Controlling the TI® SensorTag with the GA144

The Texas Instruments SensorTag provides an excellent set of devices on a small, inexpensive platform. The devices consist of the CC2541 which incorporates a BLE (Bluetooth Low Energy) radio, and an array of I^2C sensors: 3-axis accelerometer, 3-axis gyroscope and magnetometer, as well as sensors for barometric pressure, temperature and humidity. The SensorTag is powered by a CR2032 battery bucked down to 2.1V, the lowest common supply voltage for all the SensorTag's devices.

The purpose of this exercise is to demonstrate various choreographic and programming techniques, primarily toward minimizing energy consumed per unit useful work. The most powerful techniques are to minimize the number of instructions that must be executed, resulting in duty cycles of 10^{-3} to 10^{-5} cycles, and implementing I/O in such a way as to minimize power used externally. In this application we demonstrate the GA144's ability to maintain continuous situational awareness while operating from a coincell at an average power of 363 µW. Further economies may be realized through judicious system configuration changes.

Contents

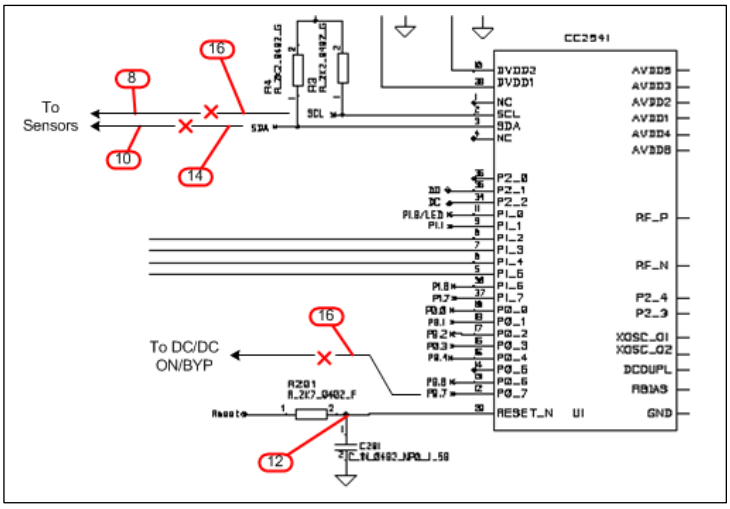
1. Test Platform

The purpose of this experiment is to interface a naked GA144 with the devices on a Texas Instruments SensorTag ("ST" for short herein.) We modify a ST so that the GA144 may control all of its devices, mount the ST on an EVB001 Evaluation Board, and connect it to the Target chip in such a way that the Target Chip is powered by the battery on the ST. Once software has been loaded into the Target chip, external connections to the Evaluation Board may be removed and the EVB, with its ST, may be operated in a mobile fashion, communicating with a PC using Bluetooth® Low Energy. This section describes the modifications and connections made to construct the platform.

The ST has a CR2032 3V watch battery powering a 2.1V buck regulator that is pretty efficient; our measurements lead us to estimate 93% at low power. Without the GA144 and running the default ST firmware, the draw of the ST is about 221 µA from the battery and 294 µA (estimated based on observed efficiency) from the regulator. We have reduced this by properly initializing the thermopile (which resets to an active state) and by replacing the firmware in the ST's onboard 8051 microprocessor. The regulator does not appear to work and successfully supply 2.1V throughout the 250 mAh life of the battery. We will obviously have to be very judicious in our use of each nanojoule to run for any length of time on this battery, so it is an excellent platform on which to demonstrate aggressive techniques for conserving energy.

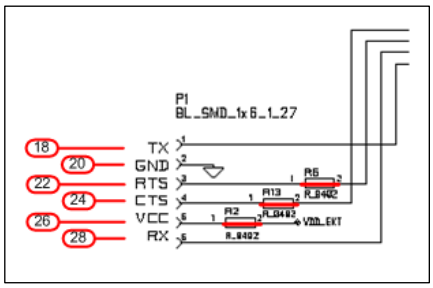
1.1 Modifications to the SensorTag

The first change is to separate the I^2C bus at the CC2541 chip. Connect wires 14 and 16 to SDA/SCL at R4 and R3 for communication with the CC2541. Cut the traces farther along after the pull-up resistors and connect wires 10 and 8 to SDA/SCL for communication with the sensors. Note that there are no pull-up resistors on this section of the bus.

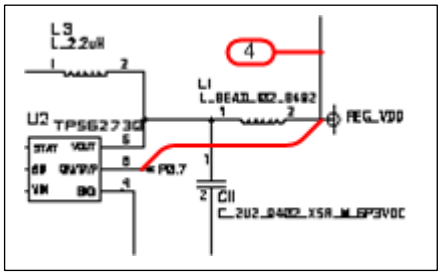


Connect wire 12 to the CC2541 side of R201/C201 so that we can perceive the reset button and so that we may reset the CC2541.

Cut the trace from CC2541 P1.1 to the gyro; this normally supplies power to that sensor. Connect wire 2 to the CC2541 side and wire 30 to the gyro side, so that we have control over that sensor's power.



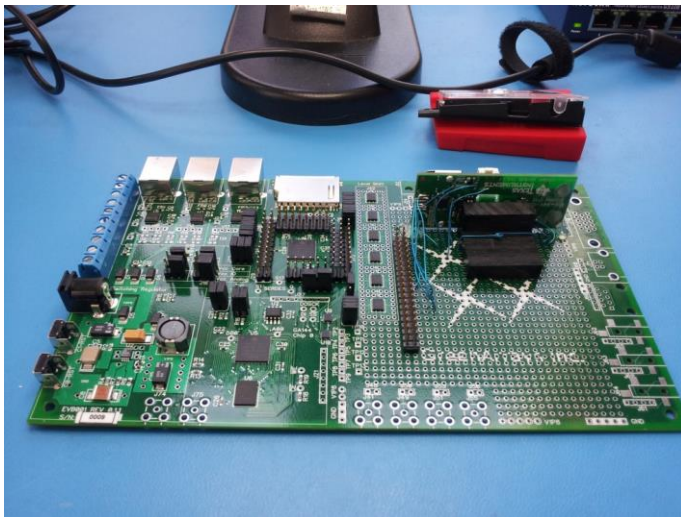
Connect wires 18 through 28 to serial edge connector pins 1 through 6 for reprogramming of the CC2541. The resistors shown in series with VCC, RTS and CTS are not populated on the board; to supply external power and/or communicate with the CC2541 chip, wires must be soldered across the three 2-pad patterns in question.



At the DC/DC Converter, connect wire 4 to REG_V_{DD} out at the ferrite bead and jumper the ON/BYP pin 5 to REG_V_{DD} so that the regulator is always enabled, powering the sensors and the Target Chip, rather than being controlled by the CC2541.

1.2 Mounting the SensorTag on an EVB001

The SensorTag is secured to the prototyping area of the Evaluation Board and wires are connected to the underside of a 40-pin, 2-row header, as shown in this image. Only the uppermost 30 pins are used, and the odd numbered pins on the left side of this photo are all connected to ground. (Pin 4 is connected to pin 32 and a jumper installed 30-32 once the GA144 is ready to control the I^2C bus.)



Pin assignments are as shown in the table below, as viewed in the photograph to the right.

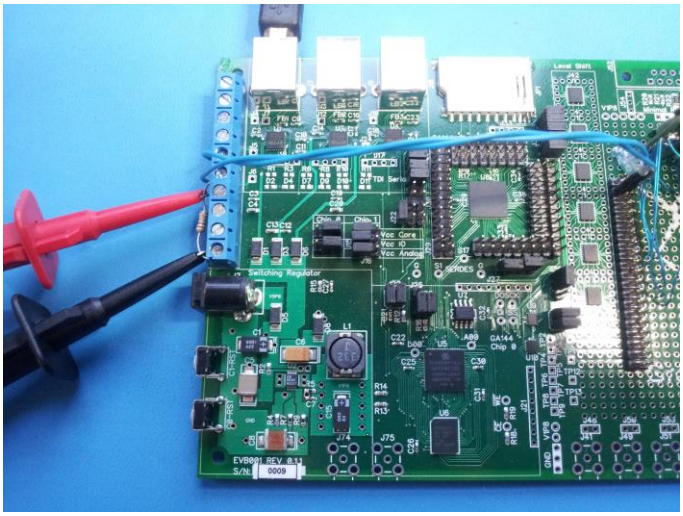
The Target Chip will be powered by 2.1V from the buck-regulated CR2032 battery on the ST. To prepare for this, run a twisted pair from pins 4 and 3 of the ST header to pins 5 and 6, respectively, of barrier strip J1 (Ground is the second of each of these pin-pairs.)

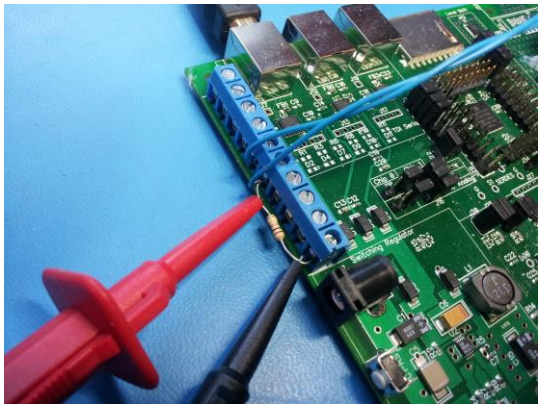
Then insert a 1Ω, ¼W resistor between pins 1 and 5 of the barrier strip J1. It is worth hand-picking this resistor to be within <1% of one ohm so you can avoid doing arithmetic to interpret current readings, which will be made by measuring the voltage across this resistor at 1 µV per µA.

The choice of resistor value is based on the premise that peak current in this application will be a small number of tens of mA, giving a voltage drop of tens of mV, while giving a signal amplitude high enough for practical measurement with a good quality meter. If the application were going to use more peak power then a smaller value resistor would be needed to avoid excessive voltage drops.

The images below illustrate this power connection as well as the jumper settings for isolating the Target Chip discussed in the next section.

ST Hdr			Reset Volts
1	2	CC2541 Pin P1.1	2.085
3	4	REG_V_{DD} (2.1V)	2.085
5	6	CC2541 ON/BYP (P0.7)	0.0
7	8	Sensor SCL	0.0
9	10	Sensor SDA	0.0
11	12	CC2541 Reset/	2.080
13	14	CC2541 SDA	2.085
15	16	CC2541 SCL	2.085
17	18	Serial pin 1 TX	2.085
19	20	Serial pin 2 GND	0.0
21	22	Serial pin 3 RTS	0.0
23	24	Serial pin 4 CTS	0.0
25	26	Serial pin 5 V_{CC}	0.0
27	28	Serial pin 6 RX	2.085
29	30	Power to gyro chip	0.0
30	32	REG_V_{DD} from pin 4	2.085





1.3 Configuring EVB001 for Isolated Target Chip Operation

It's important in this experiment to isolate the Target chip from the rest of the Evaluation Board. Otherwise power might be consumed from its rails by such things as powered-down circuitry. The following should be done immediately:

1. Remove jumpers 3-4 and 5-6 of J22 to disconnect Target Reset from FTDI chip and the power-on reset circuit.
2. Remove jumpers 9-10 and 11-12 of J23 to disconnect Target node 708's pins from FTDI chip.
3. Move J14, J15 and J16 from pins 2 and 3 to pins 1 and 2, selecting 2.1V previously connected to barrier strip J1 pin 1 for power to all three Target chip buses.
4. Connect J23 pin 10 (708.17) to J32 pin 5 (709.ao), J23 pin 12 (708.1) J32 pin 10 (717.ao), and J22 pin 6 (Target RESET-) to J32 pin 8 (713.ao). If wire wrap wire is used, make short wraps on these pins so there's room left for jumper wires. The purpose of these connections is to provide programmable pull-up devices for the sensor I^2C bus (node 708) and for the RESET- signal of the Target chip.
5. During testing, if node 705 is not used for anything else it must still be initialized so that pin 705.17 is at high impedance. On the EVB001, this pin is pulled up to the Target V_{DD}I rail with a 1k resistor to prevent SPI boot. With the default weak pull-down enabled this pin pulls about 52 µA at 2.1V.

Jumpers 1-2 of J22, and jumpers J34 and J35, must be installed while programming the Target chip. *When working with the rest of the EVB powered, it is unnecessary to remove them since the leakage through the Host chip protection diodes is negligible and is dwarfed by the 10 µA of leakage on this particular chip at 2.1V. In this condition the programmable pull-up of RESET- should be disabled. For operation with the EVB unpowered, the RESET- pull-up should be enabled and at minimum the RESET- jumper should be removed after booting and before removing power from the EVB since there is otherwise a substantial current (on the order of 1 mA) drawn by the unpowered Host chip.*

1.3.1 Initial Testing

With a voltmeter across pins 4 (REG_V_{DD}) and 3 (ground), insert battery as shown in the TI documentation and observe 2.085V on pin 4. Press button on side of the ST and observe rapid flashing on green LED D1. This indicates that the ST is not severely damaged.

Connect a PC to USB port A (J3) on the EVB001; in this case it is recognized as COM9. Make the normal configuration changes to COM9 on the PC so it can run at full speed. Run **`9 selftest`** on the Host chip, and **`9 autotest`** on the target chip to verify both chips are working and that the reset and synchronous boot lines are properly connected between the chips. Measure voltages on all pins of the header and verify they are as recorded in the table above.

1.3.2 arrayForth IDE Operations

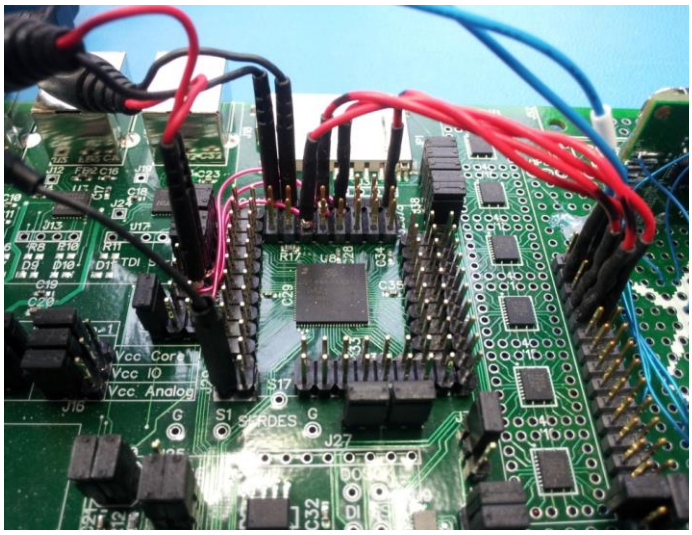
`bridge load` compiles a version configured for two chips with a default path 0 that can reach all nodes of the Target chip, with or without the polyFORTH virtual machine present on the Host chip. Extended node numbering is supported in the form **`cyyxx`** where **`c`** is zero-relative chip number; thus nodes 000 through 717 are on the Host chip, while nodes 10000 through 10717 are on the Target chip.

`talk` is extended in the **`bridge`** configuration. After resetting the Host chip and programming its node 708, the Target chip is reset and a transparent bridge for carrying port communications between the chips is installed in node 300 of each chip, dedicating those nodes to this purpose until next reset.

With the port bridges installed, the **`up`** ports of node 400 on each chip are logically connected as though they were a simple COM port. Port read/write communications, such as IDE, are basically transparent across this connection except that data transfers take 100 or more times longer, no flow control is supported, and polling of **`io`** by node 400 has limited usefulness. Node 300 will seem to be writing when a word sent by the other chip is waiting to be read, it will seem to be neither reading nor writing during serial data transmission in either direction, and it will seem to be reading at all other times regardless of the state of node 400 in the other chip.

1.4 Connecting the I²C Sensor Bus

Using twisted pairs we connect 708.17 to ST header pin 8 (SCL) and 708.1 to pin 10 (SDA). Jumper ST header pin 30 to pin 32 to provide 2.1V power to the gyro; this is necessary since when that chip is not powered it draws current from the SCL line through its pull-up resistance. When set up this way and the test code is loaded far enough to turn on the equivalent of 2.2k pull-ups on SDA and SCL, and pin 705.17 is at high impedance, the Target chip on the board used for this exercise draws a total of 10 µA from the ST power supply for all three of its buses while suspended.



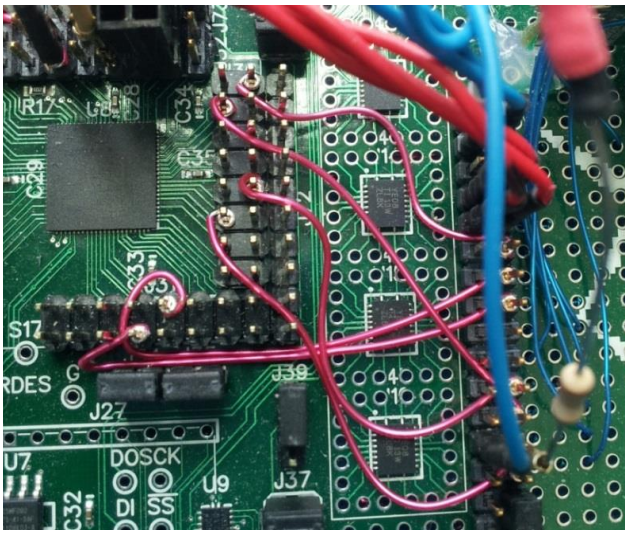
The image to right shows this set of connections as well as scope probes that will be used to observe activity on the I²C bus. Note the removal of jumpers to isolate the Target Chip electrically.

1.5 Connecting the CC2541 BLE I²C Bus and Control Lines

Communicating with this BLE chip requires four handshake lines in addition to its dedicated, 2-wire I²C bus. Make the following connections:

ST Hdr		Purpose	EVB Signal	EVB Pin	Remarks
11	12	CC2541 Reset/	517.17	J36.2	BLE RST-
13	14	CC2541 SDA	008.1	J31.10	SDA
15	16	CC2541 SCL	008.17	J31.7	SCL
21	22	Serial pin 3 RTS	317.17	J36.3	Outbound buffer available
23	24	Serial pin 4 CTS	417.17	J36.10	Inbound data ready
27	28	Serial pin 6 RX	217.17	J36.11	BLE WAKE- (~20k pull-up)

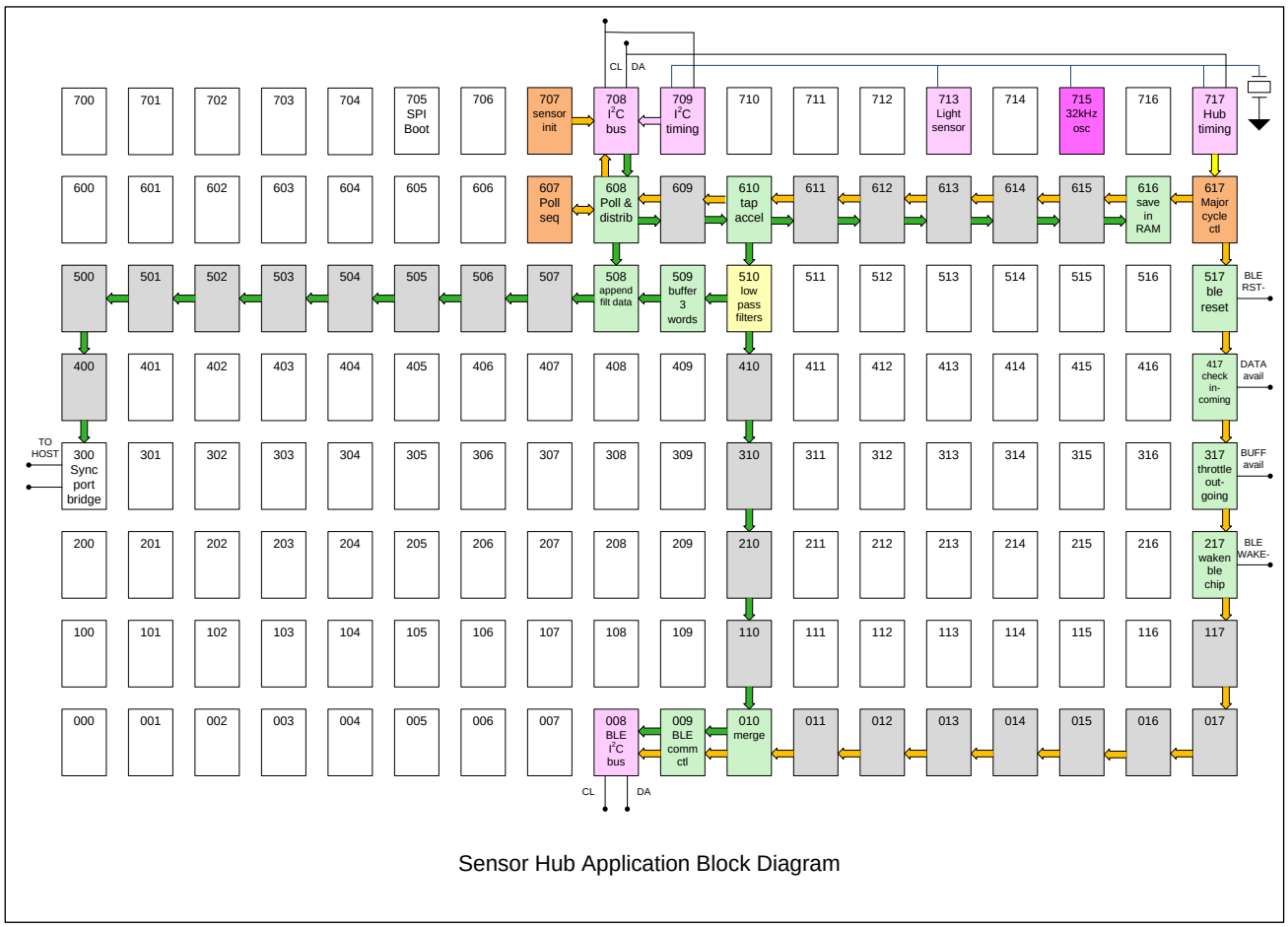
Ideally each of these connections will be a twisted pair. In this illustration, the connections were made with wire-wrap to verify that we could "get away" with it.



In this implementation we did not replace the 2.2k pull-up resistors on this bus because it was not practical to do by now. When building a new platform they should be replaced by a larger value, more like four times this resistance, to save energy; or they could be removed and pull-up provided by the remaining unused DACs on the Target chip. Unfortunately, the firmware for the CC2541 needs to stretch the clock, or we would not have to waste energy heating the pull-up resistors.

2. Architecture

The key to minimizing energy consumption with our chips is to minimize duty cycle. This application is a great example of that because none of these sensors demand an especially high processing frequency. A data acquisition and processing interval of 100 ms should be quite sufficient. During a 100 ms processing interval, each node would have time to execute up to 60 million instructions. However, the actual processing requirement is more likely to be on the order of 600 instructions per interval. Thus, each node processing data at the 100 ms interval will be running at a duty cycle on the order of 1/100,000. At 100% duty cycle a node can draw single digit milliwatts, but at these low duty cycles its average power will be in tens of nanowatts. This is comparable with the leakage of a GA144 node. A very small number of nodes will need to be running at a shorter time interval, to measure time and to operate the I^2C bus, but even these nodes will have duty cycles on the order of 1/1000. So our application must be designed primarily to avoid measuring time with program loops, to spend most of the time with all nodes suspended, and then to fight tooth and nail to minimize I/O power. The resulting gross architecture is shown in this block layout of the GA144:



Sensor Hub Application Block Diagram

In this drawing, pink arrows and nodes indicate 65 kHz stimuli; yellow arrows are stimuli at 100 ms intervals. Orange arrows are command messages and green indicate the flow of data, also at 100 ms intervals.

Node 715 accounts for most of the energy used in this application. It is a 32.768 kHz crystal oscillator and the node runs at twice that frequency since it must wake up and stimulate the crystal on each rising and falling edge of the signal. We selected this node for the oscillator because its pin is available to three analog nodes (709, 713, 717) as a phantom wake-up signal; so the energy cost of distributing this time base to three other nodes is zero, and there is no energy cost in those three nodes when they are simply ignoring the signal.

Node 709 is used to provide clock timing for the I^2C bus, thus saving considerable energy compared with program loops, and is active at 65 kHz only when the bus is active.

Node 717 is used to delay for the next processing interval and is active at 65 kHz when measuring time. Thus, like node 715, 717 is active most of the time and so it's important that it do as little as possible to keep the duty cycle down. When node 717 is only providing time delay, its average cost is on the order of 2 µA. It could also serve as a Real Time Clock if necessary, at the cost of additional energy for executing the instructions to maintain a time counter.

Node 713 may be used to measure ambient light as the voltage from a photovoltaic cell and is connected to this time base to economically measure time intervals for the A-D converter.

The three analog nodes on the top edge of the chip also provide programmable, active pull-ups for the I^2C bus lines and for the chip's reset pin. These have been optimized to source the minimum necessary current for reliable operation, thus minimizing power during bus operations; the resistor values of 2.2 kΩ on the ST are very wasteful of power and produce rise times about four times faster than required by the specifications.

Nodes 707, 708, 607 and 608 control the I^2C bus operations; they initialize the sensors to the desired state on boot, and poll raw data on command.

Node 617 determines the major processing interval of this application (100 ms), waiting for a time delay using 717 then sending a message across the 6xx node row to 607, which then polls all the sensors and causes raw data to be moved back across the 6xx row for further distribution and processing, and also to node 508 for raw diagnostic logging. In addition, node 617 generates commands sent down the right hand column of nodes for communication with the BLE chip (see discussion below.) To allow all functions to start in a coherent manner, including the BLE chip, node 617 delays 5 seconds by the crystal before initiating normal polling and BLE reporting operations.

The 54-node (6 rows, 9 columns) array with corners 009 and 517 are available for processing the sensor data, maintaining situational awareness, making decisions and so on.

Plumbing can be provided for diagnostic connection to node 400, to another I^2C or SPI bus using node 8, to the SERDES in nodes 1 or 701, to parallel devices using nodes 7 or 9, or to simple devices using the GPIO or analog nodes on the left and right edges of the chip.

During development, bridge mode IDE through port 300 is useful, and the code provided with this Application Note includes plumbing for diagnostic transmission of data to the Host chip across that bridge. For production use, booting would likely be done using SPI flash through node 705.

3. Implementation

All of the code is found in blocks 960..1080 of the arrayForth system. This section provides a walk-through of key parts of this software.

3.1 Time Base for Low Duty Cycle

We interrogate the sensors 10 Hz, and the I²C bus as noted later must be run no faster than 400 Kbits/s. These are very low frequencies relative to the speed of our nodes, and to produce such long time delays by program loops consumes current in milliamps. Instead, we use the 32.768 kHz crystal oscillator described in AN002 to measure the major polling intervals and to pace the I²C bus clock. The oscillator is placed in node 715 and the signal it generates is available at minimal power to nodes 709, 713 and 717 due to low-capacitance internal connections. These nodes provide their timing functions only when needed and are suspended when not. As a result, the average current used by the GA144 including its powering of the I²C bus pull-ups between polls is very small... 14 to 15 µA of leakage, 28 or so µA to run the oscillator and another 2 µA for node 717 to count down for the next major cycle, for a total on the order of 45 µA or 95 µW average at 2.1V. We may be able to find ways to reduce the current used in driving the crystal. Here is the oscillator code:

```
32.768 khz watch crystal from 715.17 to gnd,
,
this code takes 27 ua at 2.1v using a seiko,
vt200f-12.5pf-20ppm crystal, 42 cents unit qty
from digi-key.,
,
-osc tries exciting the crystal with n cycles
of period k returning nonzero if it didn't,
come back high after last cycle.
clang searches for resonant frequency over a,
reasonable range. initially we use 5000 cycles
and may be able to shorten this. when we find
resonance, falls thru into prep which sets up
registers and finally we camp in run which is
the low power, low duty cycle oscillator.
try is test code for finding resonance.,
,
do not connect any kind of conventional probe
to the crystal; this oscillator will not work
if you load it down even that much.
```

```
980 list
715 xtal osc reclaim 10715 node 0 org,
-osc kn-f 00 io b! for,
..02 30000 !b dup .. 2/ dup for unext 06/1,
..20000 !b .. over 1 and .. + for unext next,
..dup or !b dup 30000 for,
....drop @b - -while next ;,
......then dup or pop drop ;
clang 14 14450 200 for dup 5000 -osc while,
..drop 1 . + next clang ; then pop drop ;,
prep 1F 0 20000 800 30800 0 20000 800 30800,
..dup up a! drop
run 2C !b !b @ drop run ;
try 2E dup 5000 -osc over 1 . + ;,
34
reclaim
```

3.2 Low Energy I²C Bus for Sensors

Node 708 controls the I²C bus used to communicate with the sensors, using its two pins with 17 as clock and 1 as data. Node 709, when requested by node 708, listens to the crystal oscillator and provides bus timing signals. Node 709 also pulls up the clock line using its DAC as a programmable current source, and node 717 pulls up the data line. The DACs were initially set to value x0D giving the equivalent of the 2.2 kΩ pull-ups used on the ST. This gave extravagantly conservative rise times on the order of 60 to 66 ns. Later we reduced the DAC values to 3, giving rise times of 240 to 250 ns. The result was a decrease of a factor of three in average current used to run the I²C bus. To put this in perspective, the average ST *battery* current used by active accelerometer, GA144 and polling of four devices at 50 Hz was reduced from 758 µA to 279 µA. When polling the bus at our nominal frequency of 10 Hz, average *battery* current then became about 116 µA of which 40 to 50 are attributable to I²C bus activity. (Unfortunately, the ST has one sensor, the magnetometer, which uses clock stretching. If this were not the case we could drive the clock line push-pull and save yet more energy.)

On boot, node 708 executes its left port to accept register initialization sequences for all the sensors from node 707, which has room for 25 8-bit register settings. (If more were needed, memory in node 706 could be used; or, if two

modes of operation are contemplated, node 706 can supply an alternative initialization sequence.) After setting initial values in these registers, node 707 directs node 708 to execute its down port where the application hub will deliver on-line polling commands. Node 707 suspends until next boot. The code in node 708 provides a set of simple primitives for port execution.

```
i2c bus drive using xtal oscillator for timing
..via node 709. single ended version because,
..magnetometer does clock stretching.,
,
set sets pins then waits 1/2 bit time.
!hi set with wait if clock stretched.
c+d+ c+d- etc set bus state and delay. any,
..clock rise may be stretched.
w1 xmit bit16 i1 rcv bit1 both shift left.
w8 xmit/shift bits 15-8, ret nak bit1.
w16 xmits bits 15-0.
r8 shifts byte into bits 8-1.
strt or restart chip a;*stop ends frame.,
,
port executable functions...,
....all of these require port execution,
....help to deliver args and results.,
@regs starts burst read of chip a reg i,
..@w+ reads 16 bits msb first into bits 16-1,
..@w. @b. read final word/byte to 8-1
!af starts reg write.,
..use w8 w16 as needed then stop.
```

```
990 list
708 slow i2c reclaim 10708 node 0 org,
set n 00 !b @p ! ; .. ;
!hi n 02 dup begin over set,
..drop @b -until drop drop ;
c+d- 06 2 !hi ;*c-d+*c-d* 08 20000 set ;
c-d- 0A 20002 set ;*c+d+*c+d* 0C 0 !hi ;,
,
w1 n-n' 0E 2* -if*wnak c-d+ c+d+ ;,
..*wack then c-d- c+d- ;
w16 leap*w8 h.l-l.s then 7 for 2* w1 2/ next,
..*i1 n-n+ c-d* c+d* @b 2 and or ;
strt*rest a.x-x.nak 1D c-d+ c+d+ c+d- w8 ;
stop 21 wack c+d+ @p ! ; .. dun ;
!af a.i-s 25 strt w8 ;
r8 -n 27 7 for 2* i1 next ;
zr8+ -n 2C dup dup or*r8+ n-n 2D r8 wack ;
@regs a.i-ss 2F dup 100 or,
..push !af pop rest ;
@w+ -n 33 zr8+ r8+ ;
@w. -n 35 zr8+*@b. n-n' r8 wnak stop ;,
39 reclaim exit
```

This code must support clock stretching to accommodate the magnetometer; this feature is costly in terms of energy. Timing is provided by node 709 using a simple state machine that is exercised by port execution:

```
this dac actively pulls sensor scl line up and
..provides low energy timing for slow i2c bus.
..simple state machine receives one of two,
..instrs from 708 thru right port.,
,
pu is io for wait-high, -pu for wait-low.,
,
idl state between bursts of i2c actvity. turns
..pull-up on and expects a return instruction.
..that does not delay 708 but we delay next,
..stim by at least one clock phase and enter,
..active state.,
,
act expects return instructions, delaying each
..by one clock phase after its predecessor.,
,
dun ends a burst when act receives a call to,
..this defn. waits one clock phase before,
..entering idl for spacing.
```

```
988 list
709 i2c timing reclaim 10709 node 0 org,
host*pu D 3 15555 or ;*-pu pu 800 or ;
target,
,
edge mn-mn 00 dup !b over or @ drop dup ! ;,
,
dun mn 02 edge
idl 03 pu lit !b r--- @b pu lit over -if,
..-pu lit dup then drop 800 over edge
act mn 0E edge r--- act ;,
,
11
reclaim
```

To manually exercise the sensors using arrayForth IDE, strip the application down by commenting **/p** phrases on load descriptors for any nodes that would interfere in a path from node 10400 to 10608 (intervening nodes may still be loaded but should be left at warm .) Then adjust path 1 to make that connection. In order to run the I^2C bus, nodes 10707, 10708, 10709, 10715 and 10717 must be loaded and activated.

The original factoring of the code in node 708 allowed for manual operations via the IDE using that node. Later factoring changes were necessary to support the non-orthogonal devices on the ST. For interactive manual exploration of the devices you will need to strip the application down and place code in an adjacent node, such as 608, which may be used interactively to communicate with the I^2C bus via node 708. Such code is not presently included in this example.

3.3 Sensor Configuration and Polling

The set of I^2C devices on the ST support a least common clock frequency of 400 KHz. Their addresses are as shown in this table as are their constraints on clock timing, their ability to stretch the clock, their bus loading and their current requirements among other things.

Parameter	Unit	Barometer T5400 (C953H)	3-axis Accelerom'r KXTJ9-1007	Magne-tometer MAG3110	3-axis Gyroscope IMU-3000	IR Thermopile TMP006	Humidity & Temp SHT21	BLE and 8051 CC2541	PV Cell	Photo Diode
I^2C Bus Address	hex	EE	1E	1C	D0	88	80	AA		
I^2C V_{DD}MIN	V	1.7	1.7	1.62	1.71	2.2	2.1	2		
V_{DD}MIN	V	1.7	1.71	1.95	2.1	2.2	2.1	2		
V_{DD}MAX	V	3.6	3.6	3.6	3.6	5.5	3.6	3.6		
$I_{DISABLED}$	µA		0.9					1		
I_{LOW} (low/sby/idle/sleep/ pwrmode2)	µA	0.15 - 0.3	10	2	5	1	0.15 - 0.4	270		
I_{RUN}MIN	µA		5 12Hz				200			
I_{RUN}TYP	µA	790 baro 500 temp	29 100Hz	900 80Hz	5900 -dmp	240	300	20200		
I_{RUN}MAX	µA		138		6100 dmp	325	330			
P_{HEATER} @ 3.3V	mW						5.5			
Power-up Time	ms	10 - 11 - x	10	1.7						
Wake-up Time	ms	x - 2 - 2.5	1 / ODR							
T_{MEAS}MIN	ms					250				
T_{MEAS}MAX	ms					4000				
Some Interrupt/EOC/Ready sig		Y	Y	Y	Y	Y		Y		
Pin Capacitance	pF				5	3				
Pin Leakage	nA				100					
Max Load Driven	pF	100			400		400			
SCL_{MIN}	KHz	0	?	0	0		0			
SCL_{MAX}	KHz	3400	3400	400	400		400	<400		
Clock Stretching?		no spec	apparently	yes	apparently	no spec	no	yes		

The costs of using these things vary widely. Cheapest high rate sensor is accelerometer, and very low frequency magnetometer data are also pretty cheap. Gyro is terribly expensive, so much so that running it continuously will exhaust the ST battery in about 10 hours. In this section we discuss the gross characteristics of each sensor and the modes of operation that may be selected by configuring block 707.

The data stream distributed from sensor polling is fixed format, regardless of the operating modes of the sensors, and consists of the following sequence of *raw* 16-bit values (the first 11 values are the only ones supported initially):

Posn	Datum	Remarks	Posn	Datum	Remarks
+0	X Accel	8-bit mode, aligned high in 16-bit word, full scale ±2g.	+11	Accel who	0800
+1	Y Accel		+12	Mag who	C400
+2	Z Accel		+13	X Accel Filt	0.1 Hz low-pass
+3	X Mag	16-bit value, units are 1/10 Tesla	+14	Y Accel Filt	0.1 Hz low-pass
+4	Y Mag		+15	Z Accel Filt	0.1 Hz low-pass
+5	Z Mag				
+6	X Gyro	16-bit value, full scale is ±250 degrees/second			
+7	Y Gyro				
+8	Z Gyro				
+9	Die Temp	°C * 128			
+10	Sensor volt				

The polling operation takes 12.5 ms of I^2C bus time for the first 12 values above.

The previously mentioned initialization code in node 707 is table-driven, commanding node 708 directly:

```
this node initializes all sensors after reset
..by storing register values.,
,
table starting at zero holds up to 25 2-word,
..entries, one for each 8/16-bit register...,
,
...+0 00 aaaa aaa0 iiii iiii busadr, index,
...+1 w0 1111 1111 2222 2222 wordflg, byte1,2,
...first section of tbl disables all sensors,
,
on boot, node 708 executes left port to catch
..these commands. after exhausting the table,
..we direct node 708 to down for on-line work.
```

```
992 list
707 sensor init reclaim 10707 node 0 org,
host*/t here 2 / -1 + ; target,
,
stby acc 1E1B , 0 , mag 1C10 , 0 ,,
gyro D03E , 4000 , therm 8802 , 20400 ,,
08 /t everything standby,
,
acc 1E1B , 0 4000 , 1E21 , 0 ,,
....1E1B , 8000 C000 ,,
srst acc 1E1D , 8000 ,,
0E /t enables for condition yellow,
,
mag 1C11 , A000 , 1C10 , C100 ,,
gyro D03E , 0 , therm 8802 , 27400 ,,
16 /t enables for alerted mode,
2F org make sure room!,
!rs 2F /t above lit for,
...@p !b @+ .. @p !af .. !b @+ dup @p .. @p
..
...!b !b .. -if @p !b .. w8 then,
...@p !b @p .. w8 .. stop ..,
..!b next .. @p !b .. -d-- .. warm ;,
40 reclaim exit
```

The three instances of **`/t`** following groups of table entries are used to select the mode of operation. Uncomment the first for power measurement with all sensors in standby; the second, for minimal situational awareness using the accelerometer only; the third, for high power mode with all sensors active. Only one of these instances may be un-commented at any time. The polling cycle is defined by tables in node 607, used to command 608:

```
this node controls the sensor polling sequence
..by issuing commands to node 608 for doing,
..burst reads and distributing raw data.,
,
table starting at zero holds up to 25 2-word,
..entries, one for each burst...,
,
...+0 aaaa aaa0 iiii iiii reg index, busadr,
...+1 0000 0000 nnnn nnnn n-1 for n word read,
,
on boot and after each poll cycle, we expect a
..word of garbage from node 608 and then run,
..one polling sequence.
```

```
996 list
607 poll sequencer reclaim 10607 node 0 org,
host*/t here 2 / -1 + ;*/- - ; target,
,
table of up to 24 bursts,
acc 1E06 , 2 /- ,,
mag 1C01 , 2 ,,
gyro D01D , 2 ,,
therm 8801 , 0 , 8800 , 0 ,,
misc 1E0F , 0 , 8802 , 0 , 1C07 , 0 ,,
,
0E /t 30 org,
!rs 30 /t above @b dup or a! lit for,
..@p !b @+ .. @p @r; ..,
..!b @p !b .. @p seq ..,
..@+ !b next @p !b .. stm !rs ;,
3C reclaim exit
```

Upon stimulus from node 609, node 608 operates under direction of 607, commanding node 708 to retrieve register values from sensors. This node then distributes data across the 6xx row where data are abstracted and passed downward into the processing array where needed, and for demo purposes are also passed to node 508 for transmission to polyFORTH in the host chip:

```
                                                994 list
608 performs normal data polling as commanded   608 poll, distrib reclaim 10608 node 0 org,
by 607. the primitive is burst read of 16-bit   @r; a.i 00 @p !b !b ; .. @p @regs ..
words. node 708 is commanded to do the burst,   1w+ -n 02 @p !b .. @w+ .. @p !b @b ; .. 2/ !p
read; the resulting data are passed to nodes,   1w. -n 06 @p !b .. @w. .. @p !b @b ; .. 2/ !p
609, for internal distribution, and 508, for,   dlv n 0A up a! dup .. ! right a! .. ! ;
prototype raw data logging.,                    @nw n 0F push begin zif 1w. dlv ;,
,                                               ...then 1w+ dlv end
@r; starts a burst on chip a register i         swb n 16 push FFFF dup dup or,
@nw bursts n+1 words with msb in first reg;     ..pop dup 2* 2* a!,
@nbs bursts with lsb in first reg; on sensor,   ..9 for +* unext drop drop a and dlv ;
...tag, only accelerometer works this way.      @nbs n 1F push begin zif 1w. swb ;,
seq finishes a burst after @r; n neg is ones,   ..then 1w+ swb end
...complement of count for @nbs.,               seq n a.i 26 @r; -if - @nbs ; then @nw ;,
,                                               ,
stm waits for next cycle, passing stimulus,     stm 29 right a! @ left a! ! --l- ;,
...from 609 to 607 and slaving to 607           2E reclaim exit
```

As the data pass to the right across row 600, they may be "peeled off" and distributed downward into processing nodes.

3.3.1 Accelerometer Configuration

Provides X Y Z 16-bit signed values aligned high. LSB first, pointer does increment. Resets to standby. Output data rates 100 Hz (29 µA), 50 (16 µA), 25 (9 µA), 12.5 (5 µA) so this is fortunately a relatively cheap device to use. The high 8, or high 12 bits of the sample are valid depending on full power mode.

To disable the chip we store 0 into ctl reg 1B. To initialize for low power, we store 0 to 1B, 0 to 1B (2g full scale), 0 to data control reg 21 (12.5 Hz), 80 to 1B (enable). The 40 bit in 1B gives 12-bit data if 1.

We intend to enable the accelerometer at all times so it is by default enabled at 8-bit resolution, 12.5 Hz data, for 5 µA, with full scale (15-bit fraction) set to ±2g.

3.3.2 Magnetometer Configuration

Provides X Y Z 16-bit values. MSB first, pointer does increment. Has X Y Z offset registers. Also has die temp sensor. Minimum oversample is 16x; output data rates of 10 Hz (137.5 µA), 5, 2.5, 1.25 (17.2 µA) so it's linear energy per sample. On reset it's in standby. Might consider using trigger-immediate from standby at a rate higher than our poll to avoid wasting energy generating data even marginally faster than we are taking it.

To disable store 0 in CTRL_REG1 (10). For 1.25 Hz operation (17.2 µA), write A0 to CTRL_REG2 (11) and C1 to 10. This device does stretch the clock, forcing us to support that and also forcing use of energy-wasting pull-up.

We are running it at 1.25 Hz, and the raw values returned are taken as integers in units of 0.1 Tesla.

3.3.3 Gyro Configuration

Provides X Y Z as rates, 16-bit values as 15-bit fractions; has offset registers. MSB first, pointer does increment. Costs >6 mA while running!

We configure it for ±256 degrees/second full scale.

3.3.4 Thermopile Configuration

Provides 14-bit signed die temp and 16-bit signed sensor voltage to be combined. MSB first but no pointer increment. Transfers must be done in 16-bit units. When running continuously it can provide max 4 values/sec or can average up

to 16, no change in power. Cost is >240 µA while running. Default/reset mode: Up and running, sensor & ambient continuous conversion; 1 conversion/sec of four averaged samples.

Writing the configuration register requires a two-byte transaction; single byte does nothing.

3.3.5 Humidity Configuration

Provides Temp (14..11 bits, 85..11 ms) or RH (12..8 bits, 29..4 ms) after triggering command, using typ 300µA while making the measurement. 16-bit raw values, MSB first. The poll sequence is irregular as is the read protocol. The cost of the additional complexity in using this device will not be paid until there is a good reason for desiring the data.

3.3.6 Barometer / Pressure Sensor Configuration

Measurement is performed on request, one of four modes for control of time and power consumption. Has end of conversion pin. Thus the poll sequence is irregular.

Need to get 8 16-bit calibration parameters from device so we can convert raw data to engineering units. All data LSB first. Temperature calculation uses two parameters, one constant, three multiplies and three double-shift series. Pressure uses six parameters, five multiplies and eight double shift series. This will take more than one node. Note that the temp is in C*100 so is <16 bits, pressure in Pa where 100,000 is about one atmosphere / 1 bar; 16 bits won't cut it but 18 will, however mbar*10 does fit in 16 bits. Raw temp is used in pressure calculation in six places, raw pressure in one.

So this sensor needs initial pump of 8 parameters from I^2C to RAM, then each two words read from device get crunched to yield two words in engineering units. While this can certainly be implemented, we will need a case for needing the measures from this sensor before adding that much complexity.

3.4 Sensor Data Processing

The present code includes a simple example of signal processing, with source in blocks 1062 and 1064.. Node 610 distributes the accelerometer values downward to node 510, in which the data are processed by three first order low-pass filters whose 3dB points are set at 0.1 Hz; the filtered triad of values is then passed through node 509 where they are merged into the data stream for the host chip via 508, and are also wired downward via node 410 to node 010 where they are staged for delivery via BLE. The code for node 610 is trivial:

```
plumbing to demo sensor processing,
,
610 taps accelerometer data into 510.,
510 filters accel and feeds to 509, 410.,
509 buffers 3-vector from 510 to 508.,
508 appends filtered accel to host msg.,
410, 310, 210, 110 wire filtered accel to,
...node 010 for ble. they also serve as buffer
...for 3-vector.,
010 passes 9 words to 009; the first three are
...filtered accel, remaining six count up.
```

```
1064 list
plumbing reclaim,
,
10610 node 0 org
run 00 @ !b 2 for,
..@b dup ! a push up a! ! pop a! next,
..C 9 for @b ! unext run ; 0C,
,
10509 node 0 org
run 00 @ @ @ push over !b !b pop !b run ; 04,
10508 node 0 org
run 00 C for @ !b unext a push right a!,
..2 for @ dup !b unext pop a! run ; 0A,
,
10410 node 0 org*wire @ !b wire ; 01,
10310 node 0 org*wire @ !b wire ; 01,
10210 node 0 org*wire @ !b wire ; 01,
10110 node 0 org*wire @ !b wire ; 01,
10010 node 0 org*hose right a! @ !b,
..down a! 2 for @ !b unext 5 for,
....pop dup push - !b next hose ; 0C,
0C reclaim exit
```

Node 510 shows implementation of three simple IIR filters using the stack to maintain the three integrators. For best phase characteristics order of operations is important: First add new sample into integrator, second calculate feedback as integrator * **a**, third subtract feedback from integrator. Low-pass output is the feedback value; high-pass output is

new sample minus feedback, and the sum of the two is always identical with the input signal. The coefficient **a** is **(2πf)/s** where **s** is sampling frequency and **f** is frequency of 3dB point. The maximum integrator value will be **(1/a)+1** which in this case requires 4 bits of headroom; to keep the integrator single precision this means we must scale the 16-bit samples down 2 bits for filtering. Since the data have only 8 or 12 significant bits, an 18-bit integrator is more than enough.

The stack may safely be used as a register file so long as no more than two values are pushed onto the stack in between values that are committed to the circular stack array. In this way S and T behave as scratch registers and the circular stack array behaves as a nonvolatile register file. The code, which uses the 17-bit fractional multiply routine in ROM, is as follows:

```
                                                    1062 list
this node demonstrates first order iir lowpass      510 filter accel reclaim 10510 node 0 org
filters applied to accelerometer. in addition       .1hz si-so 00 . + 8235 *.17 dup push,
it demonstrates use of the stack as a register      ..- 1 . + . + pop commit? dup drop ;
file to hold three variables.,                      int s-s' 07 up a! @ 2* 2* 2/ 2/ 2/ 2/,
,                                                   ...1hz 2* 2* dup !b left a! ! ;,
.1hz applies a first order low pass with 3db,       ,
point at 0.1hz to incoming signal i producing       run 10 x int drop y int drop z int drop,
result o. integrator s is maintained.,              ..drop drop drop drop drop run ;,
for sampling frequency of 10 hz max integrator      ,
value is 15.91 times peak signal.,                  16 reclaim exit
,
dup drop at the end of the definition needed,
if subsequent code doesn't push integrator,
from s into stack array. subsequent code may,
only have two items on top of the integrator,
value to avoid clobbering in stack array.,
,
int given an integrator reads corresp,
new sample and distributes result to,
nodes 509 and 410. drop steps to next,
integrator. five drops at end aligns,
for the first of three integrators.
```

Plumbing for nodes 410, 409, 408 and the wires of 310, 210, 110 are all defined in block 1064 as listed above. The boot descriptors which add this example to the basic framework are in block 968; code overloading is used for those nodes that were already programmed.

3.5 Reprogramming the BLE Radio Chip

In order to communicate over BLE under the effective control of the GA144, it was necessary to reprogram the CC2541 to behave as an I^2C slave. For the present this is done with existing tools and a C program written for us by Punch Through Design LLC. This section describes the 8051 program, procedures for generating the 8051 firmware image from source, and for loading that image into the ST. Note that generation from source requires use of software that must be licensed from a third party.

If you wish to reproduce our results, please request the firmware image from GreenArrays customer support.

3.5.1 Specification of the Program

The CC/GA packet protocol is a simple straw man intended for demos and early concept testing of BLE communication from a GA144 controlled SensorTag to a specifically configured PC host system. The data direction is almost entirely uplink from the GA to the PC. Downlink extensions will be added only as the need arises.

The CC2541 and PC software will share an id code they will use to establish a connection. If the need arises to run two sets of PC/Tag links in the same locale then separately prepared, matching sets of code will be loaded into each PC and associated Tag. It is not required for the CC to inform the GA when a connection is lost. When the connection is re-established then data traffic will resume.

Whenever a connection is established and the CC2541 has a complete data packet from the GA144 it will make its best effort to transmit the packet to the PC. When the packet buffer is empty the 2541 will notify the 144 by asserting the Transmit Buffer Empty line connected from the CC to the GA. *As it was actually implemented, data only move from the 2541 to the PC at the PC's initiative by requesting the value of attribute handle x25. The 2541 indicates buffer ready regardless of whether or not the PC has read it. Consequently the 144 is continually overwriting the buffer and has no way of knowing if the data have been moved to the PC. Further, the code actually written does not de-assert the buffer ready line until after the I^2C message has been delivered; consequently there is a possible race after all.* Transmission to the 2541 consists of an outbound address octet xAA, followed by 20 octets of data.

At such time as the need arises to move data or status from the PC or CC to the GA then a Receive Data Available will be asserted from the CC to the GA. When the CC2541 receives the first (command) byte of a new transaction and has had time to digest its meaning, the appropriate handshake line should be deasserted immediately so that there is no race condition over its interpretation at the end of the transaction. *As actually implemented, this code does not de-assert the data available line until after the I^2C message has been delivered; consequently there is a possible race after all.* Inbound I^2C reading consists of an inbound address octet xAB, followed by 20 inbound octets of data.

The data are moved using fast mode (400 KHz max) I^2C, and it turns out that the CC2541 implementation *does* stretch our clock. The function byte initially specified in these messages was not actually implemented.

3.5.2 Compiling the Source

The contractor used IAR compiler version 8.11.2; we downloaded and installed that same version since it was important to us to prove that we could reproduce an identical firmware image from source.

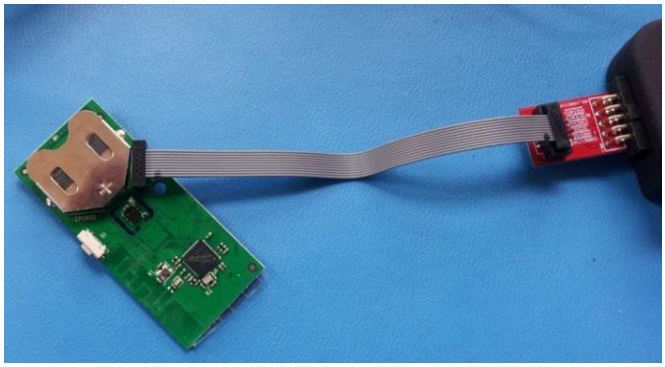
The source code was left for us on http://www.github.com by the contractor. We made accounts there and verified email. The contractor added our accounts to a working group. This allows us to see the repository. Clicked on "Clone in Windows" which took us to an opportunity to download Github for Windows. Went ahead to install it; this includes installation of .NET framework 4. Started that program and finally asked it to clone the PunchThrough/GreenArray repository. This put a directory GitHub inside User Documents and does seem to contain a full copy of the entire project repository. Made a copy of that entire thing as "experiment" and double clicked the project root file known as C:\Users\greg\Documents\Experiment\GreenArray\BLE-CC254x-1.3\Projects\ble\SensorTagGA\CC2541DB\SensorTag.eww. Got IAR compiler environment. Right clicked Sensortag root and said make. It did so. This produced a new directory ...CC2541DB\CC2541DK-Sensor with three subdirectories of compiler output, including the .d51 and .hex images. Contractor sent his version of that directory for comparison, which matched exactly.

Further recommendations on compiling provided by the contractor:

1) launch IAR
2) File..Open..Workspace
3) Navigate file selection dialogue to Projects/ble/SensorTagGA/CC2541DB/SensorTag.eww
4) In "workspace" pane (usually found on left side) there is a drop down selection for the "configuration" You should use CC2541DK-Sensor. The other two options (CC2541DK-Sensor-OAD-ImgA and CC2541DK-Sensor-OAD-ImgB) are used for over the air firmware update.
5) To do compile only click on tool bar icon that looks like 3 rectangles (yellow green red) with an array that points to "01 0101"
6) To compile and download for debugging use the green arrow toolbar icon...debugger will stop and wait at the first line of main...hit F5 to run
1. 7) Some useful IAR short cuts....ctrl-shift-F will search across the entire project
7) Another useful IAR tool...right click on a term in the source code and select "Go to definition of"
8) Output firmware images can be found in Projects/ble/SensorTagGA/CC2541DB/exe
9) all of GA source code customization should exist in two places
    - Components/hal/target/CC2541GA/...
    - Projects/ble/SensorTagGA/Source/...

3.5.3 Loading Code into the CC2541

To load a binary image into the 2541, you will need a TI "CC Debugger". You may need to shop around TI for a way to buy one that is not back-ordered. We were successful in obtaining them as part of CC2541DK-MINI, Revision 1.0. This kit included the CC Debugger (a black plastic box), along with a USB cable and two of the adaptors provided in the kit, as shown assembled at right. The remaining items in the kit are not needed at this time. The "business end" of the CC Debugger is a female 10-pin miniature header on the flat cable.



The photo at left shows a CC Debugger connected properly to a Sensor Tag. The black dot at each end of the flat cable represents the pin 1 side of the cable and its connectors.

Proceed by installing and running the TI tools as described in the following sections.

3.5.3.1 Installing TI Tools

Before plugging any device into the PC, install the full suite of relevant TI tools; this will ensure that the relevant USB drivers are also present. The tools we load are as follow:

3.5.3.1.1 SmartRF Studio 7

Download and unzip `swrc176r.zip` or later. Run the executable. Take all the default options, give all the permissions. We did not actually need to use this tool but recommend its installation anyway.

3.5.3.1.2 SmartRF Flash Programmer

Download and unzip `swrc044r.zip` or later. Run the executable. Take all the default options, give all the permissions.

3.5.3.1.3 SmartRF Packet Sniffer

Download and unzip `swrc045w.zip` or later. Run the executable. Take all the default options, give all the permissions.

Plug preprogrammed CC2540DK dongle from CC2540EMK-USB Rev 1.0.1 into a USB socket. Driver installation procedure runs and says CC2540 USB dongle is ready to use. Device manager puts it under CEBAL controlled devices.

Run packet sniffer. Select protocol & chip type: Bluetooth Low Energy. Hit start button. Sees no traffic. Push side button on SensorTag and with the default software we get a mess of packets! Specifically, about 300 of them ... 30 seconds' worth at 10 Hz.

3.5.3.1.4 TI BLE Stack and Btool

Create a directory and unzip the BLE stack (BLE-CC254x-1_3.zip) there. Run the executable. Install the dongle from CC2541DK-MINI rev 1.0; does not find driver. Browse for drivers in C/Texas Instruments/BLE-CC254x-1.3/Accessories/Drivers. Voila, now shows up in Ports as TI CC2540 Low-Power RF to USB CDC Serial Port (COM11 on one of our systems).

Without using Btool, we will have to talk the full HCI protocol ourselves in the PC. Meanwhile, Btool may be used to read buffers from and write them to the CC2541.

3.5.3.2 Installing New CC2541 Firmware

Having done all of the above, follow these steps precisely:

1. Start with powered-down ST (no battery) and CC Debugger (CCD), assembled but not plugged into USB.
2. Connect flat cable between ST and CCD.
3. Insert battery in ST
4. Plug CCD into USB port. Red LED on; installs driver software, says successful, see CCD as a CEBAL device.
5. Fire up flash programmer. May say a connected board running old FW, if so kill dialog.
6. What do you want to program; on list expect to see EB8077/CC2541/CCDebugger/05CC/0025(old). If not, repeat steps 1-6.
7. Program primary with new code .hex file
8. When it says successful, remove CCD USB connection, remove battery from ST, unplug the two.

3.6 Communicating with BLE Chip

We talk to the 8051 program described above using a fast-mode I^2C bus and four handshake lines, for the purpose of communicating with a PC.

Incidentally, there appears to be a bug in the 8051 code. After hammering it continually with outbound data at 2 Hz, the 8051 code will eventually hang (this has taken as little as a few seconds and as much as 12 hours). I_{BAT} will be on the order of 6.4 mA more than our normal background level, implying that the CC2541 is not in low power standby mode. The I^2C lines between us are both high so it is not a matter of clock stretching. Both handshake lines are low, and as a result the 144 is not sending data via I^2C any longer. However the 144 is still running normally, polling sensors, sending data to the host chip, and so on. Further, the BLE radio is still advertising, and if we ask Btool to make a connection to the 2541 it is able to do so, and we are able to read attribute 25 from Btool. However when this is the first connection made after the chip was reset, the value of the attribute is all zero even though many nonzero sets of data had been written from the 144. Since we do not plan on making operational use of the CC2541 with this code, we do not plan to go after this 8051 bug.

3.6.1 Top level control

In addition to controlling the timing of sensor polling and analysis, node 617 generates commands that follow a path down the right edge of the chip and across the bottom, ending with node 008 which drives an I^2C bus talking to the BLE chip. These commands are synchronized in time with sensor polling and indicate what should be done with the data from the preceding poll cycle. The command 1 means send data to the BLE chip; 0 means to discard the previous data. These commands immediately pass through four nodes that deal with handshake lines, and along the way the command is modified as necessary to accommodate the present status of the BLE chip.

3.6.2 Handshake Lines

Four status lines connect nodes 517, 417, 317 and 217 to the CC2541 BLE radio chip.

On boot, node 517 raises RESET- signal to the 2541 for 100 µs and then produces a 100 µs RESET- pulse. This is more than sufficient (the spec is 1 µs) for a complete, clean reset of the 2541. We have observed that the handshake lines from the 2541 are driven approximately 115 ms after start of reset, implying that this is about the boot time needed by that chip. Since node 617 is giving us a delay of at least 5 seconds before starting operations, it seems that the BLE chip should be completely ready by the time we try talking to it.

Node 417 monitors an incoming data available signal from the CC2541. This signal goes high when outbound data for attribute handle 27 has been received from the PC, and does not go back low until *after* we have read the data. The CC2541 will not go back to its low power idle mode until we have read this 20-octet buffer. The command being passed from node 517 to 317 has its bit 1 turned on if incoming data are available.

Node 317 monitors an outgoing data buffer ready line. The buffer in question is internal to the 8051 and it is available after the 8051 has finished what appear to be strictly internal operations. The high state of this line does not imply that the data have been moved to the PC, merely that we may if we wish replace the data for attribute handle 25 by overwriting it via I^2C if we wish to. The line does not go low until *after* we have written the data. Attribute 25 is moved to the PC only when requested by the PC. The command being passed from node 417 to 217 has its bit 0 turned off if the outgoing buffer is unavailable.

Node 217 controls a WAKE- signal to the 8051, and the signal is asserted for >100 ns when we pass a command from 317 to 117 that has its bit 0 (write) turned on. This wakens the 8051 so that we may overwrite attribute 25, the outbound data buffer, over the I^2C bus. The 8051 will remain active for 15 ms after WAKE- which is undoubtedly far longer than necessary.

3.6.3 I^2C Bus

Node 008 has the mechanism for manipulating the I^2C bus lines at about 375 kbps, while node 009 decides whether to write or discard data, and whether or not to read data, based on each command received.

If command bit 0 is 0, outbound data are received and discarded. When the bit is 1, received data are sent to the BLE chip over the I^2C bus after waiting for the 8051 to be ready for the transaction. The 8051 is supposedly ready to receive I^2C data after 430 µs, but that is not always true, probably due to other multiprogrammed activity of that chip. We have found that waiting about 530 µs gives better results. The outgoing I^2C frame consists of start bit, the outbound device address xAA, 20 octets of data, and a stop bit.

If command bit 1 is then found to be 1, we must read the incoming data so that the BLE chip may go idle. This is done using a frame consisting of a start bit, inbound device address xAB, receipt of 20 octets of data with NAK on the last octet, and a stop bit.

Because this bus has low valued pull-up resistors (2.2k) the current drawn is on the order of 1 mA when each line is low. As a result it would not be beneficial to drive them slowly using the watch crystal as a time base. Instead, we run as fast as we have been told the 8051 will work, using **`unext`** loop timing.

3.6.4 Message Format

The outbound message consists of 20 octets, formatted as follows:

Posn	Datum	Units	Remarks
+0	Sequence	0.5 second	Time since boot
+1	X Accel	14-bit fraction in g (full scale ±2g)	0.1 Hz first order low-pass filter applied to 8-bit noisy data.
+2	Y Accel		
+3	Z Accel		
+4	-6		Recognizable nonzero data.
+5	-5		
+6	-4		
+7	-3		
+8	-2		
+9	-1		

For example, here is a message actually received by Btool:

```
01 0D 43 68 03 68 FE 74 FF FA FF FB FF FC FF FD FF FE FF FF
```

Inbound messages consist of 20 octets. No inbound message format has been defined.

3.6.5 High-Level Protocol

At the top level, node 617 provides master system timing. It starts a polling cycle every 100 ms. At the same time it transmits a command to the BLE communication mechanism. All commands are to discard data, except that every fifth command is a transmit request. Thus we move data into the BLE chip at 2 Hz.

3.7 PC BLE Communications

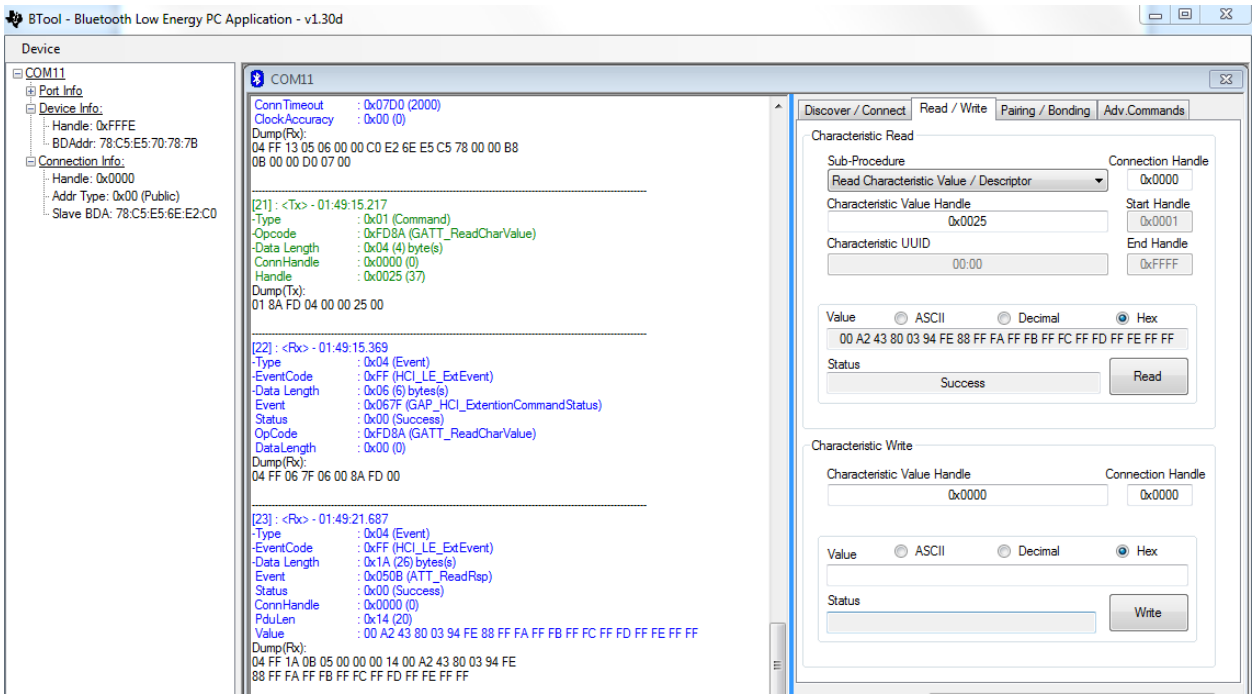
At present we use Btool to exchange data with the GA144 over Bluetooth Low Energy.

When Btool is started, use its default COM port with 115200 baud. In the Discover/Connect tab click scan and wait for a non-greyed Slave BDA (MAC address) in the Link Control section. Then click Establish and wait for the ST to appear as "Connection Info" in the tree view on the left side of Btool window.

When establishing the link with the ST using Btool, set max connection interval to the desired frequency of poll/response exchanges to be made with the ST. This will determine the latency for reading or writing; for example, when reading one exchange is used to request the value from the ST and the next exchange retrieves the value. The "max interval" sets Btool polling rate; when set to 100 ms, this means a 100 to 200 ms latency on reads.

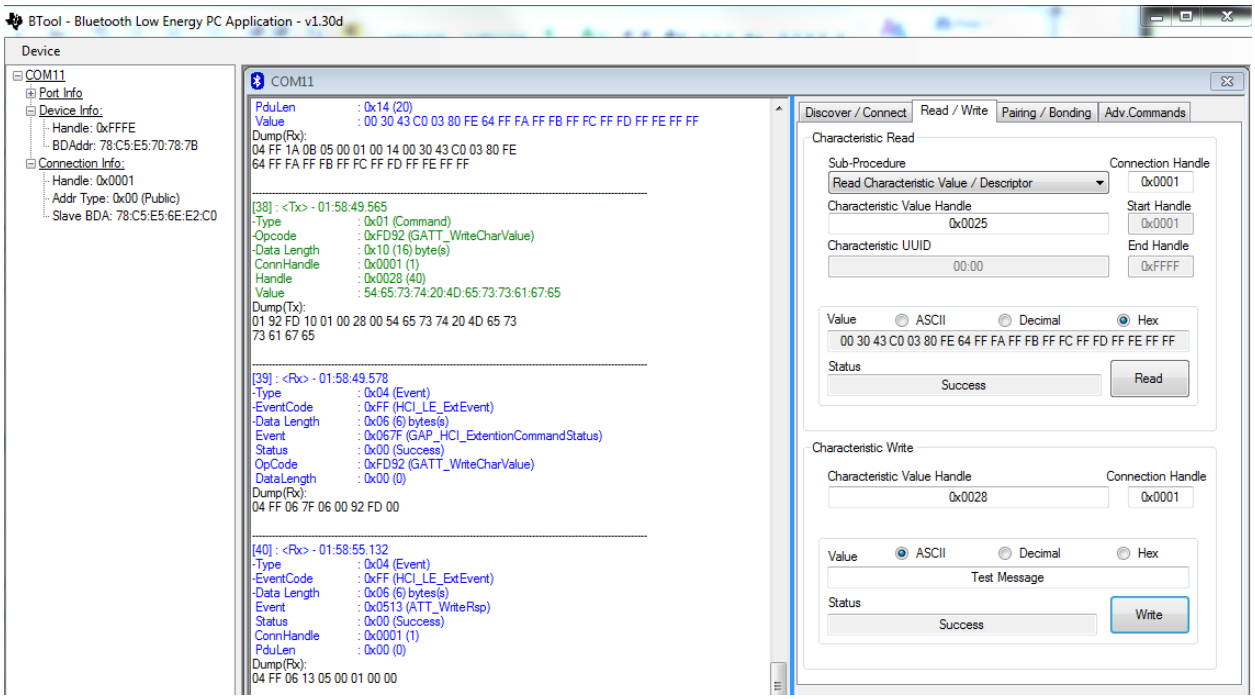
Data are moved on the "Read/Write" tab after a connection is made. To read the 20-byte buffer that the GA144 continually overwrites, read characteristic value handle x0025. To write 20 bytes to the GA144, write characteristic value handle x0028. We are also told "to enable notifications for I^2C writes, write hex 01:00 to handle x0026." We do not know what this actually means at this time.

The following screen-shots illustrate reception of ST data and transmission from the PC using Btool:



The value 00A2at the start of the message is sequence number; in this example, data were being delivered into the CC2541 buffer at 2 Hz and the sequence number is in units of ½ second. The next three values, 43 80 03 94 FE 88 are filtered X Y and Z acceleration in raw units. The rest of the buffer, FFFA FFFB FFFC FFFD FFFE and FFFF, are the static filler values supplied by node 009.

The screen-shot below shows transmission of data to the CC2541 for delivery to the GA144:



3.8 PC Communications Via Host Chip

Another method for delivering sensor data to the PC is to make use of the diagnostic "wiring" to the port bridge connecting nodes 400 of the Host and Target chips. The Host chip, running the polyFORTH Virtual Machine, is programmed to read sample data sets from the Target chip where they may be stored to SPI flash on the Host, displayed on the PC Terminal Emulator, or stored onto the PC disk using polyFORTH's serial access to the PC's mass storage. The diagram below shows the Port Bridge and special program in node 400 being dynamically accessed via Snorkel and Ganglion (the dotted line) to move data from the Target chip into external RAM:

GA144 Host comm for Target / SensorTag

The Port Bridge, developed for the Automated Testing System, is built in the IDE using the following procedure:

```
setup resets target chip and sets up the port
bridge for ide and creeper use. br

first we load sync boot master in our 300 done
by !sync
second !his loads bridge in uut node 300 using
frame which sends a boot frame starting at loc
ation a compiled for bin nd's ram
third !ours loads bridge into our node 300 br

setup does all this and leaves ide set to node
400 which may talk to 1400 through its up port
for testing of the bridged 2-chip system.
```

```
222 list
- build port bridge
set n nn 0 swap hook 2 + 10000 * io r! ;
rst n 500 set ;,
,
!sync 0 300 hook 0 64 1901 boot 19 call ;,
,
pt@ a-n @ 15555 or ;
frame a nd nn-n 2* 32768 + block + dup 2 + pt@
3 + for dup pt@ lit' 6 call 1 + next drop ;
!his 5 1904 1905 frame ;,
,
!ours off 19 call 0 64 1904 boot ent 5 call ;
,
setup 0 rst !sync 1 rst,
0 300 hook !his !ours 0 -hook,
2 -hook 1 -hook 0 -hook ;
```

A similar procedure is used when booting from flash.

The port bridge code is fully symmetrical, with identical code in node 300 of both chips:

```
this code is loaded into nodes 300 of two,
chips to make their up ports into a slow but,
transparent bridge between the chips.,
,
each node waits for port data or rising clock
edge and based on clock state after wakeup,
moves a word between sync serial and port.,
,
on the line, xmtr controls clock and data are
sampled on falling edge. clock and data set to
weak-low one half bit time from end of word.,
,
there is no flow control and read handshake,
line of the up port is not meaningful.,
,
origin moved to 5 so boot frame header can be
laid down without affecting slot 2 jumps.
```

```
638 list
1904 ats sync bridge cr
reclaim 300 node 3 8 org cr
host -cr  hd- -3 -8 + ; target br

dly 08 b !b 40 for unext ; 88ns
1bt 0B wb-w' dup dly 10000 or dly ;
zro 0E 10001 dly ; -cr  wpd 10 10001 !b ;
18o 12 w zro 17 for begin cr
15 -if 30003 1bt 2* *next drop wpd ; cr
1A then 30002 1bt 2* next drop wpd ; br

18i 1E x drop dup or !b 17 for cr
21 .. begin @b -until cr
22 .. begin @b - -until cr
23 .. - 2 and 2/ a 2* or a! next cr
27 a up a! ! br

idl 2A 165 --lu a! . @ @b -if drop 18i ; cr
2E then zro drop 18o idl ; cr
ent 31 io b! wpd begin @b - -until idl ; cr
36 here here hd- 0 5 org
frame ent dly , cr
org 36 1904 bin
```

The special program in node 400 of the Host chip waits for either an incoming ganglion packet on its right port or for an incoming data packet from the Target chip. When a ganglion packet is received, we discard the two words it gives us and wait for a Target packet, delivering its data in the ganglion reply. When a Target packet is received in between ganglion packets, the Target packet is received and discarded. This prevents loss of synchronization when the data are not being accessed from polyFORTH.

```
1002 list
508-400 demo,
10508 node 0 org*wire @ !b wire ; 01,
10507 node 0 org*wire @ !b wire ; 01,
10506 node 0 org*wire @ !b wire ; 01,
10505 node 0 org*wire @ !b wire ; 01,
10504 node 0 org*wire @ !b wire ; 01,
10503 node 0 org*wire @ !b wire ; 01,
10502 node 0 org*wire @ !b wire ; 01,
10501 node 0 org*wire @ !b wire ; 01,
10500 node 0 org*wire @ !b wire ; 01,
10400 node 0 org*wire @ !b wire ; 01,
,
400 node 0 org
run 00 r--- io b! begin
poll 02 @b 2* 2* -if right b! snork @b @b
pump 07 /s' lit for @ !b unext run,
..then 800 and if until
pass 0E /s' lit for @ unext then poll ; 12
```

The simple polyFORTH program for displaying data from the Target chip is as follows:

```
2682
 0 The Target chip offers us a packet of data every 100 ms, talking
 1    to our node 400 over the bridge.  Node 400 expects us to hand
 2    it a word of garbage through right port, whereupon it passes
 3    us  /s+1  16-bit values.  Node 400 discards packets received
 4    before we have started such a transaction.
 5
 6 See aF block    998 for the current value of  /s .
 7
 8
 9
10
11
12
13
14
15
```

```
282
 0 ( SensorTag Data)   EMPTY
 1    HEX 0C 3 + CONSTANT /s  DECIMAL
 2
 3 ( Display)  1 FH LOAD
 4 ( Data)  2 FH LOAD
 5 ( Target)  3 FH LOAD
 6
 7 ( Prime) STFET
 8
 9
10
11
12
13
14
15
```

```
2683
 0 *.  is a 14-bit fractional multiply.
 1 (D..)  formats a signed double number with n decimal places.
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
283
 0    ( Fractions)
 1 : *. ( f f - f)   M* D2* D2* SWAP DROP ;
 2
 3 : (D..) ( d n)   1- >R  SWAP OVER DABS  <#  BEGIN # NEXT
 4       46 HOLD  #S SIGN #> ;
 5 : D.. ( d n)   (D..) TYPE SPACE ;
 6 : D..R ( d n w)   >R (D..)  R> OVER - SPACES TYPE ;
 7 : N.. ( n n)   >R DUP 0<  R> D.. ;
 8 : N..R ( n n w)   >R >R  DUP 0<  R> R> D..R ;
 9
10 : .F ( f)   10000 *. 4 N.. ;
11
12
13
14
15
```

```
2684
 0 STBUF  second level buffer for raw target packet.
 1
 2 The formatting words display received values in engineering
 3    units.
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
284
 0    ( Data array)
 1 /s 1+ TABLE STBUF   STBUF  3 OFS ACC   3 OFS MAG   3 OFS GYR
 2    2 OFS TMP  2 OFS MISC   3 OFS FIL   DROP
 3
 4 : .AC ( i)   ACC + @ 100 *. 2 5 N..R ;  ( g)
 5 : .MG ( i)   MAG + @ 1 6 N..R ;  ( Teslas)
 6 : .GY ( i)   GYR + @ 12500 *. 2 7 N..R ;  ( Deg/sec)
 7 : .TM   TMP @ 10 128 */ 1 4 N..R  SPACE 1 TMP + @ 0 .R ;
 8 : .DAT   0 .AC 1 .AC 2 .AC  ."  /"
 9    0 .MG 1 .MG 2 .MG  ."  /"
10    0 .GY 1 .GY 2 .GY  ."  /"
11    .TM  ." /" ;
12
13 : .FL ( i)   FIL + @ 100 *. 2 5 N..R ;  ( g)
14 : .FILT   0 .FL 1 .FL 2 .FL  ."  / 0.1Hz lowpass" ;
15
```

```
2685
 0 STFET  starts a snorkel program to request next sample packet
 1    from node 400.  STDONE waits till the operation is complete.
 2
 3 Z  (the 3-line version) waits for a packet, displaying the time
 4    spent waiting.  The packet is moved to STBUF, STFET is initi-
 5    ated for the next packet, and the data received are displayed
 6
 7
 8
 9
10
11
12
13
14
15
```

```
285
 0    ( Data from Target)   HEX
 1 CREATE SThld  /s 1+ ALLOT
 2 HERE  :DOWN 2,  1 , 2033 ,  1 uu 6 ll 0 ,path  ( rsp) /s W,
 3    ( pay) 0 W,  0 W,   ( lng) HERE OVER - 2/ 1-
 4 CREATE ?ST   :DOWN 2,   o18 ,  ( lng) W,  ( ?bf) W,
 5    i16 ,  /s W,  SThld W,   HERE  FIN ,   CONSTANT STfin
 6 : STFET   ?ST STfin +SNORK DROP ;
 7 : STDONE   STfin sDONE ;   : ZZ  HEX SThld 10 DUMP ;   DECIMAL
 8
 9 : Z   COUNTER STDONE SPACE TIMER
10    SThld STBUF /s 1+ MOVE  STFET CR .DAT ;
11 : Z   COUNTER STDONE  0 0 TAB TIMER
12    SThld STBUF /s 1+ MOVE  STFET 1 0 TAB .DAT 10 SPACES
13    2 0 TAB .FILT 10 SPACES ;
14 : SEC ( n)   PAGE 10 * 1- FOR  Z NEXT ;
15 : HR ( n)   2* 0 DO  1800 SEC LOOP ;
```

The above code yields a display on the PC continually updating at 10 Hz, as shown here:

```
63880
 1.06 0.04-0.02 /-142.9  96.3   4.5 /  -0.13  -0.55  -0.68 /22.7 -81/
 1.05 0.04-0.03 / 0.1Hz lowpass
```

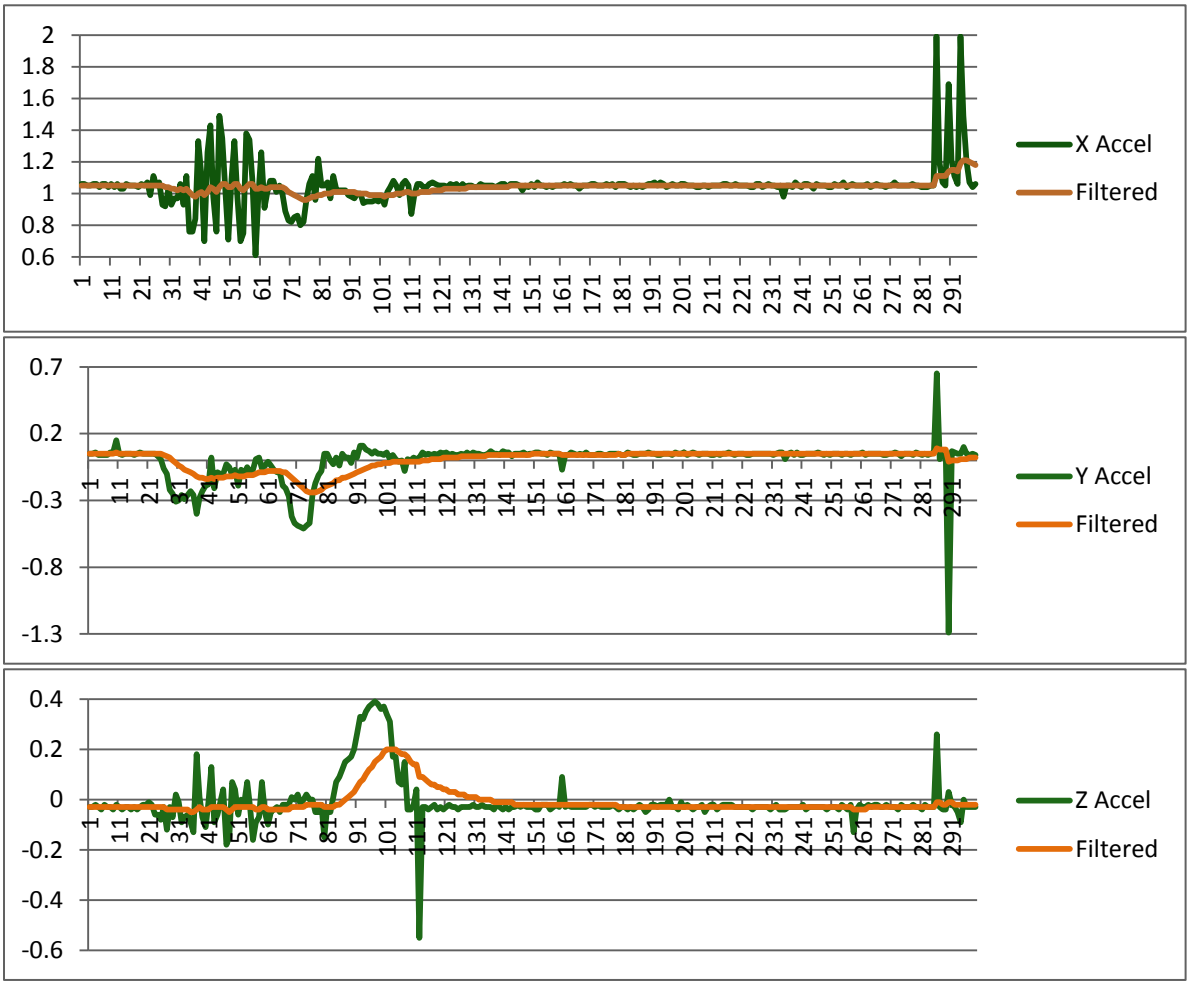
The top row shows X Y Z triads of acceleration (g), magnetometer (Teslas), Gyro (degrees/sec), and data from the thermopile sensor. The second row shows acceleration passed through the low-pass filters.

A slightly different program sends data in Comma Separated Value (CSV) format to the PC.

An example of one second's worth of the data appears below, displayed by **1 CSEC** . The plots of raw and filtered acceleration were produced by Excel from 30 seconds of such data, during which the evaluation board was manipulated on the bench. The event between 28 and 30 seconds resulted from pounding the bench with a fist.

```
286
 0    ( CSV)
 1 : N., ( n n)   >R DUP 0<  R> (D..) TYPE ." ," ;
 2 : .CAC ( i)   ACC + @ 100 *. 2 N., ;  ( g)
 3 : .CMG ( i)   MAG + @ 1 N., ;  ( Teslas)
 4 : .CGY ( i)   GYR + @ 12500 *. 2 N., ;  ( Deg/sec)
 5 : .CTM   TMP @ 10 128 */ 1 N.,  1 TMP + @ (.) TYPE ." ," ;
 6 : .CFL ( i)   FIL + @ 100 *. 2 N., ;  ( g)
 7 : .CSV   0 .CAC 1 .CAC 2 .CAC  0 .CMG 1 .CMG 2 .CMG
 8    0 .CGY 1 .CGY 2 .CGY  .CTM  0 .CFL 1 .CFL 2 .CFL ;
 9 : CSV   CR ( COUNTER) STDONE  ( TIMER ." , ")
10    SThld STBUF /s 1+ MOVE  STFET  .CSV ;
11 : CSEC ( n)   PAGE 10 * 1- FOR  CSV NEXT ;
12
13 : 1LOG ( n)   STDONE SThld STBUF /s 1+ MOVE  STFET  16 512 */MOD
14       60 MOD  300 + BLOCK +  STBUF SWAP 16 MOVE UPDATE ;
15 : LSEC ( n)   0 SWAP 10 * 1- FOR  DUP 1LOG 1+ NEXT . FLUSH ;
```

```
1.06,0.05,-0.04,-143.3,96.1,4.9,-0.23,-0.28,-0.54,23.0,-78,1.05,0.05,-0.03,
1.05,0.04,-0.04,-143.5,96.0,4.7,-0.07,-0.41,-0.58,23.0,-76,1.05,0.05,-0.03,
1.05,0.04,-0.03,-142.8,95.7,4.6,-0.13,-0.47,-0.57,23.0,-76,1.05,0.05,-0.03,
1.04,0.05,-0.03,-143.0,95.9,4.1,-0.20,-0.46,-0.38,23.0,-76,1.05,0.05,-0.03,
1.05,0.05,-0.02,-143.2,95.5,7.3,-0.03,-0.52,-0.78,23.0,-76,1.05,0.05,-0.03,
1.05,0.06,-0.03,-143.1,96.1,6.5,-0.10,-0.35,-0.78,23.0,-76,1.05,0.05,-0.03,
1.06,0.07,-0.05,-143.3,95.9,6.2,0.00,-0.59,-0.55,23.0,-76,1.05,0.05,-0.03,
1.04,0.04,-0.04,-143.0,96.3,4.5,-0.10,-0.52,-0.68,23.0,-76,1.05,0.05,-0.03,
1.05,0.05,-0.04,-142.9,95.8,4.4,0.01,-0.56,-0.58,23.0,-74,1.05,0.05,-0.03,
1.06,0.04,-0.03,-142.9,96.1,6.8,0.02,-0.57,-0.62,23.0,-74,1.05,0.05,-0.03,
```



To log one minute's worth of raw binary data to PC disk, use **60 LSEC** from the above block.

4. Results

We have shown that the GA144 can be used to poll sensors and analyze data at a very low average power, thus using a minimal amount of energy per unit time.

We have also demonstrated that at low duty cycles it is safe to operate our chips well beyond the 1.98V V_{DD} limit recommended by the foundry. This limit was intended to ensure no more than 10% degradation due to transistor aging (hot carrier injection) after 10 years of operation at 100% duty cycle. When running at 1/1000 duty cycle, the devices age 1/1000 as rapidly which allows higher voltages to be used without appreciable degradation over time.

Finally, we have demonstrated the ability to gather data and transmit them to a PC by two methods: First, by simple and low power synchronous communication to another GA144 running polyFORTH and from there to a PC by USB serial connection (this could also be done using TCP/IP over ethernet); and second, by communicating with the TI CC2541 using custom firmware and an I^2C connection.

The following table shows measured (estimated where noted) power numbers for various things, which will then be discussed. Note that the rated capacity of a CR2032 battery is 225 mAh at 3V, or 2,430 joules; calculations below assume this entire capacity is usable although in practice that does not seem to be the case.

Item	2.1v V_{DD} Usage			3V Battery Usage				
	I_{DD} µA 100% duty	I_{DD} µA in app	P_{DD} µW in app	I_{BAT} µA 100% duty	I_{BAT} µA in app	P_{BAT} µW in app	Joules per hour	Hours battery life
GA144 Fully Idle Leakage		14	29.4		10.5	31.5	0.113	21,500
+ GA144 running 12.5 pF watch xtal		27	56.7		20.2	60.7	0.218	
+ GA144 measuring 100 ms intervals		2	4.2		1.5	4.5	0.0162	
=1. GA144 total between cycles		**43**	**90.3**		**32.2**	**96.6**	**0.348**	**6,990**
+ All sensors in stby, CC2541 waiting for 3 second BLE poll					45.0	135	0.486	
=2. System between cycles					**77.2**	**232**	**0.834**	**2,910**
+ GA144 polling all sensors 14% duty and lowpass filters for accelerometer				278	38.9	117	0.420	
=3. System with GA144 awareness, all sensors & BLE in standby					**116.1**	**348**	**1.250**	**1,940**
+ GA144 reporting to Host 14% duty				15	2.1	6.30	0.0227	
=4. System with GA144 cheap report					**118.2**	**355**	**1.280**	**1,900**
+ Accelerometer lowest power mode					2.8	8.4	0.0302	
=5. System with Acceleration monitor					**121.0**	**363**	**1.310**	**1,860**
+ Magnetometer		17.2	36.1		12.9	38.6	0.139	
+ Thermopile					180.0	540	1.940	
+ Gyro					4920	14800	53.1	
=7. Sum of high-power sensors					5112	15300	55.2	
=8. System (5) w/high-power sensors					**5233**	**15700**	**56.5**	**43**
+ BLE 8051 update at 10 Hz estimated				>200	>30	>90.0	0.324	
+ BLE Radio poll at 10 Hz estimated				>2000	>20	>60.0	0.216	
=9. Sum of active BLE radio usage				**>2200**	**>50**	**>150.0**	**0.540**	
=10. System w/accel (5) and BLE					**171**	**513**	**1.850**	**1,320**
=11. System w/HP sensors (8) and BLE					**5280**	**15800**	**57.1**	**42.6**

The data indicate that the GA144, with completely idle BLE chip and only accelerometer running in the efficient mode we have chosen, should be able to monitor for simple gestures deducible from that sensor for better than two months on a CR2032 battery; longer with a more efficient overall system design.

Using the BLE radio to report data should reduce this battery life by at least a factor of two. Unfortunately, the test equipment available to us limits our ability to accurately integrate energy in dynamic power sequences when using devices that we cannot control as well as we can our own chips. We have intentionally estimated CC2541 average power on what we believe to be the low side, and hope to refine these data in the future.

The most profound decreases in battery life result when using the higher-power sensors. The magnetometer alone increases system power by ten percent. The thermopile alone more than doubles system power. The gyro is a ravenous power consumer, increasing system power consumption (and decreasing battery life) by a factor on the order of thirty.

Unless significantly more efficient sensors become available, it appears at present that a gesture recognition capability with greatest battery life would consist of GA144 with accelerometer, perhaps magnetometer, and a means of reporting observed gestures that was either very power efficient or could withstand relatively long BLE communication latencies. Before making further assumptions about BLE, we should improve our instrumentation so that we may measure its integrated power more accurately.

4.1 Further Work

Further economies in energy consumption could be had by laying out a new PCB and changing the circuitry. By selecting better sensors that don't stretch the clock, we can eliminate the pull-up resistor from the clock line, and can disable the pull-up on the data line when we are transmitting, thus spending less energy heating those resistors. Finding sensors and radios that would work at 1.8V would reduce energy consumption all around. For the low power sensors we should be able to reduce system power by ten to fifty percent for simple awareness without using the radio.

Experience leads us to anticipate that we could use less energy by implementing that part of the BLE stack we need in polyFORTH and employing a stripped-down BLE radio chip, bringing the radio nearer to the data and eliminating several layers of interface and software as well as finding opportunities to minimize use of the radio while keeping the logical connection alive. We will have a better idea of the costs and benefits of doing so after we have made more precise measurements of the operating cycles of the CC2541.

It would be interesting to add higher-level processing of the sensor data for purposes such as gesture recognition or pedometry. Doing such things continuously with conventional microprocessors consumes prohibitive amounts of energy which precludes long term operation on reasonable batteries. We should be able to do far better than that. The keys here seem to be reducing data at the sensors, minimal reporting or reporting by exception, and learning to obtain the desired results with little or no use of high-powered sensors such as the gyro provided with the ST.

Thus, items that might be on our future to-do list are the following:

- Provision to switch sensor operating mode sets
- Try 6 pF crystal to see if that makes a power difference
- Efficient ambient light sensor
- Demo target reset pull-up and stand-alone op on battery
- Communicate with BLE USB dongle directly in PC using Win32 polyFORTH and HCI protocol.
- Develop BLE stack in GA144 polyFORTH.
- Support for sensors omitted from this project (those requiring more complex reading procedures such as temperature, pressure, humidity.)

GreenArrays®

Application Note AN012 *Revised 6/06/13*

IMPORTANT NOTICE

GreenArrays Incorporated (GAI) reserves the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to GAI's terms and conditions of sale supplied at the time of order acknowledgment.

GAI disclaims any express or implied warranty relating to the sale and/or use of GAI products, including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright, or other intellectual property right.

GAI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using GAI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

GAI does not warrant or represent that any license, either express or implied, is granted under any GAI patent right, copyright, mask work right, or other GAI intellectual property right relating to any combination, machine, or process in which GAI products or services are used. Information published by GAI regarding third-party products or services does not constitute a license from GAI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from GAI under the patents or other intellectual property of GAI.

Reproduction of GAI information in GAI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. GAI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of GAI products or services with statements different from or beyond the parameters stated by GAI for that product or service voids all express and any implied warranties for the associated GAI product or service and is an unfair and deceptive business practice. GAI is not responsible or liable for any such statements.

GAI products are not authorized for use in safety-critical applications (such as life support) where a failure of the GAI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of GAI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by GAI. Further, Buyers must fully indemnify GAI and its representatives against any damages arising out of the use of GAI products in such safety-critical applications.

GAI products are neither designed nor intended for use in military/aerospace applications or environments unless the GAI products are specifically designated by GAI as military-grade or "enhanced plastic." Only products designated by GAI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of GAI products which GAI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

GAI products are neither designed nor intended for use in automotive applications or environments unless the specific GAI products are designated by GAI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, GAI will not be responsible for any failure to meet such requirements.

For current information on GreenArrays products and application solutions, see www.GreenArrayChips.com

Mailing Address: GreenArrays, Inc., 774 Mays Blvd #10 PMB 320, Incline Village, Nevada 89451
Printed in the United States of America
Phone (775) 298-4748 fax (775) 548-8547 email Sales@GreenArrayChips.com