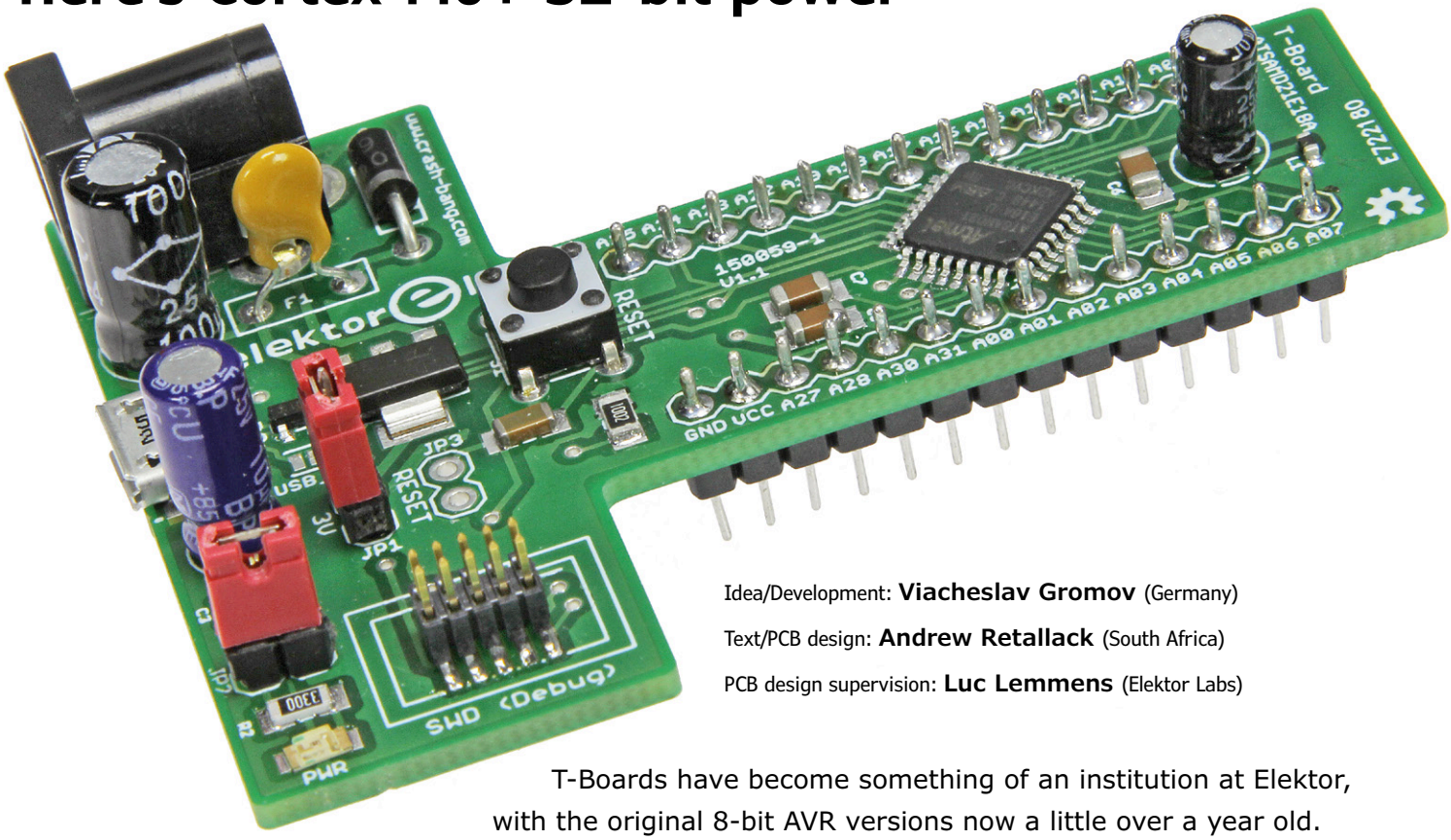
ARM'ed T-Board

Here's Cortex-M0+ 32-bit power



Idea/Development: **Viacheslav Gromov** (Germany)

Text/PCB design: **Andrew Retallack** (South Africa)

PCB design supervision: **Luc Lemmens** (Elektor Labs)

T-Boards have become something of an institution at Elektor, with the original 8-bit AVR versions now a little over a year old. Since then we've also seen wireless and audio T-Boards. Now we introduce the youngest (and most powerful) member of the family — meet the 32-bit ARM T-Board.

While 8-bit microcontrollers remain the bread and butter of the enthusiast and maker communities, there is a growing interest in 16- and 32-bit microcontrollers. Without doubt, the most popular emerging architecture is the ARM Cortex-M — a core that nearly all the major manufacturers include in their microcontroller line-ups. Unquestionably, the popularity is in part driven by declining prices (you can now buy a small Cortex-M0+ for less than $1), combined with a requirement for more processing power, features and I/O pins as interest in IoT continues to grow. Also, with Elektor Magazine running its "*From 8 to 32 Bits*" ARM programming course, the educational aspect is covered too.

While there are a range of ARM Cortex development boards around, we believe that the T-Board is a perfect platform for an ARM Cortex-M0 microcontroller. ARM chips come in small-pitch packages, and need a range of supporting components, making them not that "user-friendly" straight off the shelf — they need to be mounted on some kind of development board to be at all useful to the enthusiast, or for prototyping purposes. Sadly the existing development boards are not designed for use on breadboards, making them sometimes clumsy and hard to use in a prototyping environment. Enter the T-Board: specifically designed to be plugged directly into the breadboard with all the pins brought out and unobscured. If you've ever used another T-Board you'll know just how quick and easy it is to get up and running with a prototype!

ARMing the T-Board

The ARM-equipped T-Board houses the Atmel ATSAMD21E18A microcontroller, an ARM Cortex-M0+. If you aren't familiar with the ATSAMD21E18A, take a quick look over these specifications:

- 8/16/32-bit Timer/Counter
- 3 24-bit Timer/Counters
- 32-bit RTC with clock/calendar functions
- USB 2.0 interface (host and device functionality)
- 4 Serial interfaces (USART, I^2C, SPI, LIN)
- 12-bit DAC with 20 channels (differential and single-ended input; oversampling support in hardware)
- 10-bit DAC
- Inter-IC Sound (I^2S) Interface
- 2 Analog Comparators
- 16 External Interrupts
- 26 GPIO pins
- Peripheral Touch Controller: 10 x 6 lines
- Maximum frequency: 48 MHz
- 1.62 V to 3.63 V supply

That's quite a handful of features compared to *ye olde 8-bit micro*. We felt that

this microcontroller would be the perfect one to add to the T-Board platform: in addition to the specifications we've listed above, the pin count is manageable for a breadboard-based board; it has a very decent 256 KB of flash memory and 32 KB of SRAM; has USB support; and comes in at a reasonable price point.

Now that we've established the "What" and "Why" of the ARM'ed T-Board, let's delve into the details of the design.

The T-Board design

Figure 1 shows the schematic for the ARM-style T-Board. In common with its AVR-based siblings, primary power is supplied through a standard 2.1-mm center-positive jack (K1). A 1N4007 diode (D1) protects the board against reverse polarity, while a PTC resettable fuse (F1) protects against overcurrent with a 0.5 amp trip-rating — more than enough to power the board and a few peripherals (given that each VDD pin can draw a maximum of 92 mA). Capacitors C1, C2 and C3 smooth and filter the supply current as it passes through a 3.3 V low-drop-out voltage regulator (IC2). Unlike the Atmel AVR microcontrollers, the ARM Cortex MCUs can tolerate a maximum supply of 3.8 V. Hence we settled on the common 3.3 V for the supply. Parts R2 and LED1 combine to indicate that the board is powered. The jumpers JP1 and JP2 have been placed to help with reducing the power consumption — more on that further on.

Continuing along the journey through the schematic, you'll note the micro-USB connector at K5. The 5-V supply from the USB connection is of course fed into the voltage regulator to be filtered and dropped to 3.3 V. Electrostatic discharge (ESD) is a threat when connecting peripherals, so an ESD protection diode array (D2) has been added to protect the USB D– and D+ signal lines. We've chosen the NXP PRTR5V0U2X for this purpose; it comes in a tiny SMD package and is designed specifically to protect high-speed data lines. Buckle up: its official name according to NXP is: *Ultra low capacitance double rail-to-rail ESD protection diode*.

The fifth pin on the USB connector, USB_

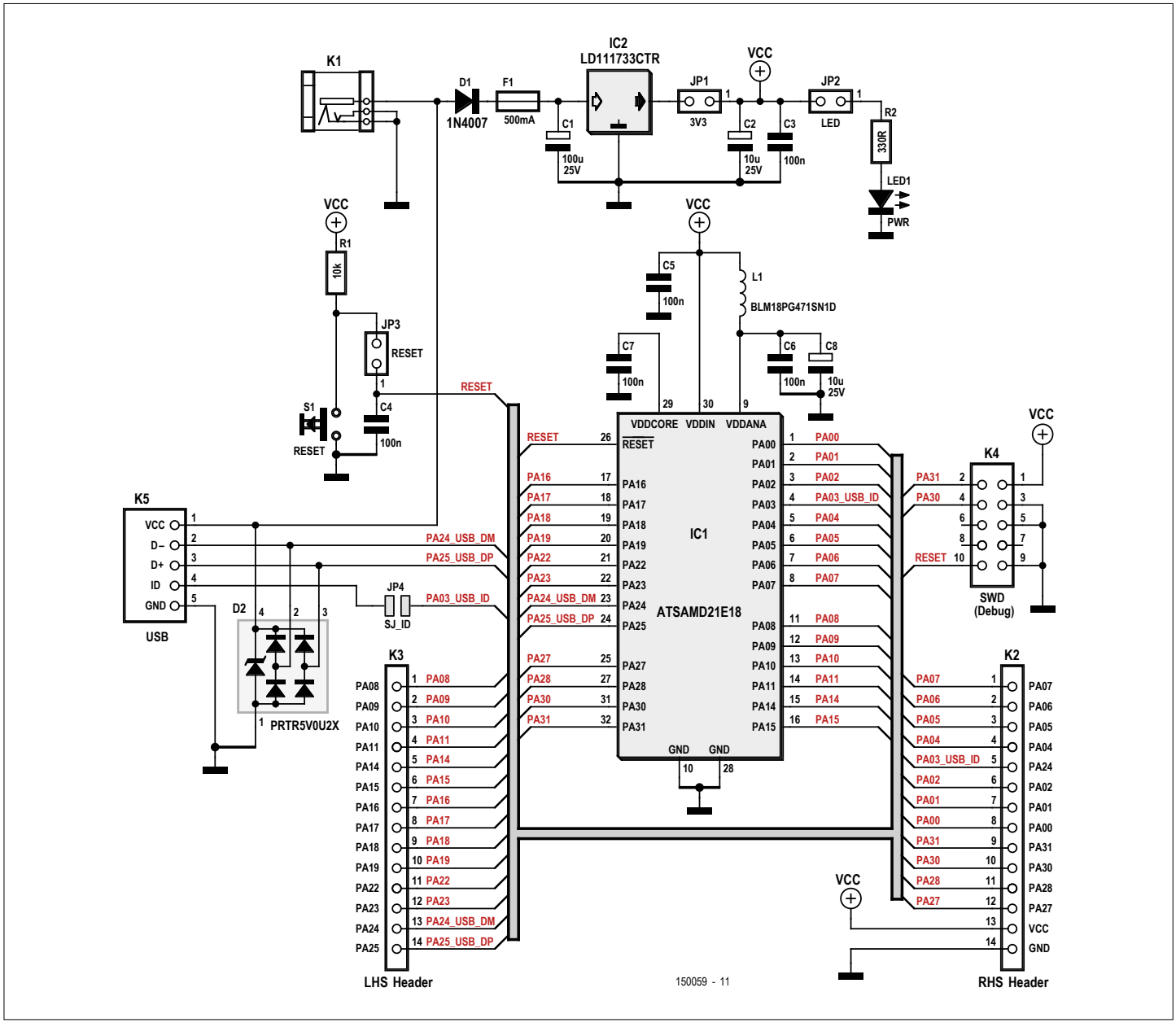
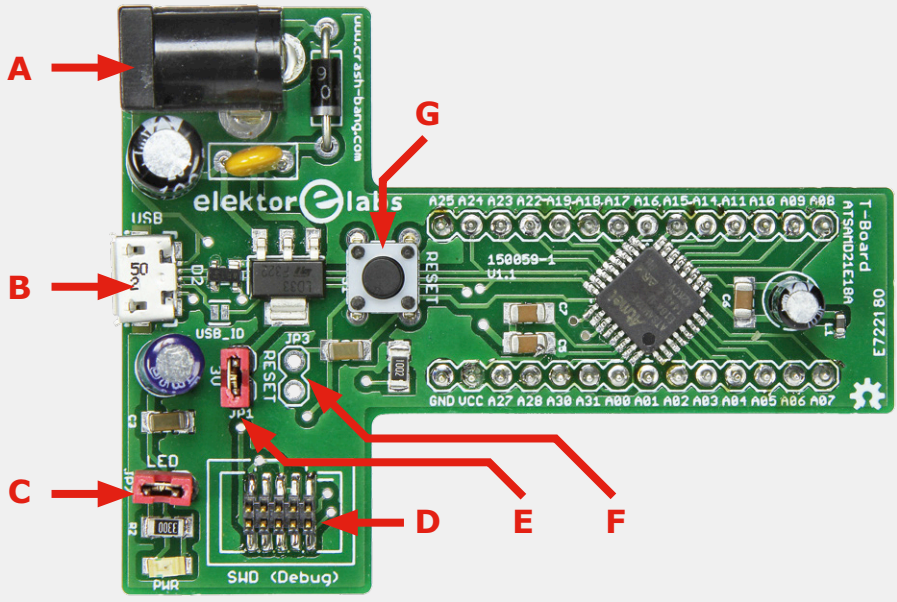


Figure 1. Schematic of the ARM'ed T-Board.

Physical Layout of Board

A. Power Connector: 2.1mm center-positive DC Jack (max. 9 V)
B. Micro USB Connector
C. LED Jumper: Connect/Disconnect the power LED
D. SWD Header: 10-pin programming header (Serial Wire Debug)
E. Current Measurement: Remove jumper to enable in-line current measurement
F. Reset Header: Enables remote reset of board
G. Reset switch



This T-Board is a perfect platform for an ARM Cortex-M0+ microcontroller

ID, is by default not connected to the MCU (in order to free up available pins). A solder jumper however allows you to connect it to pin PA03. The ID pin is used for USB On the Go (OTG); refer to Atmel's application note for more details [1].

An additional feature is a 2-pin header on the Reset line: this enables remote resetting of the microcontroller. Programming and debugging are through a 10-pin 0.05-inch pitch SWD header. SWD (Serial Wire Debug) is a subset of the JTAG interface, and only uses 6 wires. The 10-pin header is there to maintain compatibility with Atmel programmer/debuggers such as the Atmel ICE.
The rest of the board essentially contains the headers that break the microcontroller's I/O pins out, and a collection of supporting components: a healthy dose of decoupling capacitors as specified in the datasheet, as well as a ferrite bead (L1) to prevent VDD noise interfering with the analog supply VDDANA.
You'll note that there is no quartz crystal on the board. This was to give the user flexibility to choose an external crystal that suited their design best: either a crystal oscillator in the range 0.4 MHz to 32 MHz on PA14/PA15, or a 32.768-kHz watch crystal on PA00/PA01 for RTC functionality.

Reducing power consumption

You may recall from the *T-Boards Lowest Power Exercising* article back in December 2014 that the author wrote an article on ways to measure and then reduce

Component List

Resistors
SMD 1206, 1%, 0.125W
R1 = 10kΩ
R2 = 330Ω

Capacitors
C1 = 100µF 25V 20%, radial
C2,C8 = 10µF 25V 20%, radial
C3,C4,C5,C6,C7 = 100nF 50V, X7R, 1206

Inductor
L1 = BLM18PG471SN1D ferrite bead, 0.2Ω/1A, 470Ω @100MHz

Semiconductors
D1 = 1N4007 (1000V, 1A)
LED1 = LED, red, SMD 1206
IC1 = ATSAMD21E18A-AU
IC2 = LD1117S33CTR (3.3V LDO voltage regulator)
IC3 = PRTR5V0U2X (ESD protection diode)

Miscellaneous
S1 = switch, tactile, 24V, 50mA, 6x6 mm
JP1,JP2,JP3 = jumper, 1x2, vertical, 0.1'' pitch
F1 = 500mA PTC resettable fuse
K1 = DC barrel jack 2.1mm pin
K2,K3 = 14-way pinheader, SIL, 0.1'' pitch
K4 = 10-way (2x5) pinheader, 0.05'' pitch
K5 = micro USB receptacle, 2.0 type B, SMD
jumper socket 0.1'' pitch
PCB, Elektor Store # 150059-1
Alternatively: ready-assembled board, Elektor Store # 150059-91

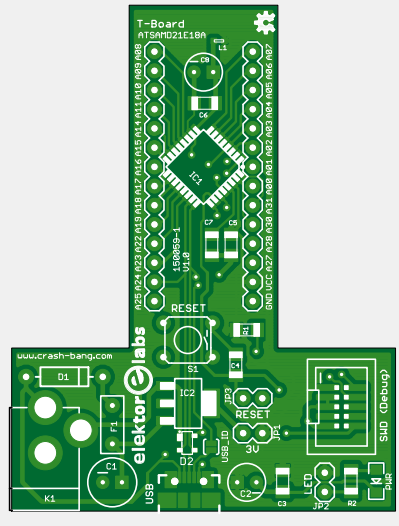


Figure 2. Component layout of the ARM'ed T-Board. Don't worry, the board is available ready manufactured from the Elektor Store.

the power consumption of the T-Board-28 (housing an ATmega328 microcontroller). The ability to achieve this kind of flexibility and control was one of the key motivations behind the design of the T-Board — and this has not been lost in the design of the T-Board ARM.

Jumper JP1 enables you to connect current-sensing circuitry to measure the consumption of the board, and therefore to assess the impact of any code changes and optimizations introduced to reduce the current draw. The original AVR T-Boards allowed you to do this; however we have improved on this in the T-Board ARM by including JP2. That jumper allows you to disconnect the power LED, so that the draw from the LED doesn't skew your measurements.

The physical board design

If you've already used a T-Board you'll see the same heritage running through this board's blood-lines. You will recall that the horizontal cross-bar of the "T" is designed to keep the bulk of the components out of the way of your breadboarding, while the vertical stem breaks the microcontroller's pins out in a way that makes it straightforward to plug into the breadboard and access them.

The T-board is of course open source in the spirit of sharing, and the design files are downloadable for you if you have the equipment and wherewithal to make your own board [3] using the parts list and the board layout in **Figure 2**. If you aren't feeling quite brave enough to tackle the 0.8-mm pitch of the ARM microcontroller TQFP package, or the invisible pins of the USB connector, then the board is available through the Elektor store, fully assembled and tested, and ready to be plugged into your breadboard.

ARMed-'n-ready

I'm sure you've heard enough about the design and want to see the board in action! Let's keep this project straightforward to illustrate the use of the board. To explore more complex projects, turn back to the series *From 8 to 32 bits: ARM Microcontrollers for Beginners* that made its debut in the January & February 2015 edition of Elektor Magazine.

Working with ARM microcontrollers is definitely more complex than with AVR microcontrollers. If you're new to them they require perseverance and some time spent reading up on their systems and architecture. Our purpose here is to introduce you to the T-Board, not to dive into the intricacies of ARM microcontrollers — we don't have nearly enough space in this article, and the "*From 8 to 32 bits*" series covers it comprehensively. Moreover, Viacheslav Gromov who kicked off this project will be covering software specifically on this ARM T-Board in a future edition. With these *caveats*, let's get cracking.

Step 1: Layout the Breadboard

The first step is to place the T-Board onto a breadboard. Position it so that the section containing all the power components projects over the edge of the breadboard and the header pins are located on either side of the breadboard's central channel. Then:

1. Connect a jumper between the GND pin and the negative power rail of the board.
2. Connect the anode of an LED to pin A07, placing the cathode in an empty row on the breadboard.
3. Place a resistor so that one leg is in the same breadboard row as the cathode, and the other leg inserted into the negative power rail.

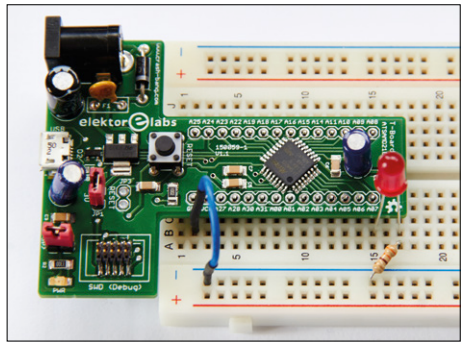


Figure 3. This ARMed-T-Board-on-a-breadboard is all ready to go.

Figure 3 shows the board plugged onto a breadboard and all connected up – I'm sure you'll agree that it's impressive to see only one jumper on the entire board!

Step 2: Create the Project

Next up is the creation of the project. We'll work in Atmel Studio 6.2 for this

Alternative IDEs

When you start working with ARM microcontrollers, you're playing in the "big league" when it comes to Integrated Development Environments. There are a range of developers who provide what seem to be fairly solid tools — but it's out of the scope of this article unfortunately to draw full comparisons between what are complex pieces of software. To give you a taste of what's out there, here are three of the more popular ones.

Atmel Studio 6.2

This in my mind is the first choice for those not professionally developing on ARM microcontrollers. It is free, is not code-limited, and is written and supported by Atmel themselves, hence is guaranteed to support not only the microcontrollers but also the Atmel programmers and debuggers. *www.atmel.com/atmelstudio*

IAR Embedded Workbench for ARM

IAR have a wide range of professional-level IDE's – and of course at professional prices! They support Atmel's AVR and ARM microcontrollers, as well as a range of ARM and non-ARM MCUs from other manufacturers. If you work with a range of microcontrollers from different manufacturers, then this may be a good bet if you can afford it. There are code-limited versions, but so limited that they are more useful for assessing the tool than for any practical projects. *www.iar.com/iar-embedded-workbench/arm*

Keil

Keil MDK-ARM is an IDE that is owned by ARM itself, and is a respected tool for development of ARM projects – primarily aimed at professionals. However, Keil do offer a "lite" edition of their MDK-ARM tool that is code-limited to 32 KB – probably enough for most enthusiasts and hobbyists. If you have the time to spend, and want to explore ARM micros from other manufacturers, then it's well worth looking at. *www.keil.com*

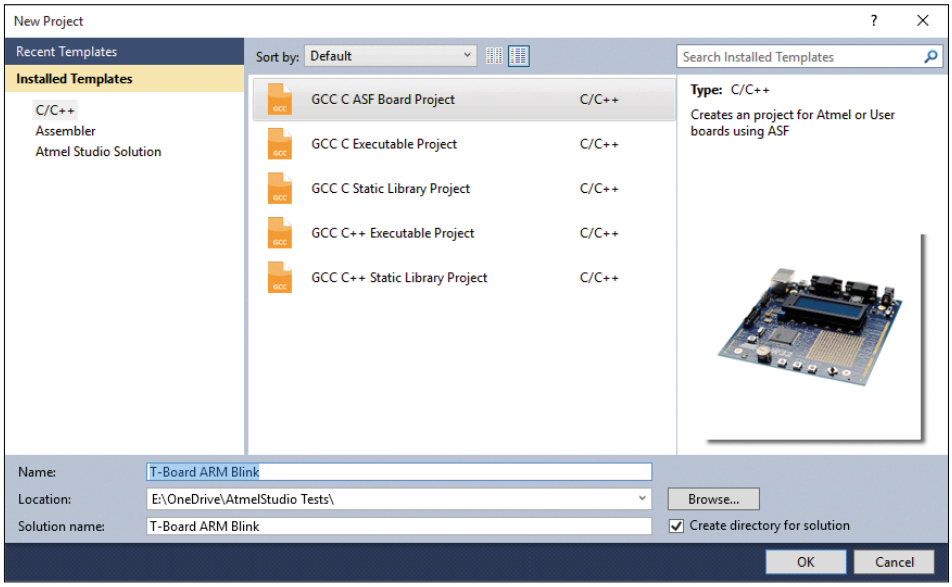


Figure 4. Initial dealings with the GCC ASF Board Project.

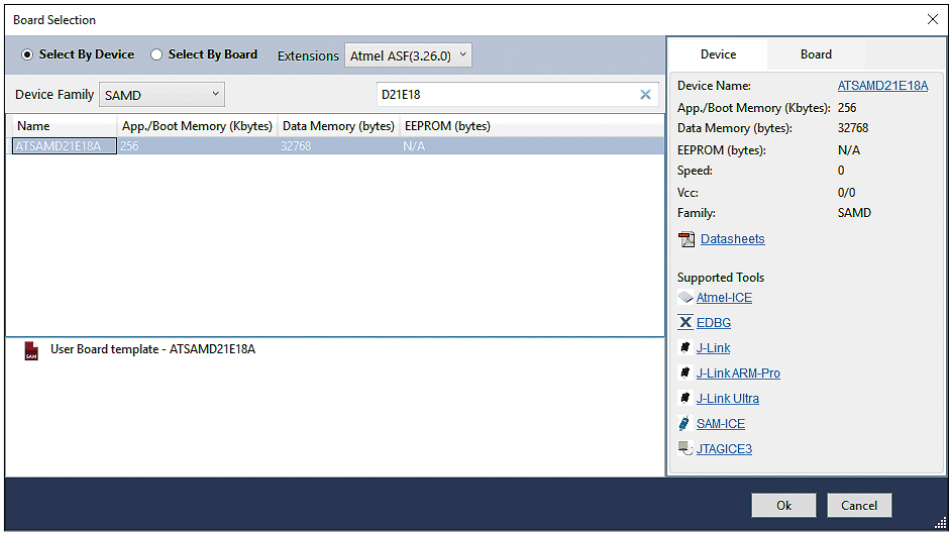


Figure 5. Selecting a non-AVR micro.

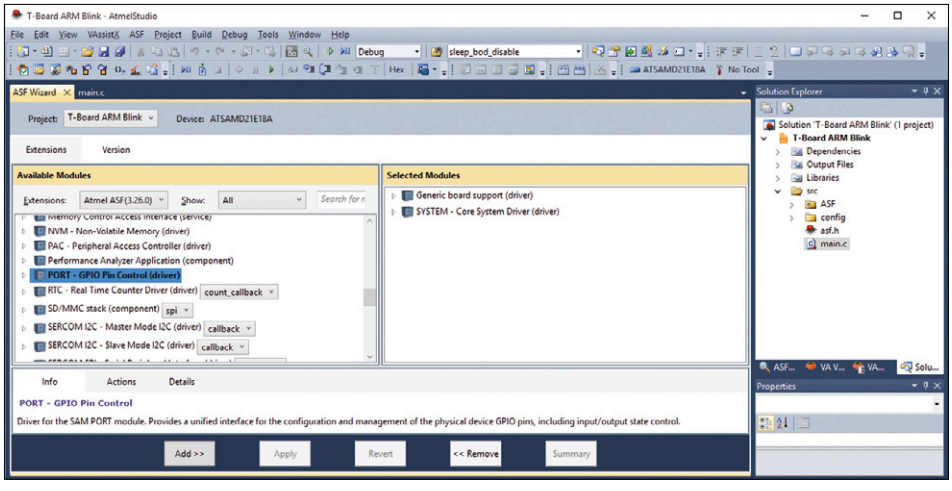


Figure 6. Here you add the PORT driver.

(see **inset**: Alternative IDEs), and using the Atmel Software Framework (ASF), which we found the quickest way to get up and running given the complexity of an ARM micro. In the "*From 8 to 32 bits*" series ASF is also used, although with an Atmel SAMD20 XPlained Pro board. We have to do things slightly differently to set the project up here, but from there on it's fairly straightforward.

Follow these steps:

1. Open Atmel Studio, and create a new project — choose the *GCC ASF Board Project*, name it and choose where to save it (**Figure 4**).
2. Select the device: The T-Board ARM uses an *ATSAMD21E18A*, so choose that. As there aren't any Atmel development boards for this microcontroller, you must select *User Board Template* and click OK (**Figure 5**).
3. An empty project has been created, with *main.c* under the *src* folder in the Solution Explorer.
4. Next we need to add the ASF PORT driver to the project. This provides a series of functions to allow us to access the pins more easily. Choose *ASF Wizard* from the *ASF* menu.
5. On the popup dialog (**Figure 6**) you'll see available modules on the left, and selected modules on the right. Scroll down the left and highlight *PORT – GPIO Pin Control (driver)*, then click Add and Apply. You've now added the PORT driver to your project.
6. Now add the code from **Listing 1** into the *main.c* module.

Step 3: Flash the T-Board

Your project is now ready to be compiled and uploaded to the T-Board:

1. Compile the project (F7), ensuring that there are no errors of course.
2. Connect power to the board through the DC jack (assuming your programmer doesn't provide power to the project). The Atmel ICE that I use is one of these that doesn't power the target. If you use a JTAGICE3 programmer then be sure to have relevant information on using it.
3. Attach the programmer to the SWD header on the board, and then connect to the USB port on the PC.
4. Select the programmer from the *Tools* menu, under *Device Programming.*

Web Links

[1] Atmel USB 2 GO: www.atmel.com/Images/Atmel_11201_USB-OTG-Like-Connector-Implementation_SAM9G-SAM9X-SAMA5D3_Application-Note.pdf

[2] T-Boards low power article: www.elektormagazine.com/140413

[3] Project resources page: www.elektormagazine.com/150059

[4] Atmel Studio: www.atmel.com/tools/atmelstudio.aspx

[5] www.der-hammer.info/terminal/

5. Finally upload the program: choose *Start without Debugging* from the *Debug* menu.
6. You should now see a blinking LED!

The next steps

Now that you've seen the T-Board in action, you're probably chomping at the bit to start additional projects. If you're new to ARM Cortex microcontrollers, I'd challenge you to take one of the previous projects in the "*From 8 to 32 bits*" series and use the T-board in place of the Xplained Pro board. You'll cut down the number of jumpers, making the projects easier to design and troubleshoot.

Meanwhile, co-designer Viacheslav here presents another kick-off program specially written on the occasion of this "hardware" dominated article: see **Listing A** and the **inset:** My/Your/The/A First Project on the next two pages.

We hope that you enjoy using the ARMed T-Board both as a learning tool and a way to get your next project prototyped and running more quickly! ⏮

(150059)

Mind U! **...continued overleaf**

Listing 1. Blink-a-LED (in ARM T-Board Style)

```
#include <asf.h>

#define LED_PIN PIN_PA07   //LED is connected to PA07

//Function Prototypes
void configure_port_pins(void);


int main (void)
{
   system_init();   //ASF Routine to initialise the system, clocks, etc.

   configure_port_pins();   //Configure GPIO pins

   SysTick_Config(system_gclk_gen_get_hz(GCLK_GENERATOR_0));       //Enable SysTick interrupt:
                                                                   //reads frequency to sets interrupt at 1 Sec

   while (1)
   {
      //Nothing needed here as we are using interrupts to toggle the LED
   }
}


//Function to configure the port pins
void configure_port_pins(void)
{
   struct port_config config_port_pin;                //Structure used to store parameters
   port_get_config_defaults(&config_port_pin);        //Read the current configuration of the pins into the Struct
   config_port_pin.direction = PORT_PIN_DIR_OUTPUT;   //Set Struct to indicate pin is an Output
   port_pin_set_config(LED_PIN, &config_port_pin);    //Use the struct to set the direction for the LED's pin
}


//Interrupt Handled for SysTick Interrupts
void SysTick_Handler(void)
{
   port_pin_toggle_output_level(LED_PIN);   //Toggle the LED's Pin
}
```

My/Your/The/A First Project

By **Viacheslav Gromov**

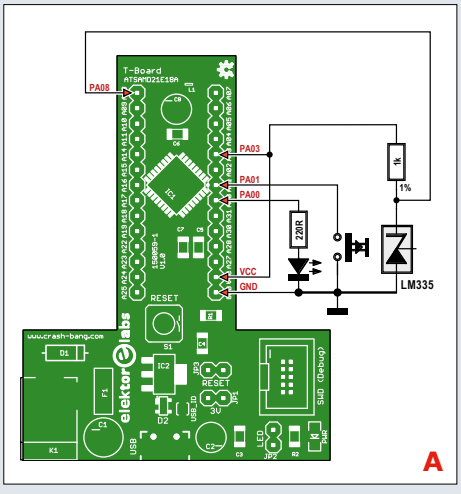
Let´s make a simple project! Let's switch one LED on and off, sample one pushbutton, and measure the temperature with an LM335 and ADC. We want to control everything from the PC through USB with a no-frills terminal emulator program.

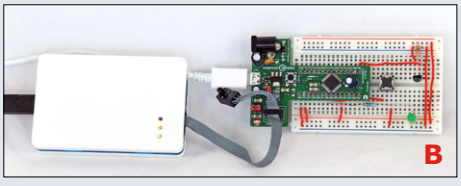
1. The Hardware

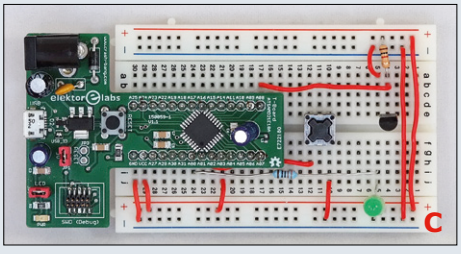
Figure A shows how the external parts connections to the ARM'ed T-Board, **Figure C**, the practical realization. With all parts connected up, you can use a USB-A connector to USB Micro-B connector cable to connect the T-Board to your PC. This type of cable is often used for charging smartphones or tablets, so you should have one ready to hand. You also need to connect a debugger like Atmel ICE to the T-Board on the SWD (Debug) connector and of course also to your PC (**Figure B**).

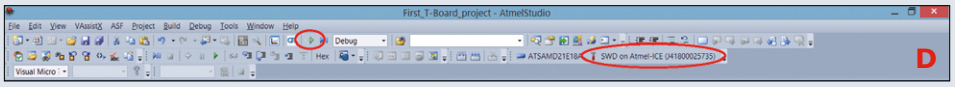
2. The Software

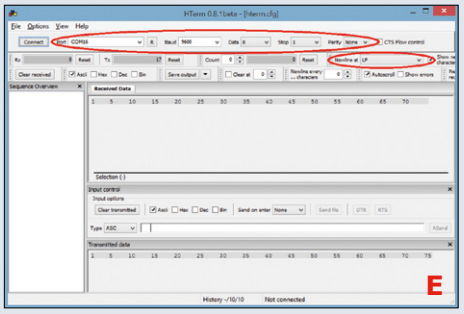
First, download the Atmel Studio project named "First_T-Board_project" from [3].

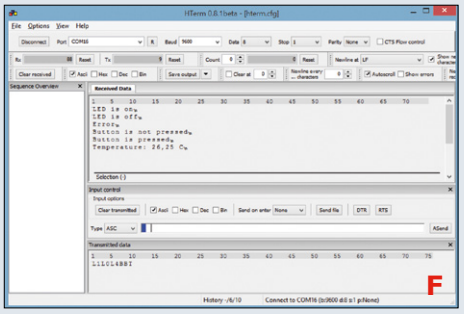


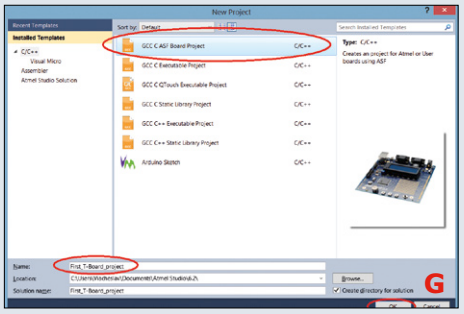


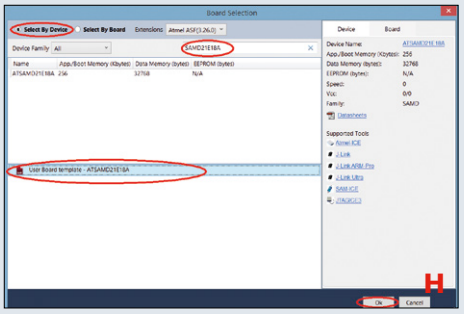












Next, open it in Atmel Studio 6, which you can download under [4] after a minor registration or as a guest, if you haven't installed it yet. If you are not familiar with the Atmel SAM D family and/or Atmel Studio look in our *ARM Microcontrollers for Beginners* course.

The program starts with the declarations of the variables and arrays, function prototypes and configuration-functions for the peripherals. After that, in the main function, the system, the interrupts and the delay-functions get initialized and all the configuration-functions get called. **Listing A** shows the endless loop with the main part of code. The whole code is in a `switch` statement which receives one ASCII-character with `udi_cdc_getc()`on the USB and chooses the correct branching to follow. In **Table A** you can see all characters (combinations) you can send later to the board with the terminal program. For each of the three letters there is a `case`. The number of the `case` isn't the letter, it's the ASCII-number for the letters: 76 stands for "L", 66 for "B" and 84 for "T". Please note that these are all capital letters.

In the first `case`, 76, there is an `else-if` statement which turns the LED on, when variable `number` is 1, and off, when `number` equals 0. The variable `number` gets the (second) character from the USB with `number = udi_cdc_getc();` before this. Also, "LED is on" and "LED is off" will be sent with `udi_cdc_write_buf(&buffer, x)` over the USB. For arguments it needs only a pointer to a string with the data (buffer) it has to send, and the number of characters due sending (x). But here, we don't use a pointer to a buffer — to keep things simple, we embed a text like "LED is on"

Table A. Mini Command Set

L	**1**	Sending 'L1' (for: LED) causes the LED to be switched on and you receive a message "LED is on". Sending 'L0' switches the LED off and you get "LED is off". Sending 'L' with a number other than 0 or 1 returns: "Error".
	0	
B		Sending "B" (for: Button) causes the T-Board to respond with "Button is pressed" or "Button is not pressed".
T		Sending 'T' (for: Temperature), causes the T-Board to emit a string with the currently measured temperature x, like: "Temperature: x C".

direct into the command like this: `udi_cdc_write_buf("LED is on\n ", 10);`. The \n stands for new line. If this variable is found to contain another value than one or zero, "Error" will be sent to the computer.
In the next `case`, 66, an `if` statement processes the state of the button pin, PA01, with: `port_pin_get_input_level(PIN_PA01)` and sends either a "Button is pressed" or "Button is not pressed" message over the USB.
The last *case*, 84, measures the voltage on ADC channel 16 (PA08) supplied by the LM335, and calculates the temperature from this voltage. A string like "Temperature: x C" gets sent over USB in a more circumstantial way due to the specific requirements of the float-temperature-value described in the ARM Course.

3. The First Test

Let's try it out! Please select your debugger above on the right and start compiling and transfer the program with a click on the green "Start Without Debugging"-button (**Figure D**). After the MCU is programmed, the T-Board will register on your computer as a "Communication Device Class ASF example" – yep, a virtual serial interface. Maybe your computer needs some time to install the right drivers. After it's done, you can open the device manager on the PC to see which COM port number's been assigned to your T-Board under "Ports (COM & LPT)". Now open a terminal program like HTerm [5], and set up the comms like in **Figure E**:

- COM port number (see Device Manager)
- 9600 baud
- 8 data bits
- 1 stop bit
- no parity
- new line on LF

With all settings adjusted and DTR On, connect with the board and test it with our command-capital letters like in **Figure F**. Congratulations, your first project with the ARM'ed T-Board is running now! That was all copycat though – now on with building a new project.

4. Create a new project with the ARM'ed T-Board & Atmel Studio

The project generation is nearly the same as described in the first part of the ARM course. You also need to select a "GCC C ASF Board Project" under File/New/Project… (**Figure G**), but in the next step, after you pressed "OK", it's different — you now need to select our MCU (ATSAMD21E18A) by Device like in **Figure H**. After you pressed "OK" once more the c project is generated. Now you can import the ASF libraries you need in the ASF Wizard and write your code!

Listing A. The endless loop of our get-u-going program looks like this.

```
while(1){
switch (udi_cdc_getc())
{
	case 76: //if received "L"
	number = udi_cdc_getc(); //get the second character (a number)
	if (number == 49) //if received a one
	{
		port_pin_set_output_level(PIN_PA00, 1);	//put the LED on
		//send "LED is on" with a "new line" on USB
		udi_cdc_write_buf("LED is on\n", 10);
	}
	else if(number == 48) //if received a zero
	{
		port_pin_set_output_level(PIN_PA00, 0); //put the LED off
		//send "LED is off" with a "new line" on USB
		udi_cdc_write_buf("LED is off\n", 11);
	}
	else //if received a wrong number (not 1 or 0)
	{
	udi_cdc_write_buf("Error\n", 6); //send "Error" and a "new line" on USB
	}
	break;
	case 66: //if received "B"
	if(!port_pin_get_input_level(PIN_PA01)) //get the level on PA01 (button)
	{
		//if pressed, send "Button is pressed" and a "new line" on USB
		udi_cdc_write_buf("Button is pressed\n", 18);
	}
	else
	{
	//if not pressed, send "Button is not pressed" and a "new line" on USB
		udi_cdc_write_buf("Button is not pressed\n", 22);
	}
	break;
	case 84: //if received "T"
	adc_start_conversion(&adc_instance);  //start ADC-conversion
	//read and save the conversion result
	while(adc_read(&adc_instance, &data) == STATUS_BUSY){}
	//calculate the temperature
	temperature = (25 + (data * 0.000805 - 2.945) / 0.01) * 100;
	//reformat the result in a buffer
	sprintf(temperature_string, "%i", temperature);
	//send "Temperature:" with USB
	udi_cdc_write_buf("Temperature: ", 13);
	//send the temperature on USB
	for(i = 0; i < 4; i++){
	if((temperature >= 1000) && (i == 2)) udi_cdc_putc(44); //make a comma
	if((temperature < 1000) && (i == 1)) udi_cdc_putc(44);  //make a comma
	udi_cdc_putc(temperature_string[i]);  //send a digit on USB
	//send "C" and"new line" on USB at the end of transmission
	if(i == 3) udi_cdc_write_buf(" C\n", 3);
	}
	break;
	}
}
```