B31DG - Embedded Software Assignment 2

First Program

To design the cyclic executive I modified the example cyclic executive excel spreadsheet to have 5 tasks as well as the correct timings and deadlines, I then filled it out by setting the more frequent task first (1 & 2) as these had less room between the deadlines so were harder to schedule. I then filled out the remaining tasks whilst making sure the slack was roughly the same and not too tight to leave some wriggle room. I then also added an extra column which defined the 2D array of 1 & 0's of which tasks would run in which frames, this made converting it to C really easy. Here is the spread sheet …

	Pi:	4	3	10	10	5			
	Ci:	0.6	0.35	1.5	1.1	0.5			
#frame	time	T1	T2	T3	T4	T5	Total Execution Time (ms)	Slack	{
0	0	1	1	0	0	1	1.45	0.55	{ 1, 1, 0, 0, 1 },
1	2	0	0	1	0	0	1.5	0.5	{ 0, 0, 1, 0, 0 },
2	4	0	1	0	1	0	1.45	0.55	{ 0, 1, 0, 1, 0 },
3	6	1	1	0	0	0	0.95	1.05	{ 1, 1, 0, 0, 0 },
4	8	1	0	0	0	1	1.1	0.9	{ 1, 0, 0, 0, 1 },
5	10	0	1	0	0	1	0.85	1.15	{ 0, 1, 0, 0, 1 },
6	12	1	1	0	0	0	0.95	1.05	{ 1, 1, 0, 0, 0 },
7	14	0	0	1	0	0	1.5	0.5	{ 0, 0, 1, 0, 0 },
8	16	0	1	0	1	0	1.45	0.55	{ 0, 1, 0, 1, 0 },
9	18	1	1	0	0	1	1.45	0.55	{ 1, 1, 0, 0, 1 },
10	20	1	1	0	0	1	1.45	0.55	{ 1, 1, 0, 0, 1 },
11	22	0	0	1	0	0	1.5	0.5	{ 0, 0, 1, 0, 0 },
12	24	1	1	0	0	0	0.95	1.05	{ 1, 1, 0, 0, 0 },
13	26	0	0	0	1	1	1.6	0.4	{ 0, 0, 0, 1, 1 },
14	28	1	1	0	0	0	0.95	1.05	{ 1, 1, 0, 0, 0 },
15	30	0	1	0	0	1	0.85	1.15	{ 0, 1, 0, 0, 1 },
16	32	0	0	1	0	0	1.5	0.5	{ 0, 0, 1, 0, 0 },
17	34	1	1	0	0	0	0.95	1.05	{ 1, 1, 0, 0, 0 },
18	36	0	1	0	1	0	1.45	0.55	{ 0, 1, 0, 1, 0 },
19	38	1	0	0	0	1	1.1	0.9	{ 1, 0, 0, 0, 1 },
20	40	0	1	0	1	0	1.45	0.55	{ 0, 1, 0, 1, 0 },
21	42	1	1	0	0	1	1.45	0.55	{ 1, 1, 0, 0, 1 },
22	44	0	0	1	0	0	1.5	0.5	{ 0, 0, 1, 0, 0 },
23	46	1	1	0	0	0	0.95	1.05	{ 1, 1, 0, 0, 0 },
24	48	0	1	0	0	1	0.85	1.15	{ 0, 1, 0, 0, 1 },
25	50	1	0	0	0	1	1.1	0.9	{ 1, 0, 0, 0, 1 },

26	52	1	1	0	0	0	0.95	1.05	{ 1, 1, 0, 0, 0 },
27	54	0	1	0	1	0	1.45	0.55	{ 0, 1, 0, 1, 0 },
28	56	0	0	1	0	0	1.5	0.5	{ 0, 0, 1, 0, 0 },
29	58	1	1	0	0	1	1.45	0.55	{ 1, 1, 0, 0, 1 },
30	60	0	0	0	0	0	0	2	}

With the lowest slack being 0.4ms this means that if the button pressed (task 7) exceeds 0.4ms the RT requirements will fail, and as doWork takes a average of 0.5ms this will happen in all cases.

```
PERFORMANCE SUMMARY
Start monitoring at 38783
Task #1  0/2500 violations  First job from 39300 to 39920
Task #2  0/3333 violations  First job from 39924 to 40281
Task #3  0/1000 violations  First job from 41272 to 42773
Task #4  0/1000 violations  First job from 43627 to 44828
Task #5  0/2000 violations  First job from 40282 to 40785

abort() was called at PC 0x400d9153 on core 0
```

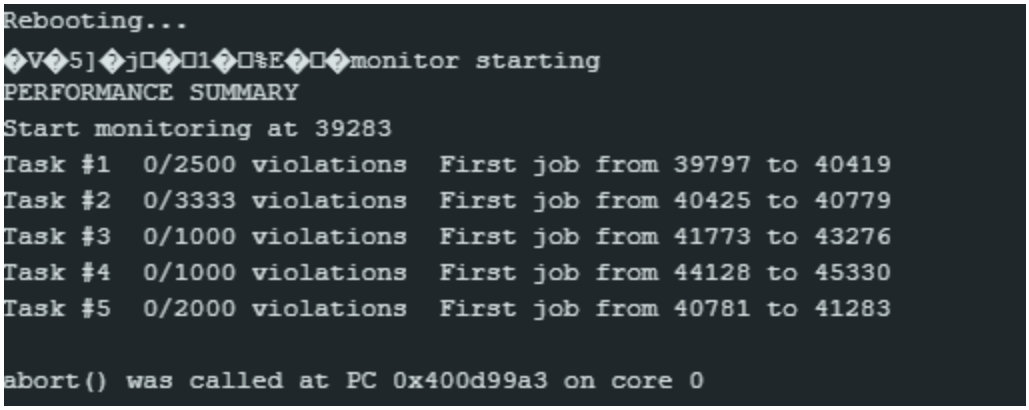
Second Program

The second Program uses FreeRTOS with everything being defined as a task, I worked out the priorities by setting the high frequency task first as there deadlines would have less time in between then so needed to be prioritized so no deadlines were passed. I then fine tuned the priorities slowly to make sure that no violations were happening.

For task 6 I used Mutexes to make sure that task 3 & 4 were finished before trying to set the LED, as i just needed a binary semaphore (yes task finished, no task not finished) mutex were perfect.

The stack size was determined by slowly raising the size until the program was working as expected.

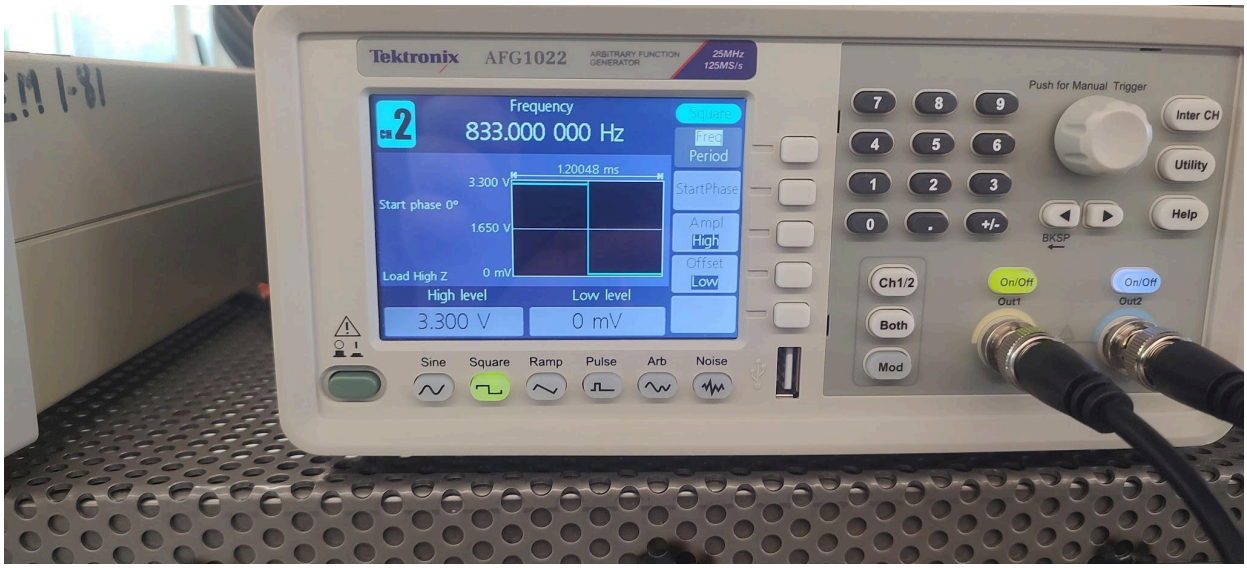
For task 7 the longest time possible before the LED turns on is the same as the longest task (1.5ms) as worst case the button is pressed just after the tasks starts and the task 7 will be next in the queue but need the previous task to finish first before running itself.

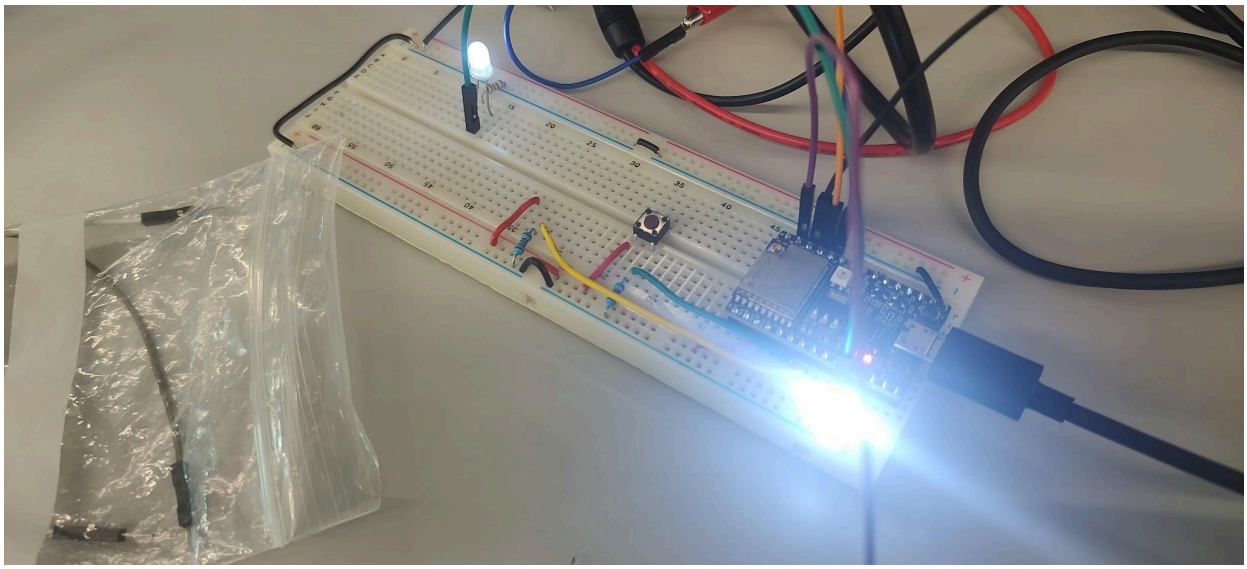


Cyclic Executive VS FreeRTOS

The main difference between the 2 approaches is that in the cyclic executive approach each timing and deadline needs to be calculated before, and if a task runs over, it just runs over there is no error handling / runtime schedule changing, this can be a good thing as it gives the user more control but at the cost of flexibility. Whereas in FreeRTOS the only control of timings you have is setting priorities, this gives little control as FreeRTOS does all the timing behind the scenes, but it is a lot quicker to develop and simpler to create as most of the work is already done for you. There are also optimisations made behind the scenes which can make the program run faster or even on multiple cores, FreeRTOS also supports real time timing changes as new tasks may come up and need to be run e.g. task 7. In summary it's a balance of control vs flexibility, where in a cyclic executive you have complete control but little flexibility and in FreeRTOS you have little control but lots of flexibility.

Results


Tektronix
AFG1022
ARBITRARY FUNCTION GENERATOR
25MHz
125MS/s
Frequency
833.000 000 Hz
Freq
Period
StartPhase
Ampl
High
Offset
Low
1.20048 ms
3.300 V
1.650 V
0 mV
Start phase 0°
Load High Z
High level
Low level
3.300 V
0 mV
Sine
Square
Ramp
Pulse
Arb
Noise
Push for Manual Trigger
Inter CH
Utility
Help
BKSP
Ch1/2
Both
Mod
On/Off
Out1
On/Off
Out2



B31DG - Embedded Software Assignment 2