2.txt and bytestream_DTM1.txt files.

To copy-paste these previously generated byte streams to the EFR32's source code, open the Simplicity Studio V5 and import the *eink_brd4182a_simple_driver.sls* project from the *SimplicityStudio\endnode\example_1* folder. Next, navigate to the project-related root folder, open the *app.c* file, then copy-paste the previously generated DTM1 and DTM2 values to the *app.c* file related to the *dtm1_buffer* and *dtm2_buffer* named arrays (see line 25). Do overwrite the previous values.

```
...

// !!! GENERATED FRAMES HERE !!! ///
static uint8_t dtm1_buffer[BUFFER_SIZE] = ...  // generated stream here
static uint8_t dtm2_buffer[BUFFER_SIZE] = ...  // generated stream here

...
```

Next, compile the previously imported and modified project and load the newly generated *.hex file to the EFR32. When the example starts, the EPD display first shows the image, then becomes red, then black, and finally, the white image is shown.

5.1.1 SPI 9-bit Description

The EFR32MG22 uses its USART0 peripheral to perform the communication between the E Ink display driver and itself, with the following attributes:

- baudrate: 115200 bps
- format: 9 bits

5.2 E Ink Driver Example with Wireless Frame Update

This example's end node side works as the previous example with extended wireless functionality. In the following steps, the new image will not be compiled into the EFR32xG22 firmware. Instead, the new frame will be transferred via wireless proprietary protocol.

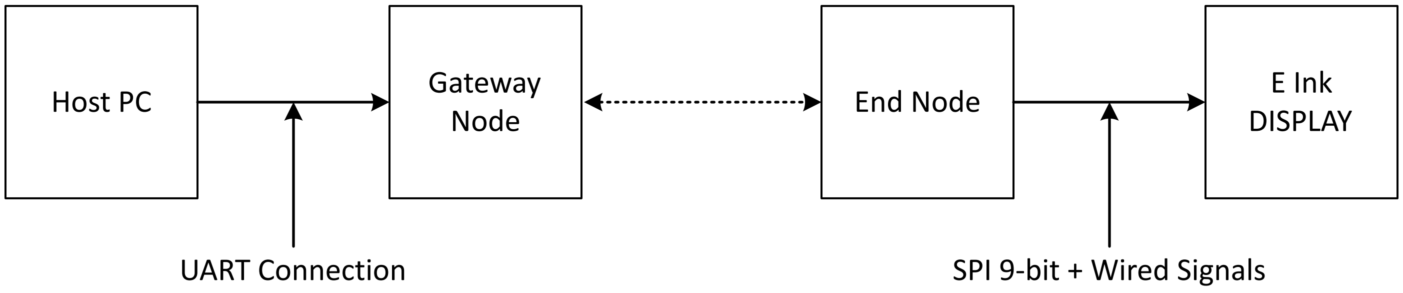


Figure 5.2. E Ink Display and EFR Driver via Wireless Update

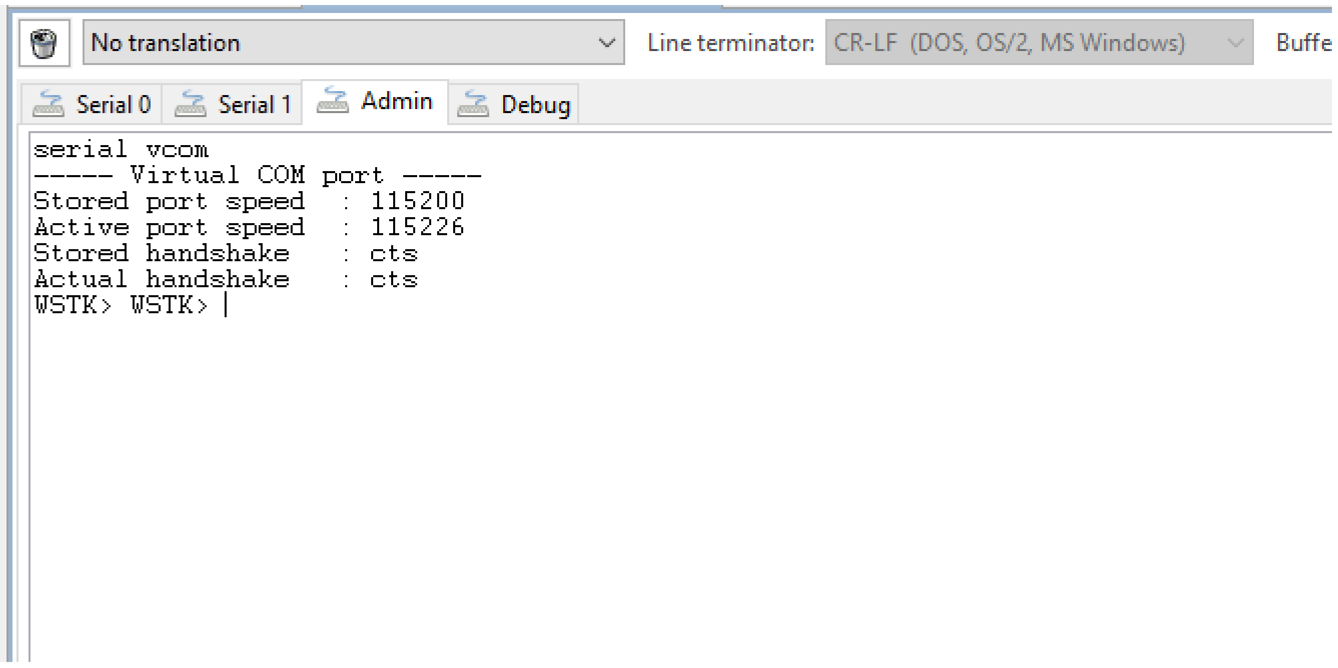
The following boards and parts are required for this demo:

- For the end node side, the same as in the previous example (mainboard, radio board, and wirings)
- For the gateway side, Silicon Labs BRD4001A WSTK mainboard and Silicon Labs BRD4163A radio board

Hardware and firmware preparation to execute this example:

- On the end node side, the related hardware connection is the same as in Figure 2.4 WSTK and E Ink Driver Wiring on page 5.
- On the gateway side, the BRD4163A radio board should be mounted to the BRD4001A mainboard, then connect the mainboard to the Host PC.
- Import both *eink_brd4182a_end_node* and *eink_brd4163a_gateway* projects into Simplicity Studio V5.
- Compile the *eink_brd4182a_end_node* and load the project-related *.hexfileto the EFR32xG22 board (end node).
- Finally, compile another *eink_brd4163a_gateway* project and load the generated *.hex file to the BRD4163A board (gateway).

Note: If the gateway-related VCOM cts/rts settings are incorrect, the settings should be modified. The proper settings are highlighted in the figure below. The knowledge base article, KBA_BT_1208: Using Virtual COM Port (VCOM), describes the process of modifying and checking the actual values of the CTS/RTS pins.



```
serial vcom
----- Virtual COM port -----
Stored port speed   : 115200
Active port speed   : 115226
Stored handshake    : cts
Actual handshake    : cts
WSTK> WSTK> |
```

Figure 5.3. Gateway Node VCOM Settings

The E Ink image processing tool can be used to send the new image to the end node. The preparation of the image is the same as in the previous example. After the preparation is finished, the E Ink image processing tool shall connect to the gateway, so click on "Settings - Port", then select the gateway-related port number. The previous port number step does not initiate the send process, so send should be initiated manually. Click "Process" -> "Send to device" menu.

The blinking of the LEDs both on the end node and gateway signals ongoing wireless traffic. At the end, the end node updates the new image frame to the E Ink display.

5.2.1 Host PC Description

The host PC used in testing was a Windows 10-based computer; however, any computer that is capable of communicating with RS232 or with an extended USB-UART converter and can run the Python script can be used. In this example, the host PC was used to perform the image manipulation, create the output byte streams for DTM1 and DTM2 arrays, and send these streams to the Gateway Node via UART Connection. For executing the E Ink image processing tool, Python version 3.9 should be used. The script requires additional libraries that are listed in the pc_tool/readme.md file.

5.2.2 UART Connection Description

The host PC and gateway node use UART for their interactions, namely with the WSTK-related VCOM port and the host PC's USB socket. The VCOM port uses the WSTK programmer IC, so in this setup, no additional USB-UART converter is required.

The UART communication has the following parameters:

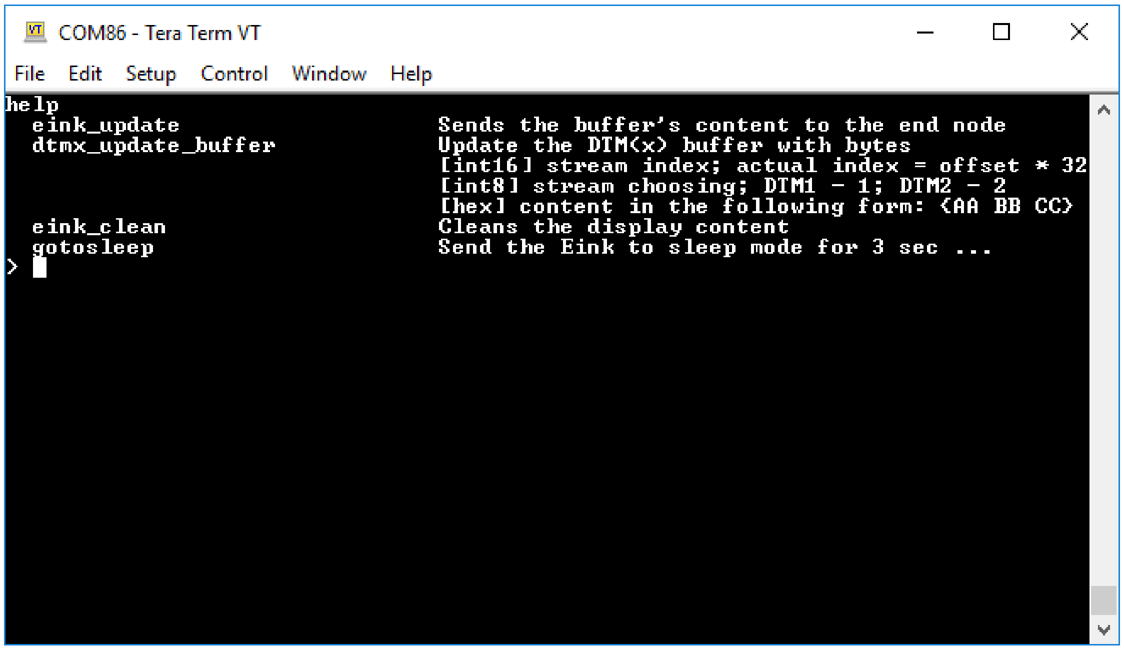
- baudrate:115200
- format: 8N1
- cts/rtsused (only from host PC to gateway node)

The commands between the two participants are Gecko CLI based commands. The E Ink image processing tool issues these commands specially created for this application, such as sending the DTM stream-related content, updating the E Ink display or cleaning the E Ink display with the gateway node. An advantage of this approach is that you can perform this task without the E Ink image processing tool. Any third party terminal tool (such as TeraTerm) can send the data to the gateway and start the wireless transmission.

The following CLI commands are implemented:

- `eink_update`: Sends the gateway node-related DTM buffer values to the end node DTM buffers, then updates the E Ink display with the new received image frame. After the command has been executed, the gateway sends a `gotosleep` command to the end node.
- `dtmx_update_buffer`: Updates the gateway node local DTM buffers with argument specified values. It will not start the sending process between the end node and the gateway, it just fills the gateway buffers.

  The `dtmx_update_buffer` command has three arguments:
  - 1st:offset: The 3rd argument values will be inserted to the DTM buffer with this offset. For the calculation and to get the exact DTM index, multiply the issued offset by 32. For example: `dtmx_update_buffer 01 {FF FF FF}` will update the DTM1 buffer 0, 1st and 2nd element with 0xFF (because 0*32 = 0). However, `*dtmx_update_buffer 1 1 {FF FF FF}` will update the DTM1 buffer 32th, 33th, and 34th with 0xFF (because 1*32 = 32).
  - 2nd: DTM1 or DTM2 buffer. "1" means DTM1 and "2" means DTM2 buffers.
  - 3rd: Data field for sending desired data to the DTM buffers. It allows maximum 32 bytes. The format is: `{FFFF FF}`.
- `eink_clean`: Gateway sends a command to the end node, which causes the end node to issue a function call to fill the E Ink display's memory according to the content that just became white. After the command has been executed, the gateway sends a `goto-sleep` command to the end node.
- `gotosleep`: Gateway sends a command to the end node that causes the end node to go to an EM4 state for 10 seconds. After the 10th second, the end node wakes up and waits for a new command. At default, if there is no new command, the gateway issues a new `gotosleep` command.



```
help
  eink_update                   Sends the buffer's content to the end node
  dtmx_update_buffer            Update the DTM(x) buffer with bytes
                                [int16] stream index; actual index = offset * 32
                                [int8] stream choosing; DTM1 - 1; DTM2 - 2
                                [hex] content in the following form: {AA BB CC}
  eink_clean                    Cleans the display content
  gotosleep                     Send the Eink to sleep mode for 3 sec ...
>
```

Figure 5.4. CLI Commands

This CLI command approach also has a negative side effect. The CLI has some timing overhead, so the wired transmission is much slower than wireless. It can be eliminated in the future by using just the bare metal UART peripheral for the wired communication.

5.2.3 RF Connection Description

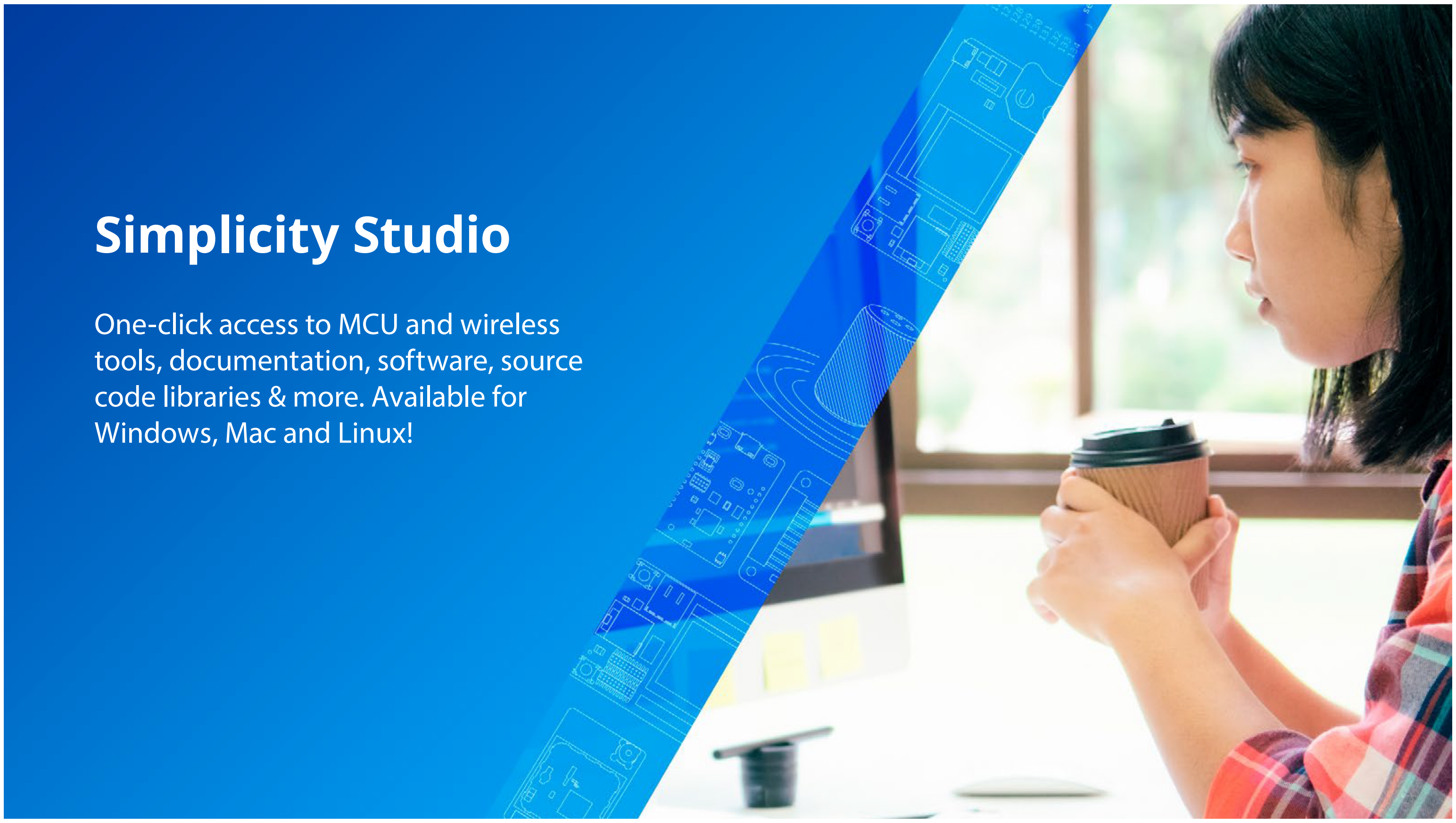
The gateway node and the end node uses proprietary protocol to exchange data between each other.

The main details of the RF communication are:

- 2.45 GHz
- 2GFSK modulation 250 kbps speed
- Variable payload length

For more details, check the attached project-related radio configurator's settings.

The gateway node slices the DTM buffers' content into 127 byte chunks, and these chunks will be sent to the end node one by one. The RF frames are variable length frames, so the payload first byte contains the total length of the payload. With this approach, if the last chunk is not 127 bytes long, then the sender and the receiver node can handle this scenario. The end node performs a CRC check on the received DTM data. If the received data fails on the CRC checking, then the end node sends a NACK message to the gateway and the gateway retransmits the previously failed message.


Simplicity Studio
One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com